

Probabilistic Robotics Course

Least Squares application: Odometry Calibration

G. Grisetti, B. Della Corte

`{grisetti,dellacorte}@dis.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

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

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

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

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

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

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

Update the estimate with the perturbation

$$\Delta \mathbf{x} \leftarrow \text{solve}(\mathbf{H} \Delta \mathbf{x} = -\mathbf{b}) \quad \leftarrow \text{Solve linear system}$$

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

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 \mathbf{u}_i , affected by a systematic error.
- For each \mathbf{u}_i we have a ground truth \mathbf{u}_i^* provided us by an external sensor.
- There is a function $\mathbf{f}_i(\mathbf{x})$ which, given some bias parameters \mathbf{x} , returns an unbiased odometry for the reading \mathbf{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

$$\mathbf{x} = (x_{11} \quad x_{12} \quad x_{13} \quad x_{21} \quad x_{22} \quad x_{23} \quad x_{31} \quad x_{32} \quad x_{33})^T$$

- 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
 - $u_x^* \ u_y^* \ u_t^* \ u_x \ u_y \ u_t$
 - \mathbf{u}^* and \mathbf{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 $\mathbf{A} = v2t(\mathbf{u})$ that given a transformation expressed as a vector $\mathbf{u} = [u_x \ u_y \ u_t]$ returns an homogeneous transformation matrix \mathbf{A} .
- Write a function $\mathbf{u} = t2v(\mathbf{A})$ dual of the previous one.
- Write a function $\mathbf{T} = \text{compute_odometry_trajectory}(\mathbf{U})$ 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 $\mathbf{e}_i(\mathbf{X})$ for a line of the measurement matrix. Call it *error_function(i,X,Z)*.
- Define the Jacobian function for the measurement \mathbf{i} (call it *jacobian(i,Z)*).
- Write a function $\mathbf{X} = \text{ls_calibrate_odometry}(\mathbf{Z})$ which constructs and solves the quadratic problem. It should return the calibration parameters \mathbf{X} .
- Write a function $\mathbf{Uprime} = \text{apply_odometry_correction}(\mathbf{X}, \mathbf{U})$ 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

```
1 #computes the homogeneous transform matrix A of the pose vector v
2 function A = v2t(v)
3     c = cos(v(3));
4     s = sin(v(3));
5
6     A=[c, -s, v(1) ;
7        s,  c, v(2) ;
8        0,  0,  1   ];
9 end
10
11
12
13 #computes the pose vector v from an homogeneous transform A
14 function v = t2v(A)
15     v(1:2,1) = A(1:2,3);
16     v(3,1) = %TODO;
17 end
```

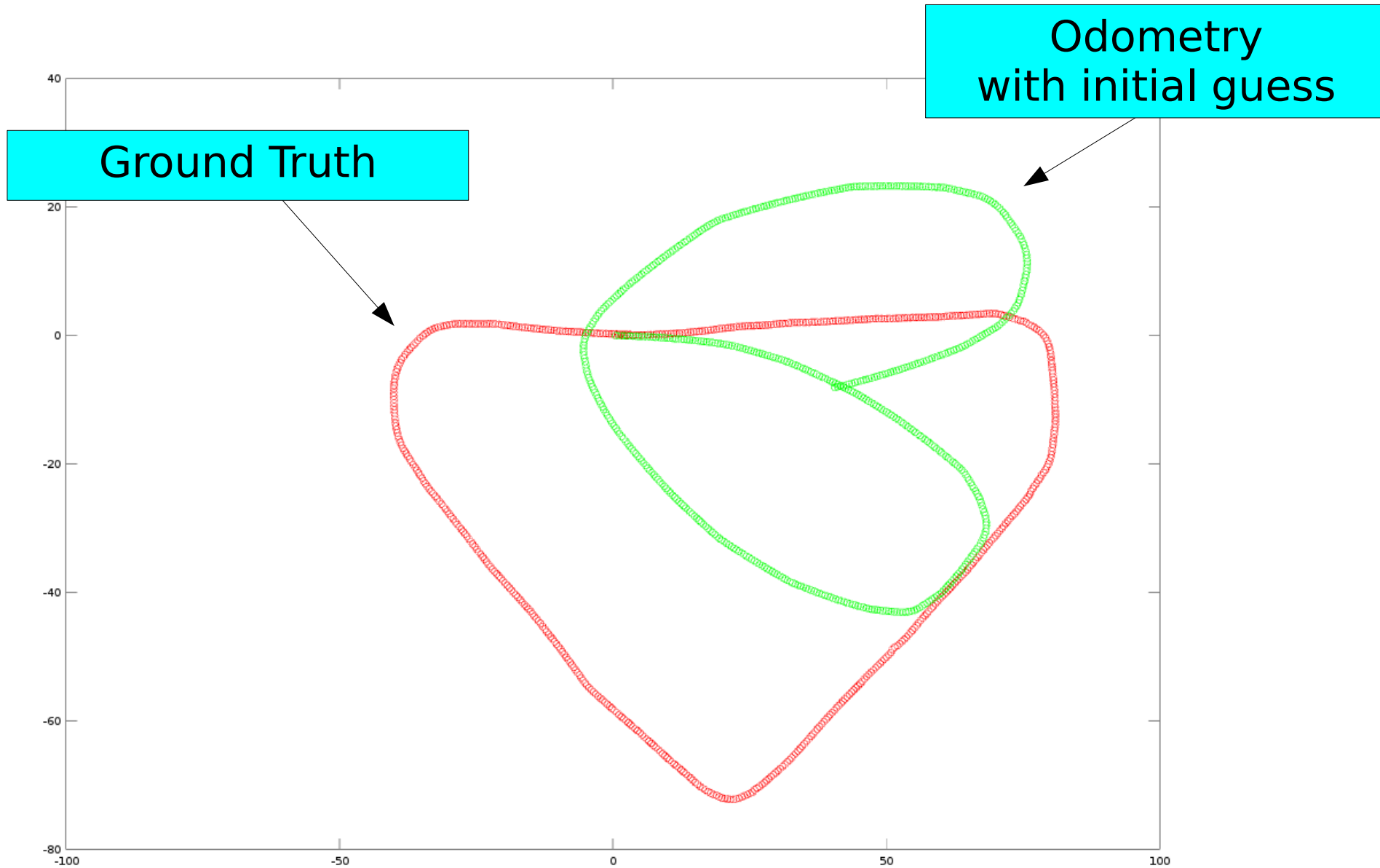
Recover *theta* from
the rotation matrix

compute_odometry_trajectory

```
1 #computes the trajectory of the robot by chaining up
2 #the incremental movements of the odometry vector
3 #U: a Nx3 matrix, each row contains the odometry ux, uy utheta
4 #T: a Nx3 matrix, each row contains the robot position (starting
   from 0,0,0)
5 function T = compute_odometry_trajectory(U)
6     T = zeros(size(U,1),3);
7     P = eye(3);
8
9     for i = 1:size(U,1)
10         u = U(i,1:3)';
11         P *= %TODO;
12         T(i,1:3) = t2v(P)';
13     end
14 end
```

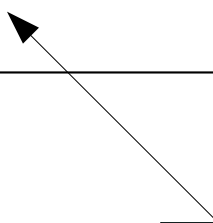
Update robot pose by
chaining the relative
transformations

Trajectories




error_function

```
1 #this function computes the error of the i^th measurement in Z
2 #given the calibration parameters
3 #i: the number of the measurement
4 #X: the actual calibration parameters
5 #Z: the measurement matrix
6 #e: the error of the ith measurement
7 function e = error_function(i,X,Z)
8     uprime = Z(i,1:3)';
9     u = Z(i,4:6)';
10    e = %TODO;
11 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

```
1 #derivative of the error function for the ith measurement in Z
2 #does not depend on the state
3 #i: the measuement number
4 #Z: the measurement matrix
5 #A: the jacobian of the ith measurement
6 function A = jacobian(i,Z)
7     u = Z(i,4:6);
8     A = zeros(3,9);
9     A(1,1:3) = %TODO;
10    A(2,4:6) = %TODO;
11    A(3,7:9) = %TODO;
12 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

```
1 #this function solves the odometry calibration problem
2 #given a measurement matrix Z.
3 #Every row of the matrix contains
4 #z_i = [u'x, u'y, u'theta, ux, uy, ytheta]
5 #Z: The measurement matrix
6 #X: the calibration matrix
7 #returns the bias correction matrix BIAS
8 function X = ls_calibrate_odometry(Z)
9     #accumulator variables for the linear system
10    H = zeros(%TODO,%TODO);
11    b = zeros(%TODO,%TODO);
12    #initial solution (the identity transformation)
13    X = eye(3);
14
15    #loop through the measurements and update the
16    #accumulators
17    for i = 1:size(Z,1),
18        e = error_function(i,X,Z);
19        A = jacobian(i,Z);
20        H = %TODO;
21        b = %TODO;
22    end
23
24    #solve the linear system
25    deltaX = -H\b;
26    #this reshapes the 9x1 increment vector in a 3x3 atrix
27    dX = reshape(deltaX,3,3)';
28    #computes the cumulative solution
29    X = X+dX;
30 end
```

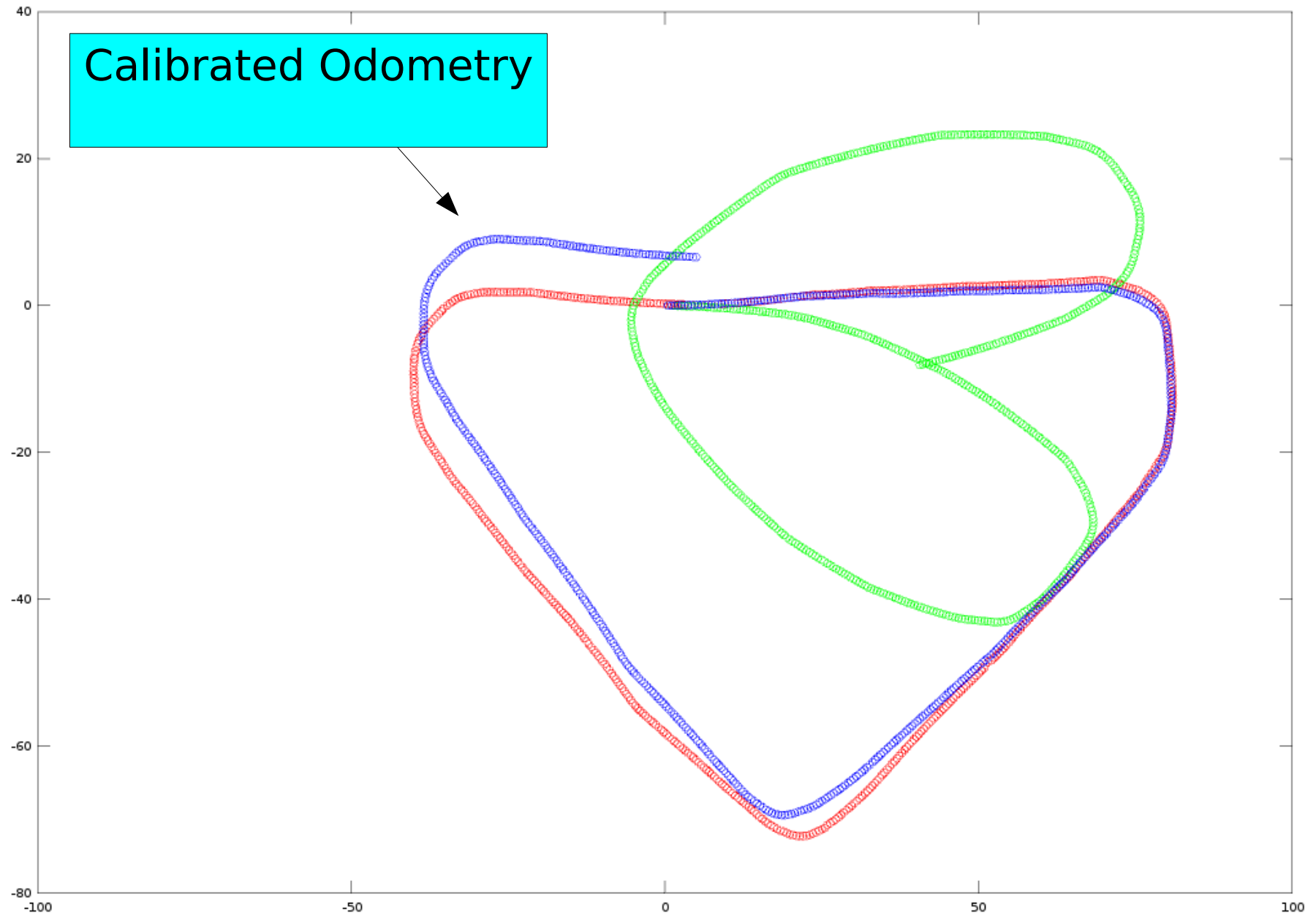
$$\begin{aligned} \mathbf{H} &\leftarrow \mathbf{H} + \mathbf{J}^{[i]T} \boldsymbol{\Omega}^{[i]} \mathbf{J}^{[i]} \\ \mathbf{b} &\leftarrow \mathbf{b} + \mathbf{J}^{[i]T} \boldsymbol{\Omega}^{[i]} \mathbf{e}^{[i]} \end{aligned}$$

applyOdometryCorrection

```
1 #computes a calibrated vector of odometry measurements
2 #by applying the bias term to each line of the measurements
3 #X: 3x3 matrix obtained by the calibration process
4 #U: Nx3 matrix containing the odometry measurements
5 #C: Nx3 matrix containing the corrected odometry measurements
6
7 function C = apply_odometry_correction(X, U)
8     C = zeros(size(U,1),3);
9     for i = 1:size(U,1),
10         u = U(i,1:3)';
11         uc = %TODO;
12         C(i,:) = uc;
13     end
14 end
```

Apply calibration correction to each odometry measurement

Plot



Part 2

Calibrating an Unicycle

Differential Drive (Unicycle)

- r, l : motion on the ground wheels
- R : radius of curvature
- $\Delta x, \Delta y, \Delta\theta$: displacement from previous origin

$$r = \Delta\theta \cdot (R + b/2)$$

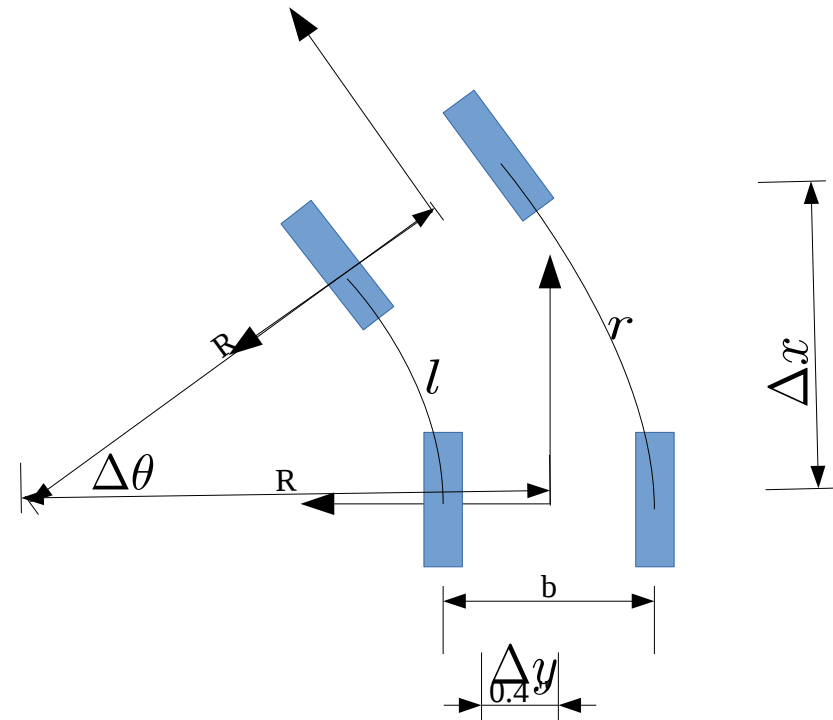
$$l = \Delta\theta \cdot (R - b/2)$$

$$\Delta\theta = \frac{r - l}{b}$$

$$R = \frac{r + l}{2\Delta\theta}$$

$$\Delta x = R \sin(\Delta\theta)$$

$$\Delta y = R(1 - \cos(\Delta\theta))$$



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

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

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

Differential Drive (Unicycle)

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

$$\begin{aligned}\frac{\sin \theta}{\theta} &\simeq \overbrace{1 - \frac{\theta^2}{6} + \frac{\theta^4}{120} - \frac{\theta^6}{5040}}^{P_x(\theta)} \\ \frac{1 - \cos \theta}{\theta} &\simeq \overbrace{\frac{\theta}{2} - \frac{\theta^3}{24} + \frac{\theta^5}{720}}^{P_y(\theta)}\end{aligned}$$

Differential Drive

We can rewrite the stuff as

$$\Delta_{+}^{[i]} = r^{[i]} + l^{[i]}$$

$$\Delta_{-}^{[i]} = r^{[i]} - l^{[i]}$$

$$\Delta\theta^{[i]} = \frac{\Delta_{-}^{[i]}}{b}$$

$$\Delta x^{[i]} = \frac{\Delta_{+}^{[i]}}{2} P_x(\Delta\theta^{[i]})$$

$$\Delta y^{[i]} = \frac{\Delta_{+}^{[i]}}{2} P_y(\Delta\theta^{[i]})$$

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 **$h(\mathbf{x})$**
(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.