Robotics 2

Odometry Calibration by Least Square Estimation

Giorgio Grisetti
Kai Arras
Gian Diego Tipaldi
Cyrill Stachniss
Wolfram Burgard

Least Squares Minimization

- The minimization algorithm proceeds by repeatedly performing the following steps:
  - Linearizing the system around the current guess ***x*** and computing the following quantities for each measurement

$$\mathbf{e}_i = \mathbf{e}_i(\mathbf{x}) \qquad \mathbf{A}_i = \left.\frac{\partial \mathbf{e}_i}{\partial \mathbf{x}}\right|_{\mathbf{x}} \qquad (1)$$

  - Computing the terms for the linear system

$$\mathbf{b}^T = \sum_i \mathbf{e}_i^T \mathbf{\Omega}_i \mathbf{A}_i \qquad \mathbf{H} = \sum_i \mathbf{A}_i^T \mathbf{\Omega}_i \mathbf{A}_i \qquad (1)$$

  - Solving the system to get a new optimal increment

$$\mathbf{\Delta x}^* = -\mathbf{H}^{-1}\mathbf{\Delta x}$$

  - Updating the previous estimate

$$\mathbf{x} \leftarrow \mathbf{x} + \mathbf{\Delta x}^*$$

Odometry Calibration

- We have a robot which moves in an environment, gathering the odometry measurements $\boldsymbol{u_i}$,affected by a systematic error.
- For each $\boldsymbol{u_i}$ we have a ground truth $\boldsymbol{u^*_i}$
- There is a function $\boldsymbol{f_i(x)}$ which, given some bias parameters $\boldsymbol{x}$, returns a an unbiased odometry for the reading $\boldsymbol{u_i'}$ as follows

$$\mathbf{u}'_i = f_i(\mathbf{x}) = \begin{pmatrix} x_{11} & x_{12} & x_{13} \\ x_{21} & x_{22} & x_{23} \\ x_{31} & x_{32} & x_{33} \end{pmatrix} \mathbf{u}_i$$

Odometry Calibration (cont)

- The state vector is

$$\mathbf{x} = \begin{pmatrix} x_{11} & x_{12} & x_{13} & x_{21} & x_{22} & x_{23} & x_{31} & x_{32} & x_{33} \end{pmatrix}^T$$

- The error function is

$$\mathbf{e}_i(\mathbf{x}) = \mathbf{u}_i^* - \begin{pmatrix} x_{11} & x_{12} & x_{13} \\ x_{21} & x_{22} & x_{23} \\ x_{31} & x_{32} & x_{33} \end{pmatrix} \mathbf{u}_i$$

- Its derivative is:

$$\mathbf{A}_i = \frac{\partial \mathbf{e}_i(\mathbf{x})}{\partial \mathbf{x}} = - \begin{pmatrix} u_{i,x} & u_{i,y} & u_{i,\theta} & & & & & & \\ & & & u_{i,x} & u_{i,y} & u_{i,\theta} & & & \\ & & & & & & u_{i,x} & u_{i,y} & u_{i,\theta} \end{pmatrix}$$

Exercise

- Write a program to calibrate the odometry
- We provide an input file obtained from a real robot.
- Format of z.dat:
  - Every line is a single odometry measurement
    - $u'_x \; u'_y \; u'_t \; u_x \; u_y \; u_t$
  - **u'** and **u** are respectively the true and the measured odometry of the system in relative coordinates (e.g. motion of the robot between two consecutive frames).

Exercise (in sequential steps)

- Load the measurement matrix
- Write a function ***A***=*v2t(**u**)* that given a transformation expressed as a vector ***u*=[*u*$_x$ *u*$_y$ *u*$_t$]** returns an homogeneous transformation matrix ***A.***
- Write a function ***u=****t2v****(A)*** dual of the previous one.
- Write a function ***T=****compute_odometry_trajectory****(U)*** that computes a trajectory in the global frame by chaining up the measurements (rows) of the Nx3 matrix U. *Hint: use the two functions defined above. Test it on the input data by displaying the trajectories.*
- Define the error function $e_i(X)$ for a line of the measurement matrix. Call it *error_function(**i,X,Z**)*.
- Define the Jacobian function for the measurement ***i*** (call it *jacobian****(i,Z).***
- Write a function ***X=****ls_calibrate_odometry****(Z)*** which constructs and solves the quadratic problem. It should return the calibration parameters ***X.***
- Write a function ***Uprime=****apply_odometry_correction****(X,U)*** which applies the correction to all odometries in the Nx3 matrix U. Test the computed calibration matrix and generate a trajectory.
- Plot the real, the estimated and the corrected odometries.
- In the directory you will find an octave script 'LsOdomCalib' which you can use to test your program.

v2t & t2v

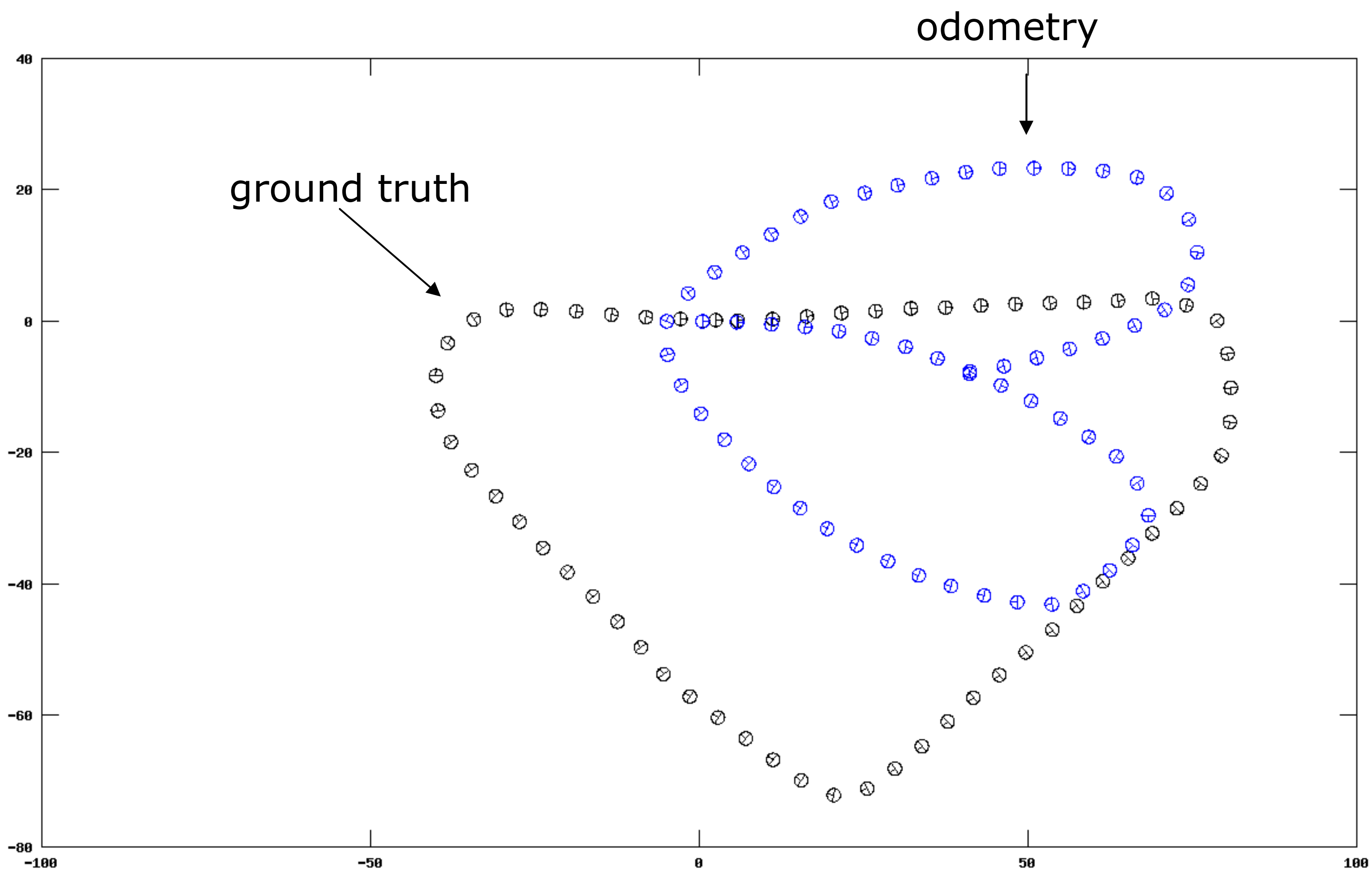
```
function v=t2v(A)
   v(1:2, 1)=A(1:2,3);
   v(3,1)=atan2(A(2,1),A(1,1));
endfunction

function A=v2t(v)
   c=cos(v(3));
   s=sin(v(3));
   A=
   [c, -s, v(1) ;
    s,  c, v(2) ;
    0   0  1   ];
endfunction
```

compute_odometry_trajectory

```
function T=compute_odometry_trajectory(U)
  T=zeros(size(U,1),3);
  P=v2t(zeros(1,3));
  for i=1:size(U,1),
      u=U(i,1:3)';
      P*=v2t(u);
      T(i,1:3)=t2v(P)';
  end
end
```

Trajectories



Error function

```
function e=error_function(i,X,Z)
  uprime=Z(i,1:3)';
  u=Z(i,4:6)';
  e=uprime-X*u;
end
```

Jacobian

```
function A=jacobian(i,Z)
  u=Z(i,4:6);
  A=zeros(3,9);
  A(1,1:3)=-u;
  A(2,4:6)=-u;
  A(3,7:9)=-u;
end
```

Quadratic Solver

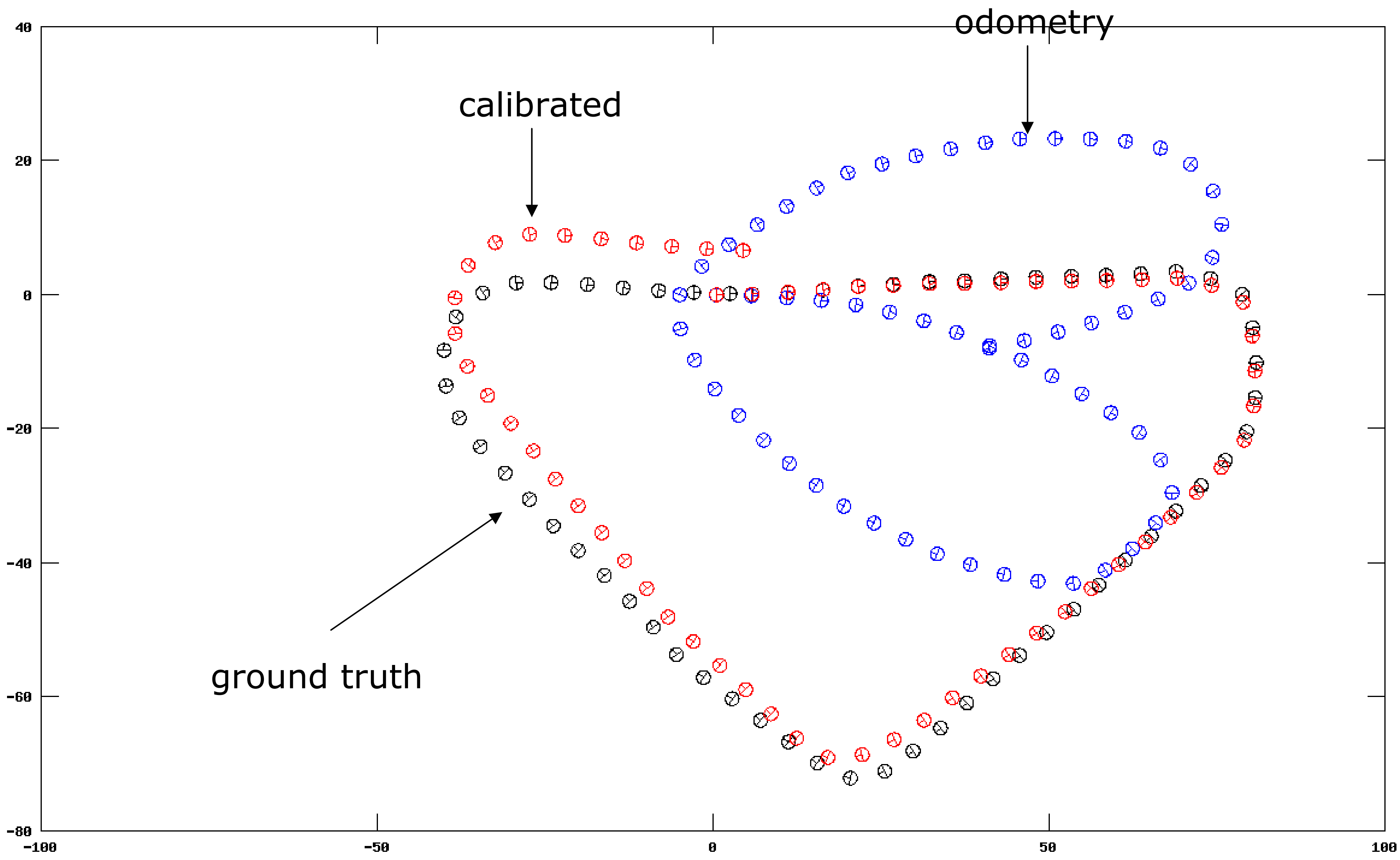
```
function X=ls_calibrate_odometry(Z)
    #accumulator variables for the linear system
    H=zeros(9,9);
    b=zeros(9,1);
    #initial solution (can be anything, se set it to the identity transformation)
    X=eye(3);

    #loop through the measurements and update the
    #accumulators
    for i=1:size(Z,1),
        e=error_function(i,X,Z);
        A=jacobian(i,Z);
        H=H+A'*A;
        b=b+A'*e;
    end
    #solve the linear system
    deltaX=-H\b;
    #this reshapes the 9x1 increment vector in a 3x3 atrix
    dX=reshape(deltaX,3,3)';
    #computes the cumulative solution
    X=X+dX;
end
```

applyOdometryCorrection

```
function
  C=applyOdometryCorrection(bias, U)
  C=zeros(size(U,1),3);
  for i=1:size(U,1),
      u=U(i,1:3)';
      uc=bias*u;
      C(i,:)=uc';
  end
endfunction
```

Plots


odometry
calibrated
ground truth
40
20
0
-20
-40
-60
-80
-100
-50
0
50
100

Questions ☺

- Did you find this practical useful?
- Would you feel confident to apply least squares to more complex problems?
- Which one of the wheels of the robot was deflated (right or left)?

For the next lecture you must have understood the basic concepts on least square minimization: we will do SLAM.