Little try on g2o_framework's curve_fitting

Author:ChengZhang He ,Date: 2017/6/21

1. Purpose

> Curve_fit is a program in the /g2o/bin when it's built. The program use randomized data set and want to recover the real/designed curve. The g2o developer write this little program to show g2o is a general framework containing general functions.
> The main work of the report is to test the program and change it to more complex situation.
> In brief, I test the program's perform in different Gaussian noise and I try the Gaussian noise in linear and rate. Also, I use two different data set to test it . One is equidistant and another is randomized generated . After that, I add a linear part in it to test it's robust and the stability of the algorithm. After all , I modified the original program to 3 dimension.

2. Introduction

> First , I should introduce the framework we used ———— g2o to prevent the readers confused by my later explanation.
> g2o is a general framework for graph optimization . It use graph to model an optimization problem such that the vertex presents the data we want to optimize and every edge presents the relation between the data and the optimization function .
> We use the graph optimization model because in many industry problems the information matrix is sparse , the optimized value maybe used only in several items in optimized function .
> All we talked above is about non-constraint optimization problem. If the problem is strictly constrained , we should use special add operator on the vertex class.
> The curve fitting is a non-constraint problem.
> The problem is:

> > Giving two parameter vectors (x , y) , minimize the sum of all || y_i - f(x_i) ||^2 and f is controlled by a parameter vector z.
>
> So as I said above the sum of all || y_i - f(x_i) ||^2 is a Unary edge element and parameter vector z is the only vertex to optimize. In our special problem, we use linear solver and dense solver to optimize the problem.
> The original form of f is y = a*e^(- lambda * x) + b.
> I use y = a*e^(- lambda * x) + b + c * x. to check it's stability (By the absolution of c. If abs(c)>0.01 , I think it's unstable.The max Gaussian noise is 0.03 by our test.) And I use z = a*e^(- lambda * (x^{2+y}2)^0.5) + b to model for it's 3 dimension part.

> The origin work use randomized data set and GaoXiang use euidistant data set. Both are used in this report.

3.Detail work

> The first two pictures are rate Gaussian noise by variance of 0.1 and 0.02 in equidistant data set. They are in \Picture\formalized data\rate\

> The second two pictures are linear Gaussian noise by variance of 0.2 and 0.04 in equidistant data set. They are in \Picture\formalized data\linear\

> The third two pictures are rate Gaussian noise by variance of 0.1 and 0.02 in random data set. They are in \Picture\randomized data\rate\

> The forth two pictures are linear Gaussian noise by variance of 0.2 and 0.04 in random data set. They are in \Picture\randomized data\linear\

> These pictures are generated by test1.m and test2.m if you change some of it's comments to code.
> I should make some explanation on why I choose different variance for linear Gaussian noise and rate Gaussian noise. It's because the rate use * and the random y of the data set is around 2.
> For randomized data set and linear Gaussian noise by variance of 0.2 and 0.04,I revise this part:

```
for ( int i=0 ; i < numPoints ; ++i){
double x = g2o::Sampler::uniformRand(0,10);
double y = a*exp(-lambda * x) + b ;
```

```
y+= g2o::Sampler::gaussRand(0,0.2);
//if varaince is 0.04 ,changed it to 0.04!
points[i].x() = x;
points[i].y() = y;
```

> For randomized data set and rate Gaussian noise by variance of 0.1 and 0.02 ,I revise this part:
>
> ```
> for ( int i=0 ; i < numPoints ; ++i){
> double x = g2o::Sampler::uniformRand(0,10);
> double y = (1 + g2o::Sampler::gaussRand(0,0.2)) * (a*exp(-lambda * x) + b) ;
> //if varaince is 0.04 ,changed 0.2 of g2o::Sampler::gaussRand(0,0.2) to 0.04!
> points[i].x() = x;
> points[i].y() = y;
> ```

> For equidistant data set, You only need to change '`double x = g2o::Sampler::uniformRand(0,10);'`

> to '`double x = i*1.0/100;'`
> `For the + c * x form ,you can see it under \Picture\Robus`
> `The three dimension model picture is showed below \Pictrue\3D .You can also use test3.m for better watch.`
>
> And the code is also in my github www.github.com/hechzh/g2o and it's name is the best comment for what it do. Of course, if you want to run them, you need to change their name to Curve_fit and make it.
>
> And a lot of other data and origin program is in the workspace \code.

Conclusion

> The conclusion is that the origin algorithm of Gaussion noise can be up to 0.4 but if we add c * x to it , it only converges when variance < 0.04. It shows although the data set is unstable, the origin algorithm still give some solution similar to real curve. It may confuse people if the Priori knowledge is not exact.
> And next part ,I want to study slam2d.

Refrences

> g2o:A General Framework for Graph optimization --Rainer Kummerle
> <14 Lectrues About SLAM>--Xiang Gao