# Probabilistic Robotics Course

# Least Squares application: Odometry Calibration

**Omar Salem**
salem@diag.uniroma1.it

Dept of Computer Control and Management Engineering
Sapienza University of Rome

# Algorithm (one Iteration)

Clear **H** and **b**, aka assign the correct dimensions

$$\mathbf{H} \leftarrow 0 \qquad \mathbf{b} \leftarrow 0$$

For each measurement, update **H** and **b**

$$\mathbf{e}^{[i]} \leftarrow \mathbf{h}^{[i]}(\mathbf{x}^*) - \mathbf{z}^{[i]}$$

Compute **error**

$$\mathbf{J}^{[i]} \leftarrow \left.\frac{\partial \mathbf{e}^{[i]}(\mathbf{x})}{\partial \mathbf{x}}\right|_{\mathbf{x}=\mathbf{x}^*}$$

Compute **Jacobian**

$$\mathbf{H} \leftarrow \mathbf{H} + \mathbf{J}^{[i]T}\boldsymbol{\Omega}^{[i]}\mathbf{J}^{[i]}$$

$$\mathbf{b} \leftarrow \mathbf{b} + \mathbf{J}^{[i]T}\boldsymbol{\Omega}^{[i]}\mathbf{e}^{[i]}$$

Update the estimate with the perturbation

$$\boldsymbol{\Delta}\mathbf{x} \leftarrow \mathrm{solve}(\mathbf{H}\boldsymbol{\Delta}\mathbf{x} = -\mathbf{b})$$

Solve linear system

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

# Methodology

Identify the state space **X**

- Qualify the domain
- Find a locally Euclidean parameterization

Identify the measurement space(s) **Z**

- Qualify the domain
- Find a locally Euclidean parameterization

Identify the prediction functions **h(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 $\boldsymbol{u}^*_i$ provided us by an external sensor.
- There is a function $\boldsymbol{f_i(x)}$ which, given some bias parameters $\boldsymbol{x}$, returns 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

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

- Its derivative is

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

# Exercise

- Write a program to calibrate the odometry
- We provide an input file obtained from a real robot.
- Format of dataset:
  - 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).

# 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.

# T2v && v2t

```
#computes the homogeneous transform matrix A of the pose vector v
function A = v2t(v)
    c = cos(v(3));
    s = sin(v(3));

    A=[c, -s, v(1) ;
       s,  c, v(2) ;
       0,  0,  1   ];
end



#computes the pose vector v from an homogeneous transform A
function v = t2v(A)
    v(1:2,1) = A(1:2,3);
    v(3,1) = %TODO;
end
```

Recover *theta* from the rotation matrix

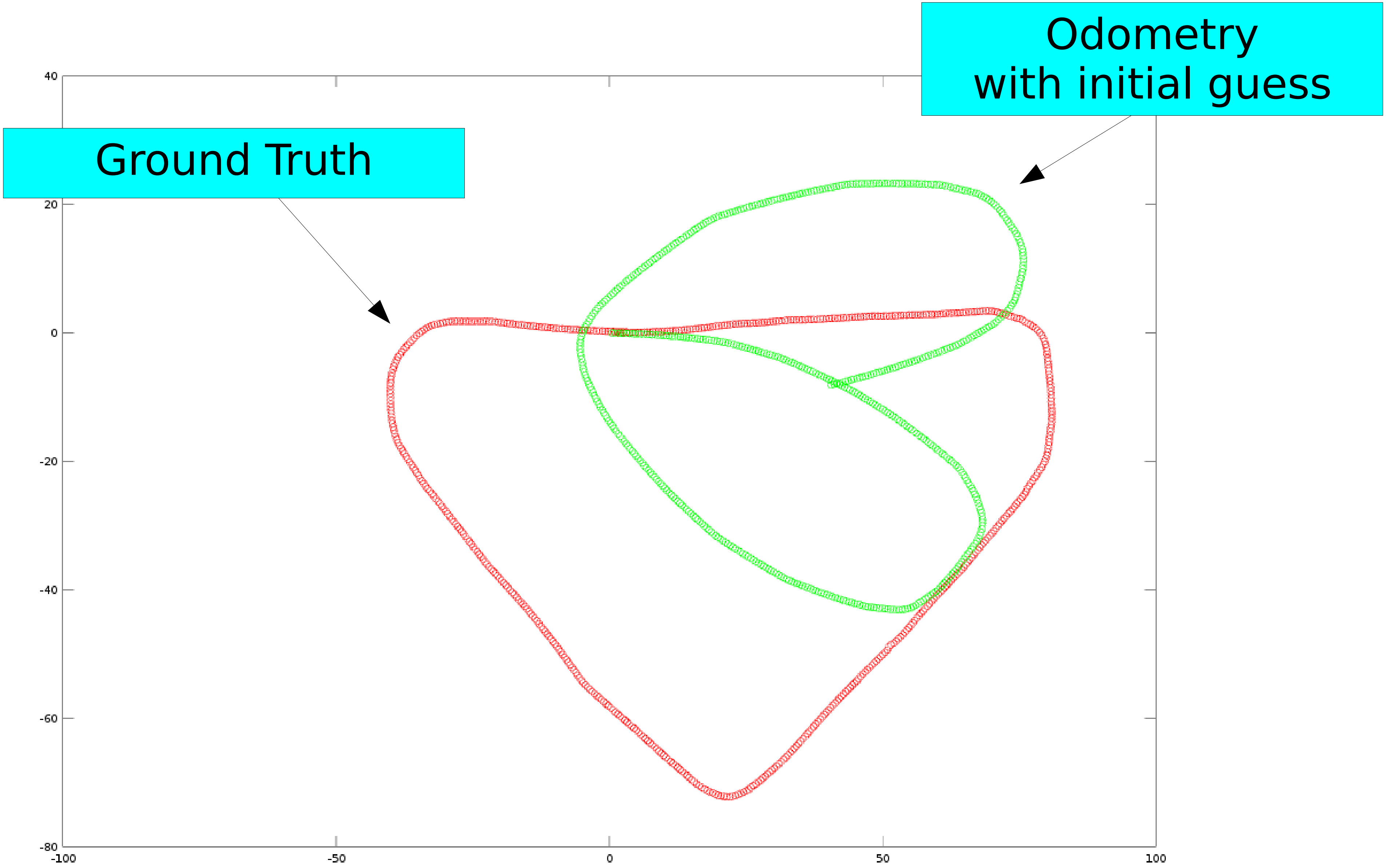
# compute_odometry_trajectory

```
#computes the trajectory of the robot by chaining up
#the incremental movements of the odometry vector
#U: a Nx3 matrix, each row contains the odometry ux, uy utheta
#T: a Nx3 matrix, each row contains the robot position (starting from 0,0,0)
function T = compute_odometry_trajectory(U)
    T = zeros(size(U,1),3);
    P = eye(3);

    for i = 1:size(U,1)
        u = U(i,1:3)';
        P *= %TODO;
        T(i,1:3) = t2v(P)';
    end
end
```

Update robot pose by chaining the relative transformations

# Trajectories



# error_function

```
#this function computes the error of the i^th measurement in Z
#given the calibration parameters
#i: the number of the measurement
#X: the actual calibration parameters
#Z: the measurement matrix
#e: the error of the ith measurement
function e = error_function(i,X,Z)
    uprime = Z(i,1:3)';
    u = Z(i,4:6)';
    e = %TODO;
end
```

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

# jacobian

```
#derivative of the error function for the ith measurement in Z
#does not depend on the state
#i: the measuement number
#Z: the measurement matrix
#A: the jacobian of the ith measurement
function A = jacobian(i,Z)
    u = Z(i,4:6);
    A = zeros(3,9);
    A(1,1:3) = %TODO;
    A(2,4:6) = %TODO;
    A(3,7:9) = %TODO;
end
```

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

# Quadratic Solver

```
#this function solves the odometry calibration problem
#given a measurement matrix Z.
#Every row of the matrix contains
#z_i = [u'x, u'y, u'theta, ux, uy, ytheta]
#Z: The measurement matrix
#X: the calubration matrix
#returns the bias correction matrix BIAS
function X = ls_calibrate_odometry(Z)
    #accumulator variables for the linear system
    H = zeros(%TODO,%TODO);
    b = zeros(%TODO,%TODO);
    #initial solution (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 = %TODO;
        b = %TODO;
    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
```

$$\mathbf{H} \leftarrow \mathbf{H} + \mathbf{J}^{[i]T}\boldsymbol{\Omega}^{[i]}\mathbf{J}^{[i]}$$

$$\mathbf{b} \leftarrow \mathbf{b} + \mathbf{J}^{[i]T}\boldsymbol{\Omega}^{[i]}\mathbf{e}^{[i]}$$

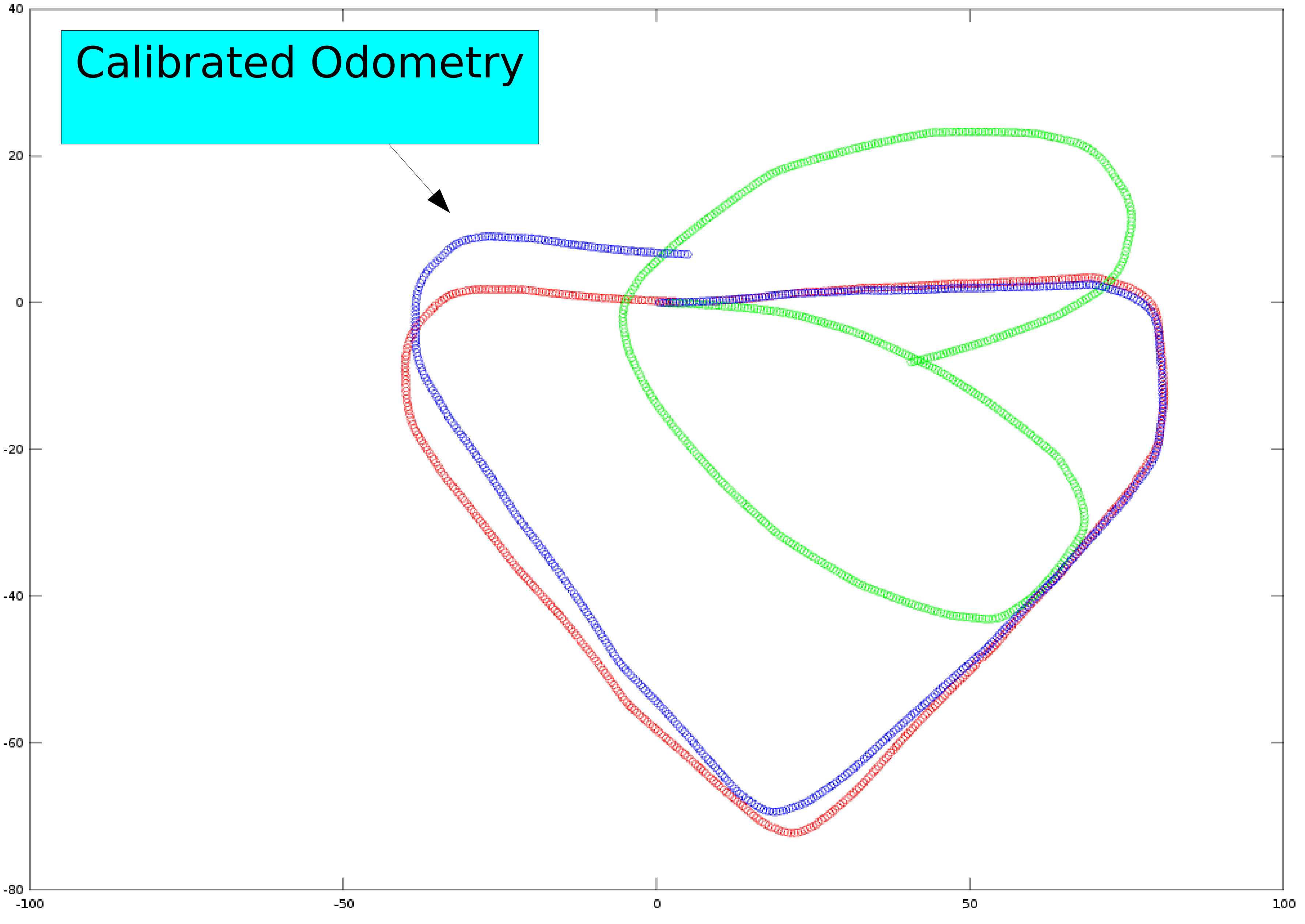
# applyOdometryCorrection

```
#computes a calibrated vector of odometry measurements
#by applying the bias term to each line of the measurements
#X:    3x3 matrix obtained by the calibration process
#U:    Nx3 matrix containing the odometry measurements
#C: Nx3 matrix containing the corrected odometry measurements

function C = apply_odometry_correction(X, U)
    C = zeros(size(U,1),3);
    for i = 1:size(U,1),
        u = U(i,1:3)';
        uc = %TODO;
        C(i,:) = uc;
    end
end
```

Apply calibration correction to each odometry measurement

# Plot

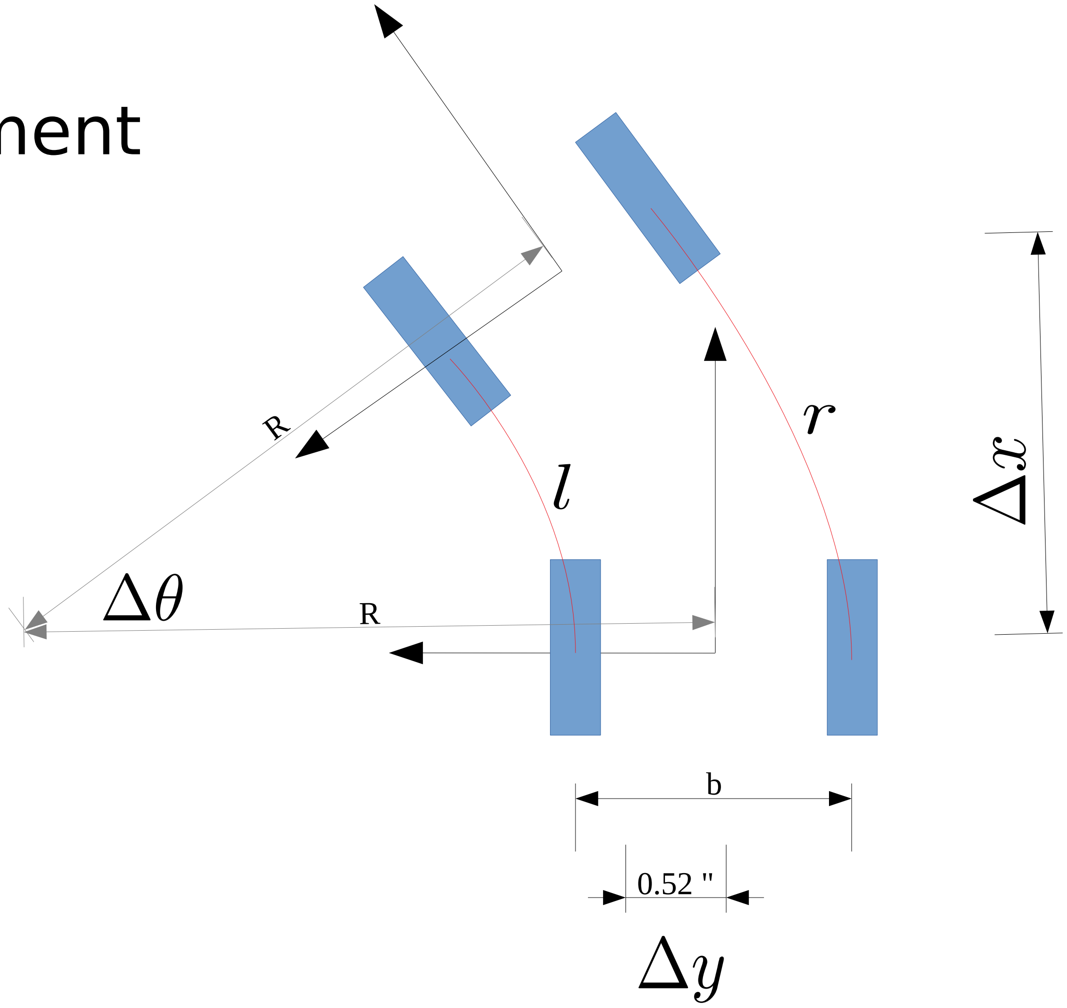


# Calibrating a Unicycle

# Differential Drive (Unicycle)

- r, l: motion on the ground wheels
- R: radius of curvature
- Dx, Dy, Dtheta: displacement from previous origin

$$
\begin{aligned}
r &= \Delta\theta \cdot (R + b/2) \\
l &= \Delta\theta \cdot (R - b/2) \\
\Delta\theta &= \frac{r - l}{b} \\
R &= \frac{r + l}{2\Delta\theta} \\
\Delta x &= R \sin(\Delta\theta) \\
\Delta y &= R(1 - \cos(\Delta\theta))
\end{aligned}
$$



# Differential Drive (Unicycle)

Exact Integration, at time [i] with constant velocity in the interval

encoder to meters factor

right wheel motion $r^{[i]} = k_r t_r^{[i]}$ right encoder ticks

left wheel motion $l^{[i]} = k_l t_l^{[i]}$ left encoder ticks

angular displacement $\Delta\theta^{[i]} = \dfrac{r^{[i]} - l^{[i]}}{b}$ baseline

x displacement $\Delta x^{[i]} = \dfrac{r^{[i]} + l^{[i]}}{2} \dfrac{\sin \Delta\theta^{[i]}}{\Delta\theta^{[i]}}$

y displacement $\Delta y^{[i]} = \dfrac{r^{[i]} + l^{[i]}}{2} \dfrac{1 - \cos \Delta\theta^{[i]}}{\Delta\theta^{[i]}}$

# Differential Drive (Unicycle)

Angles usually small, remove singularity around Delta theta=0 by Taylor expansion (up to 4th order)

$$\frac{\sin\theta}{\theta} \simeq \overbrace{1 - \frac{\theta^2}{6} + \frac{\theta^4}{120} - \frac{\theta^6}{5040}}^{P_x(\theta)}$$

$$\frac{1-\cos\theta}{\theta} \simeq \overbrace{\frac{\theta}{2} - \frac{\theta^3}{24} + \frac{\theta^5}{720}}^{P_y(\theta)}$$

# Differential Drive

We can rewrite the stuff as

$$
\begin{aligned}
\Delta_+^{[i]} &= r^{[i]} + l^{[i]} \\
\Delta_-^{[i]} &= r^{[i]} - l^{[i]} \\
\Delta\theta^{[i]} &= \frac{\Delta_-^{[i]}}{b} \\
\Delta x^{[i]} &= \frac{\Delta_+^{[i]}}{2} P_x(\Delta\theta^{[i]}) \\
\Delta y^{[i]} &= \frac{\Delta_+^{[i]}}{2} P_y(\Delta\theta^{[i]})
\end{aligned}
$$

# Calibrating a Unicycle

Given:

- encoder measures (relative): $t_r^{[i]},\ t_l^{[i]}$
- position of the robot on the plane, from external measurement system: $\Delta x^{[i]},\ \Delta y^{[i]},\ \Delta \theta^{[i]}$
- nominal value of parameters

$$k_l = -1,\ k_r = 1,\ b = 0.3$$

- The data are provided in a text file (matrix), where each row [i] is

$$t_l^{[i]},\ t_r^{[i]}\ \Delta x^{[i]},\ \Delta y^{[i]},\ \Delta \theta^{[i]},$$

Requested:

- Kinematic parameters: $k_r, k_l, b$

# Do it yourself

- Identify the state space **X**
  - Qualify the domain
  - Find a locally Euclidean parameterization

Identify the measurement space(s) **Z**

  - Qualify the domain
  - Find a locally Euclidean parameterization

Identify the prediction functions **h(x)**
(done by me in the previous slides :)

# Hints:

- Check the observation function by displaying the trajectory from a sequence of increments.
- In the estimation process, skip increments that are too small. They don't bring any information to the estimate.
- At first use a numeric Jacobian.