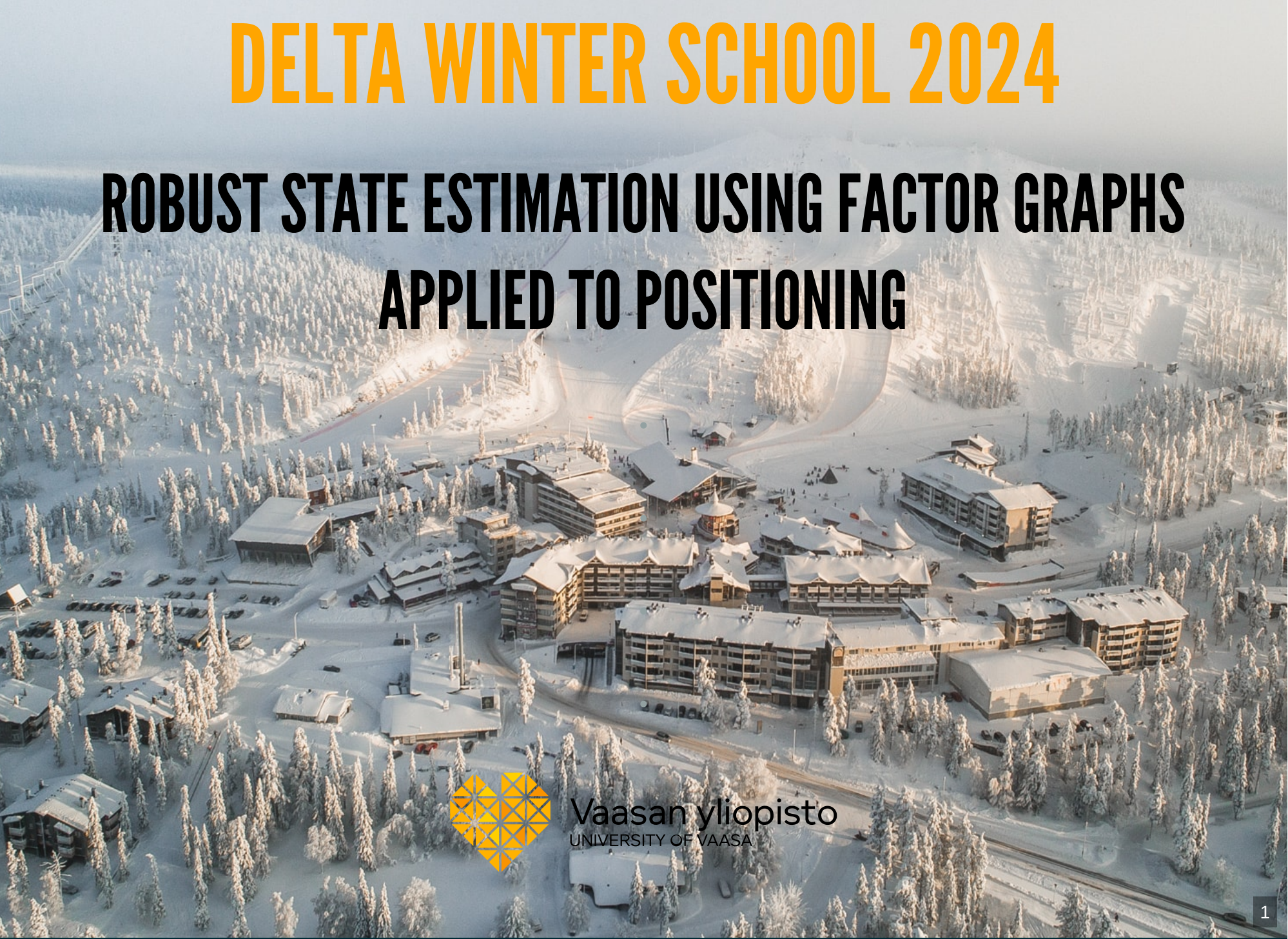
DELTA WINTER SCHOOL 2024

ROBUST STATE ESTIMATION USING FACTOR GRAPHS APPLIED TO POSITIONING



Vaasan yliopisto
UNIVERSITY OF VAASA

CONTENTS

- State estimation applications
- Methods and robustness
- Factor graph
- Case example

State observer or state estimator is a system that provides an estimate of the internal state of a given real system, from measurements of the input and output of the real system.

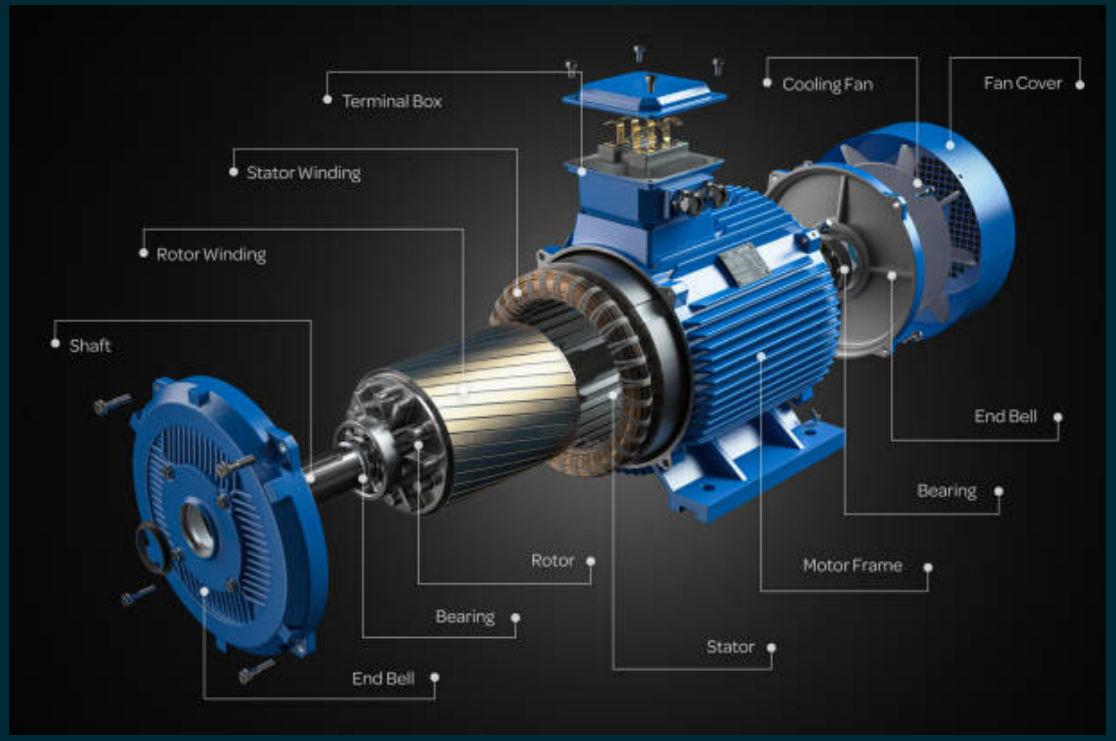
Linear system x Non-linear system

$$x_{k+1} = \mathbf{A}x_k + \mathbf{B}u_k$$
$$y_k = \mathbf{C}x_k + \mathbf{D}u_k$$

$$x_{k+1} = f(x_k) +$$

is the state, y_k is the measurement (observation) and u is the input. $f()$ and $h()$ are dynamic and observation functions.

INDUCTION MOTOR STATE ESTIMATION



- Inputs
  - Current
  - Voltage
- Outputs
  - Torque
  - Speed
  - Temperature
  - Noise and vibration
  - Current
  - Magnetic field

INDUCTION MOTOR STATE ESTIMATION

Typical uses for sate estimation:

1. Fault diagnostics (FD)
    - Bearings, Windings, Rotor, Shaft, Stator, Power electronics, Gears
2. Condition monitoring (CM)
3. Performance optimization (PO)

R. R. Kumar, M. Andriollo, G. Cirrincione, M. Cirrincione, and A. Tortella, “A Comprehensive Review of Conventional and Intelligence-Based Approaches for the Fault Diagnosis and Condition Monitoring of Induction Motors,” Energies, vol. 15, no. 23, Art. no. 23, Jan. 2022, doi: 10.3390/en15238938.

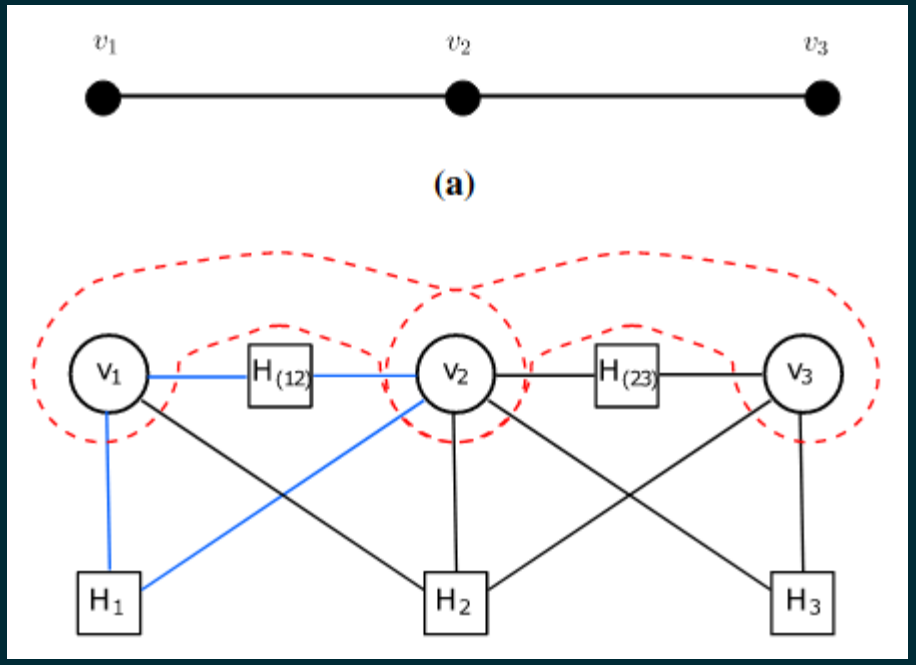
DIESEL ENGINE MONITORING AND CONTROL

- Observe noise and vibrations, cylincder pressure, injector current and rail pressure, etc.
- Estimate combustion noise, mechanical noise, injector noise,
- Fault diagnostics
- Combustion noise control
- Tuning the engine for new fuels, like hydrogen and ammonia

SMART GRIDS

- The supervisory control and protection requires reliable state estimation.
- Sate estimation is challenged by noise, measurement sparsity, sensor errors, and cyber attacks.
- The purpose of the robust state estimation is to estimate the value of the needed variables with high confidence, despite noise and outliers.
- Provides also resilience over cyber attacks

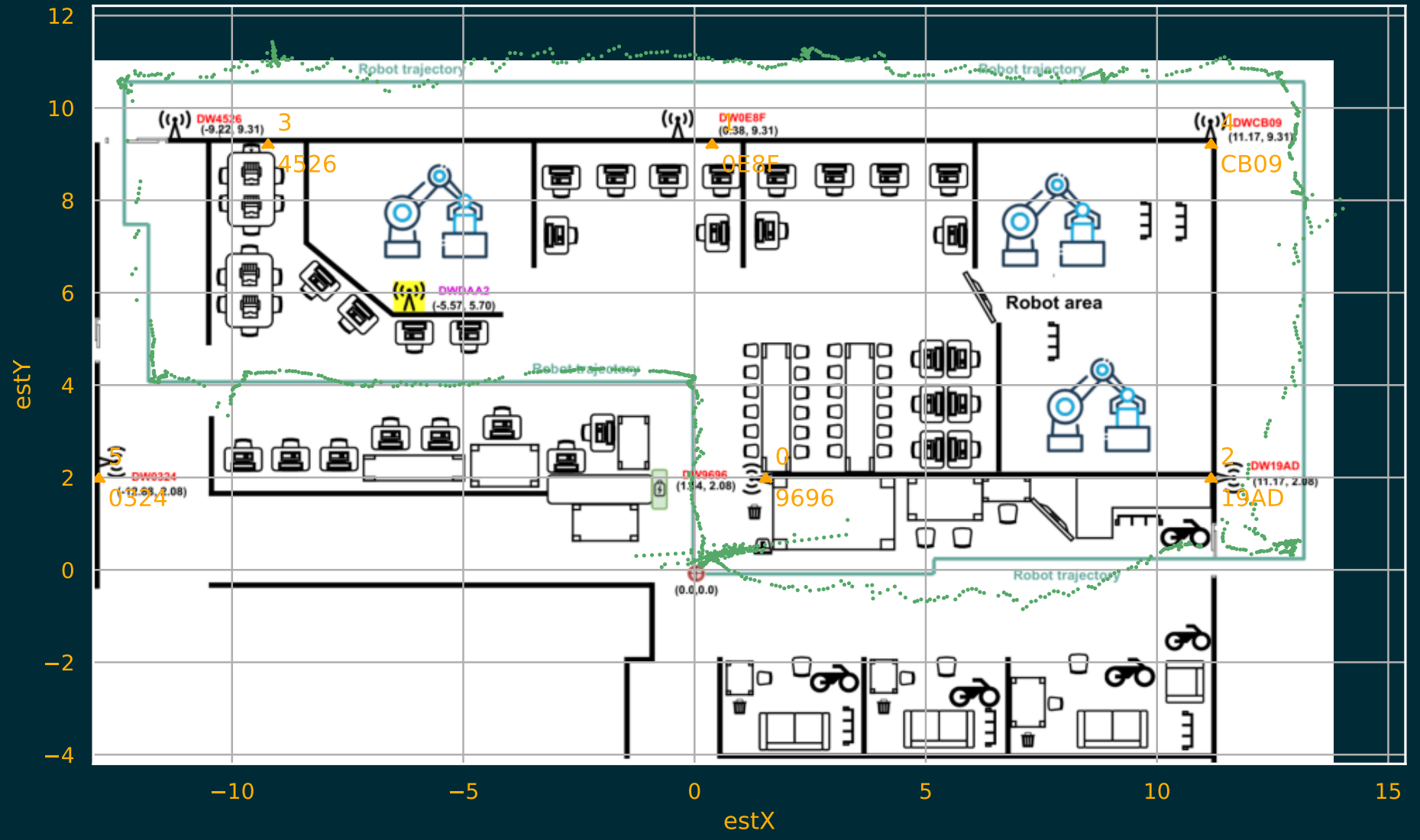
SMART GRID STATE ESTIMATION



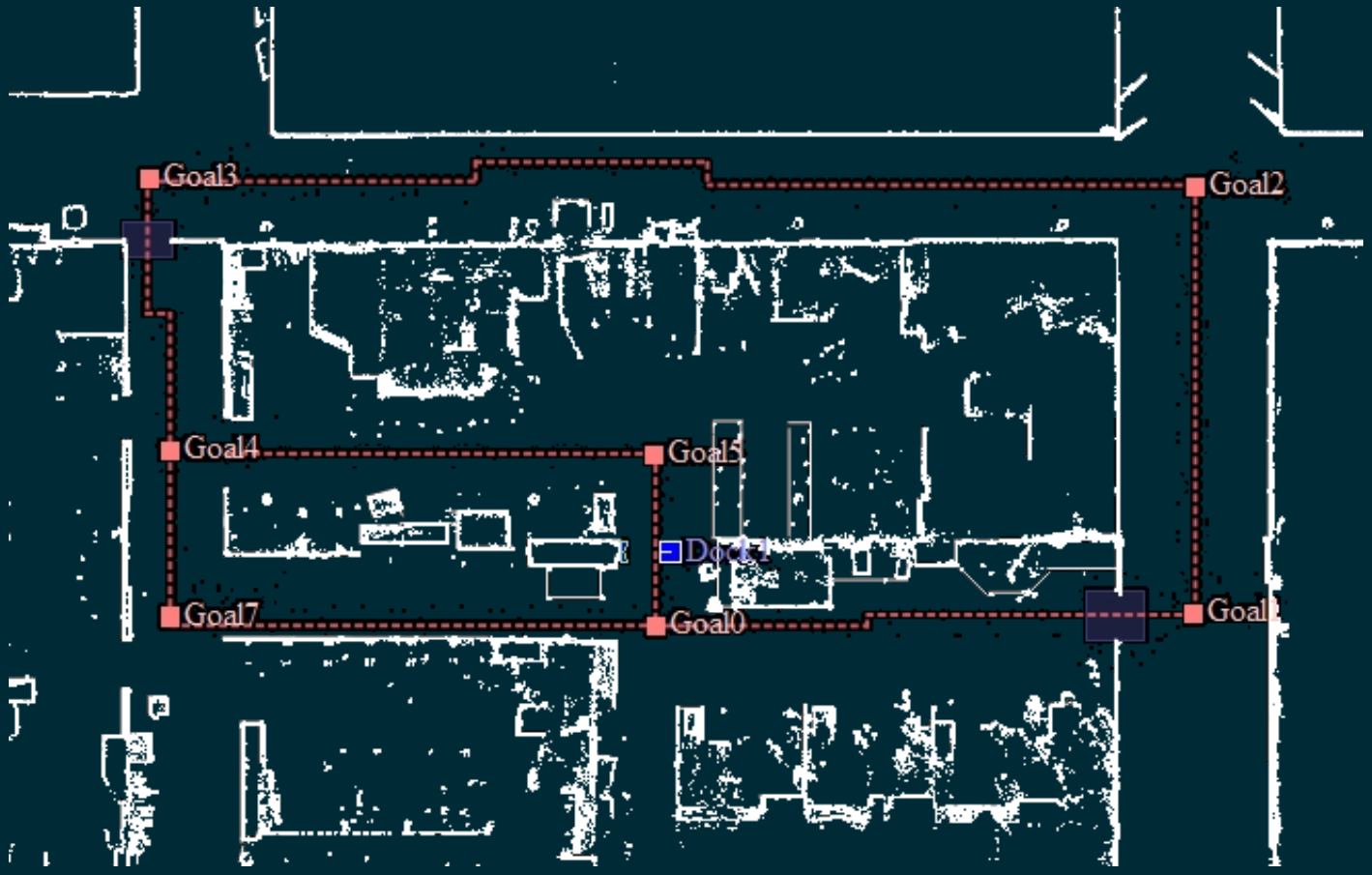
- State variables
  - Voltages, v_i
  - Currents, i_i
  - Phases, ϕ_i
  - active and reactive power flows, , p_i q_i
- Status information
  - Switch status

T. Ritmeester and H. Meyer-Ortmanns, “Belief propagation for supply networks: efficient clustering of their factor graphs,” Eur. Phys. J. B, vol. 95, no. 5, p. 89, May 2022, doi: 10.1140/epjb/s10051-022-00336-7.

ROBOT POSITIONING



HOW ROBOT SEES THE LAB WITH LIDAR USING SLAM



POSITION OBSERVABLES

- Ranges from the radio based positioning systems
- Signal strength or channel status information
- Acceleration from IMU
- Odometry readings
- Images
- Lidar / Radar ranges

Goals:

- Estimate the precise position of the mobile robot, car, pedestrian or other item

STATE ESTIMATION METHODS

LMS SOLUTION

TOA solution can be found with the following way:

$$k_i = x_i^2 + y_i^2 + z_i^2, (i = 1, 2, \ldots, n$$

Solution is

$$\mathbf{t} = \frac{1}{2}(\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T\mathbf{b}$$

Weighted solution:

$$\mathbf{t} = \frac{1}{2}(\mathbf{A}^T\mathbf{W}\mathbf{A})^{-1}\mathbf{A}^T\mathbf{W}\mathbf{b}$$

$$\mathbf{A} = \begin{bmatrix} x_2 - x_1 & y_2 - y_1 & z_2 - z \\ x_3 - x_1 & y_3 - y_1 & z_3 - z \\ \ldots & & \\ x_n - x_1 & y_n - y_1 & z_n - z \end{bmatrix}$$

[1] G. Shen, R. Zetik, and R. S. Thoma, "Performance comparison of TOA and TDOA based location estimation algorithms in LOS environment," in Navigation and Communication 2008 5th Workshop on Positioning, Mar. 2008, pp. 71–78. doi: 10.1109/WPNC.2008.4510359.

BAYESIAN METHODS

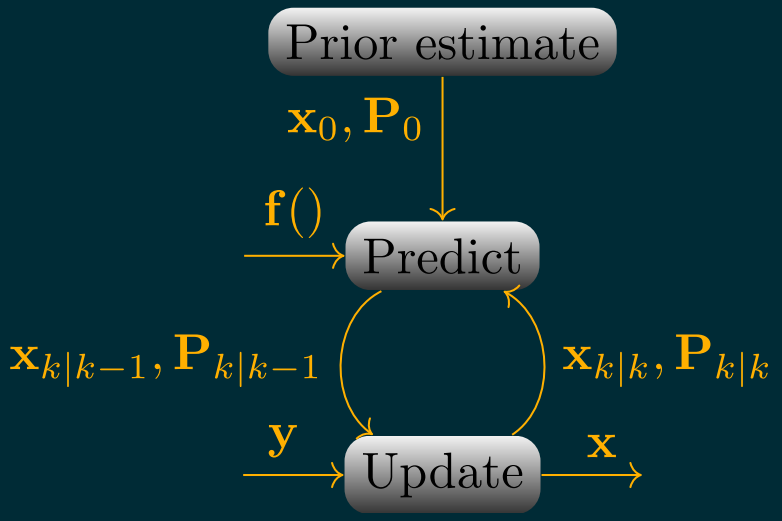
- If the system is well known, the output can be predicted from the input, i.e.: $p(y|x)$ is known. (Predictive inference)
- But we need to estimate x when based on measurements y. This can be done utilizing Bayesian rule (diagnostics inference)

$$p(x|y) = \frac{p(y|x)p(x)}{p(y)}$$

$$\text{Posterior} = \frac{\text{Likelihood} \cdot}{\text{evidenc}}$$

- Typical Bayesian methods are: Kalman filters, Particle filters, Markov models, Bayesian belief networks and Factor Graphs

BAYESIAN ESTIMATION



- Recursive or iterative
- Predict the next state using for example system dynamics, .
- Updates the state using the measured information of observable variables
- Keeps the system covariance, , up to date, to decide how much weight each variable deservers

KALMAN FILTER

1. Initialization:
   - $\hat{X}_0, P_0^-, Q_0, R_0$
2. Prediction:
   - Prior estimate of the state: $m_k^- = f(m_{k-1}, k-1)$
   - Prior estimate of the covariance: $P_k^- = F_x(m_{k-1}, k-1) P_{k-1} {F_x}^T(m_{k-1}, k-1) + Q_{k-1}$
3. Update step
   - Measurement residual update: $V_k = y_k - h(m_k^-, k)$
   - Measurement covariance update:
   - Kalman gain calculation: $K_k = P_k^- {H_p}^T(m_k^-, k) S_k^{-1}$
   - Updating the posterior state: $m_k = m_k^- + K_k V_k$
   - Updating the posterior covariance: $P_k = P_k^- - K_k S_k K_k^T$
4. Return to step 1, repeat for k positioning steps.
5. Output
   - Estimated state vector: $\hat{X}$

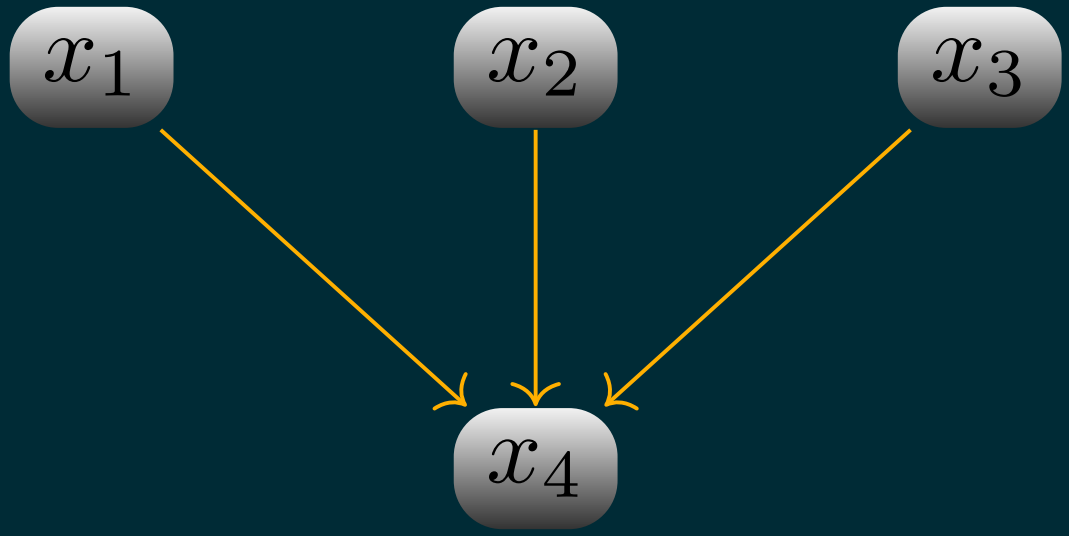
PARTICLE FILTER

1. Initialization:
   - $x_0^i \sim p_{x_0} \quad , i = 1, \ldots, N$
2. Prediction:
   - Prior estimate of the particle positions:
   - $x_k^i := f(x_{k-1}^i, k-1)$
3. Update step
   - Update particle weights:
   - $w_k^i = w_{k-1}^i \cdot p(y_k = h(x_k^i))$
   - Resample: Take N particles $x_k^i{}_{i=1}^N$ from set x_k^i so that the probability to select each sample is w_k^i.
4. Return to step 2 repeat for k positioning steps.
5. Output
   - State vector estimate:

$$\bar{x} = \sum_{i=1}^{N} (w^i x^i)$$

GRAPHICAL MODELS

BAYES NET

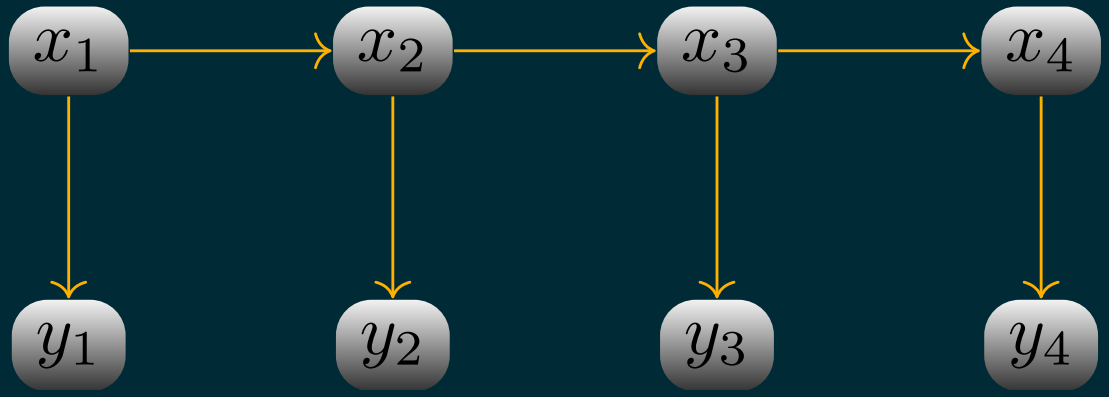


- In general joint probability $p(\mathbf{x}) = p(x_1, x_2, x_3, x_4)$.
- Utilization of independence allows factorized calculation of joint distribution .

$$p(\mathbf{x})$$

$$p(\mathbf{x}) = p(x_1) \cdot p(x_2) \cdot p(x_3) \cdot p(x_4|x_1, x_2, x_3)$$

HIDDEN MARKOV MODEL



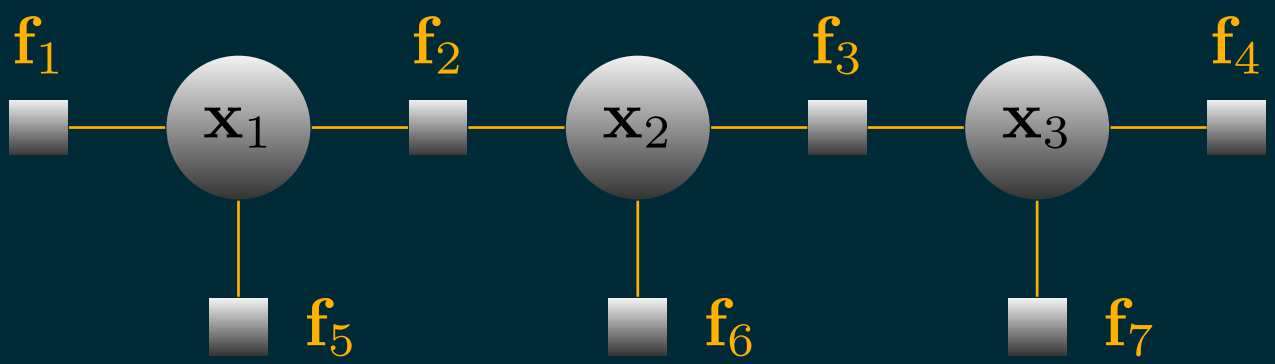
- Prior: $P(x_1)$
- State transition probability: $P(x_{i+1}|x_i)$
- Measurement / observation probability: $P(y_i|x_i)$

$$p(\mathbf{x}, \mathbf{y}) = p(x_1) \prod_{i=1}^{N-1} p(x_{i+1}|x_i) \prod_{i=1}^{N}$$

FACTOR GRAPH (FG)

- FG is a relatively new method for state estimation.
- It improves the robustness for two reasons
  1. It can probabilistically combine network topological information to the measurements.
  2. The iterative solution allows using using robust loss functions such as Huber's or Tukey's.
- We will use a generic robust loss function, which can be smoothly adjusted from LMS shape to robust Huber and Tukey forms.

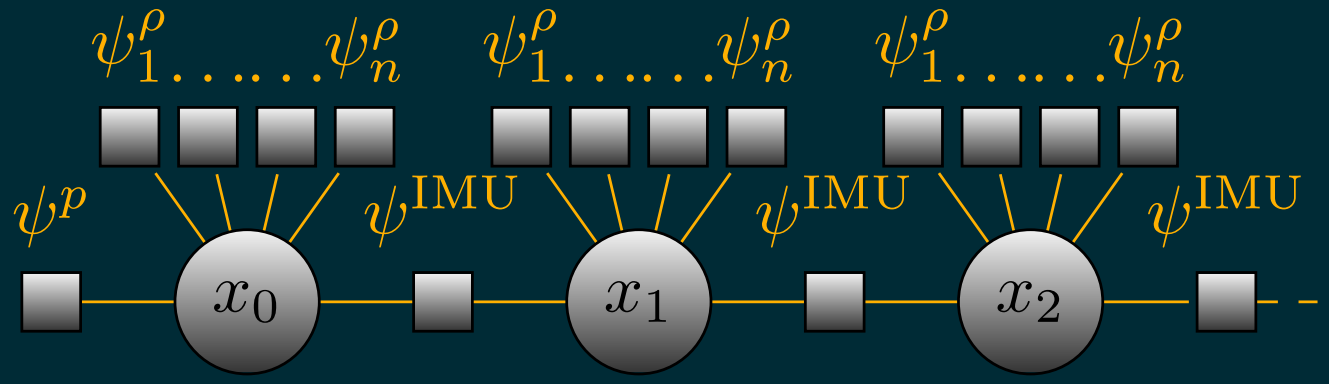
FACTOR GRAPH



- Generalized (in)depency model which can describe Bayesian networks, Markov chains Kalman filter and much more
- Allows factorized calculation of joint probability density

$$f(x_1, x_2, x_3) = \prod f_i(x_i)$$
$$= f_1(x1) f_5(x_1) f_2(x_1, x_2) f_6(x_2) f$$

FACTOR GRAPH FOR POSITIONING

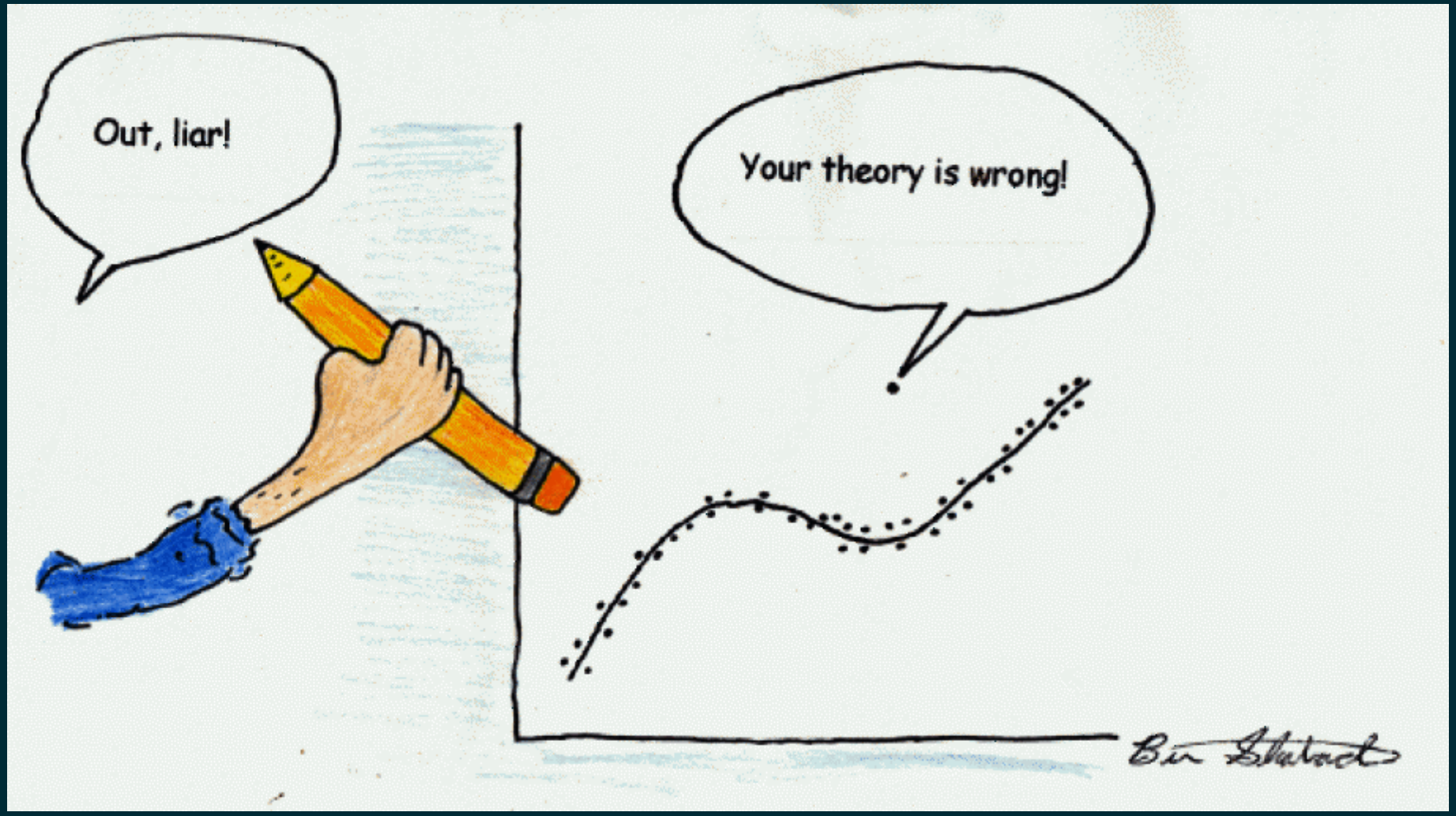


- The FG is used for estimating state variables, marked as circles
- The factors (rectangles) are probabilistic constraints, such as measurements, which are represented as functions, returning the probability of the observation.
- The factors between state variables represent their mutual dependencies

ROBUSTNESS

OUTLIER?

OUTLIER?



WHY WE NEED ROBUST METHODS?

- Errors are not always nice Gaussian noise.
- Sensor data may include also outliers, due to various reasons:
  - Sensor or programming fault
  - Exceptional sonditions: NLOS, and multipath propagation, ...
  - Novel event!
- Detection
  - Z-score, RAIM, Isolation forest, ...
- Handling
  - RANSAC
  - Robust statistics

DETECTION

- Z-score: A value is an outlier if it far from the center

$$z = \frac{x - \mu}{\sigma}$$

- Robust score: The same but robust

$$r_z = \frac{x - \mathrm{Med}(x)}{\mathrm{IQR}(x)}$$

- RAIM, Receiver autonomous integrity monitoring: ν is the residuals vector, n is the degrees of freedom and m is number of measurements. The significance level of the χ^2 is often $\alpha = 0.001$.

$$\frac{\nu^T \nu}{m - n - 1} < \chi^2_\alpha(m - n - 1)$$

RANSAC

- Random sample consensus is an iterative method to find out a concensus set of inliners

1. Select a minimum random subset of variables and fit the model using this subset
2. Other varibles are added as long as they do not look like outliers
3. The expanded model is called as the consensus set
4. The process can be repeated until good enough model is obtained

M-ESTIMATORS

Maximum likelihood estimators.

$$\hat{\mu_M} = \underset{\mu}{\operatorname{argmin}} \sum_{i=1}^{n} \rho(x_i - \mu)$$

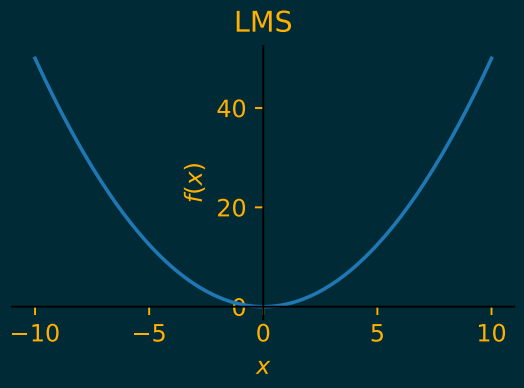
If ρ is differentiable with $\psi = \rho'$, then $\hat{\mu}_M$ satisfies:

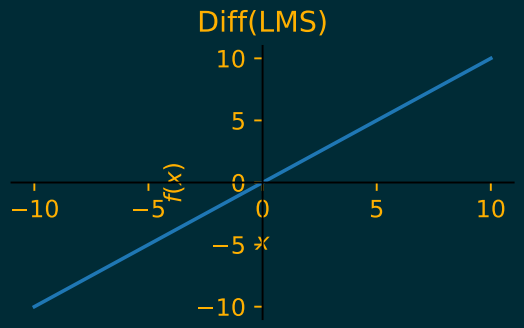
$$\sum_{i=1}^{n} \psi(x_i - \hat{\mu}_M) = 0$$

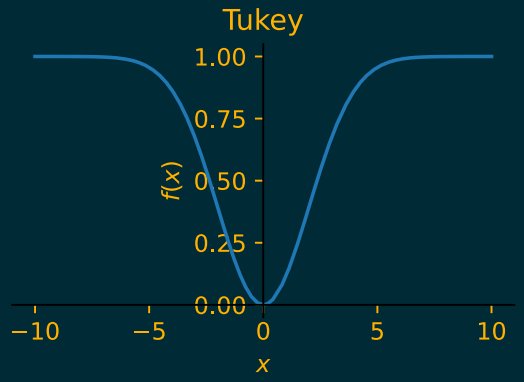
- ρ is the loss function of the estimator, in LMS case $\rho(x) = 1/2x^2$
- The influence function, ψ, measures how the estimate changes when contamination is added in x. For lms case $\psi(x) = x$

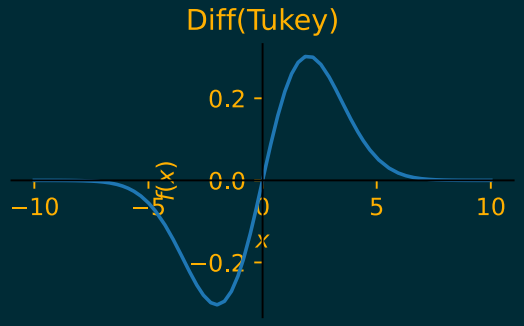
Robust Statistics, Peter Rousseeuw

ROBUST LOSS FUNCTION









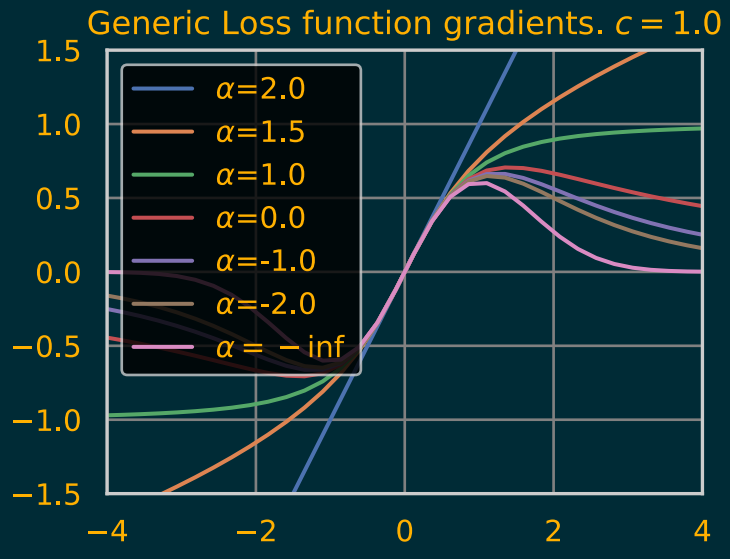
$$\rho_{\mathrm{lms}} = \frac{1}{2}(\hat{x} -$$

$$\psi_{\mathrm{lms}} = \frac{d}{dx}\rho_{\mathrm{l}}$$

$$\rho_{\mathrm{tk}} = 1 - e^{($$

$$\psi_{\mathrm{tk}} = \frac{d}{dx}\rho_{\mathrm{tk}}$$

GENERIC, ROBUST LOSS AND INFLUENCE FUNCTIONS

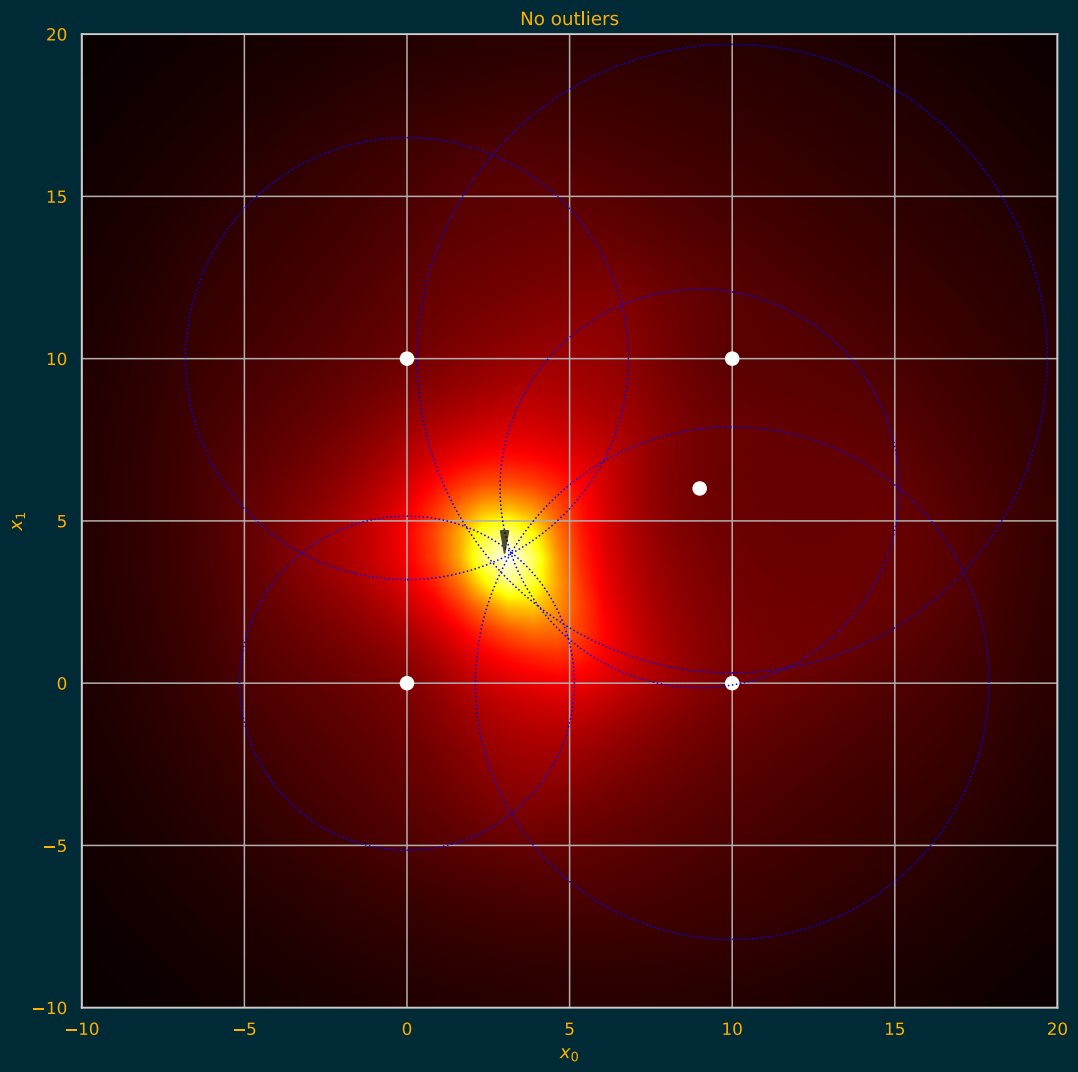


$$\rho(x, \alpha, c) = \begin{cases} \frac{1}{2}(x/c)^2 & \text{if } \alpha = \\ \log\left(\frac{1}{2}(x/c)^2 + 1\right) & \text{if } \alpha = \\ 1 - \exp\left(-\frac{1}{2}(x/c)^2\right) & \text{if } \alpha = \\ \frac{|\alpha-2|}{\alpha}\left(\left(\frac{(x/c)^2}{|\alpha-2|} + 1\right)^{\frac{\alpha}{2}} - 1\right) & \text{other} \end{cases}$$

- L2-loss, when $\alpha = 2$
- Huber-loss, when $\alpha = 1$
- Tukey-loss, when $\alpha = -\infty$

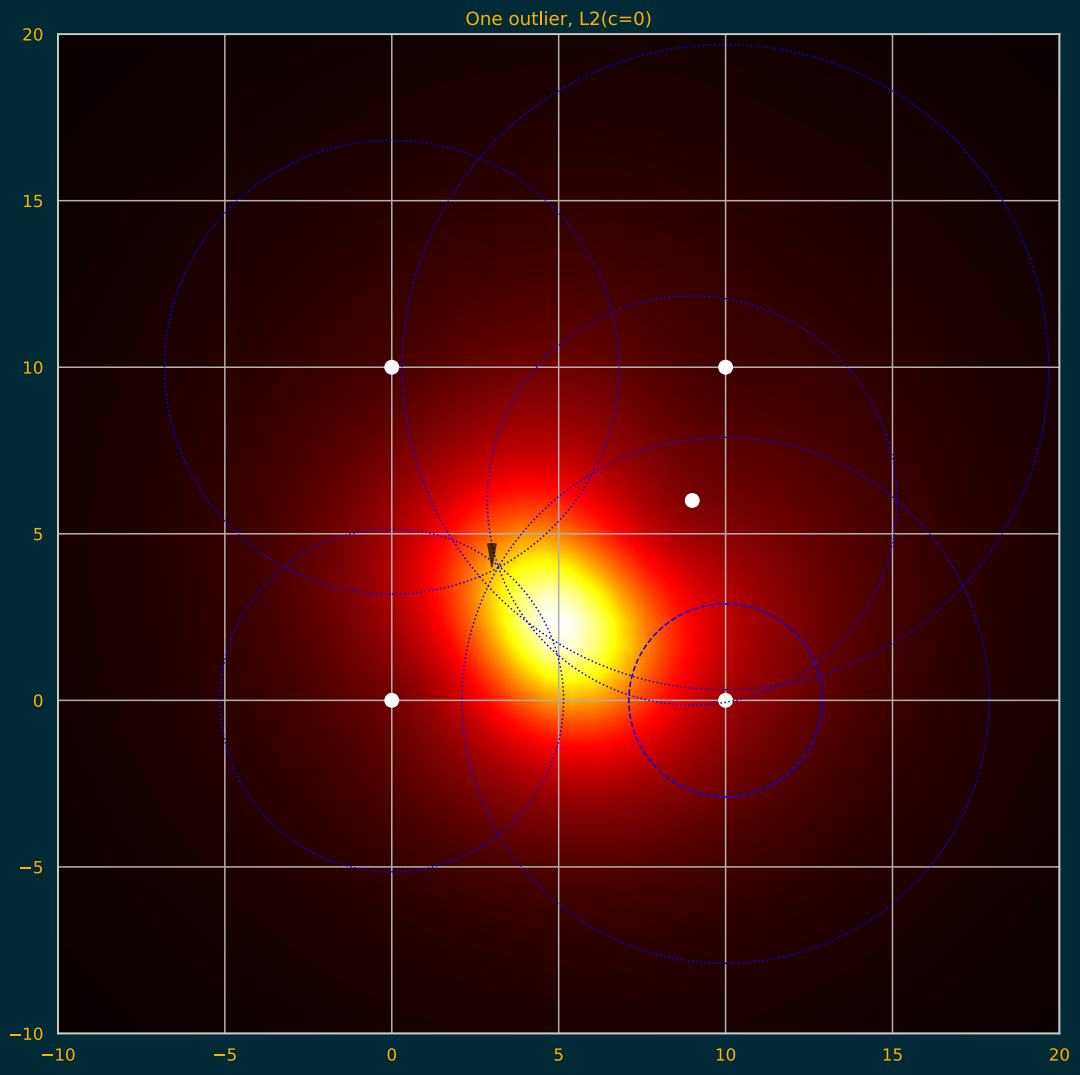
J. T. Barron, “A General and Adaptive Robust Loss Function,” in 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), Jun. 2019, pp. 4326–4334. doi: 10.1109/CVPR.2019.00446.

FITNESS SPACE, NO OUTLIERS



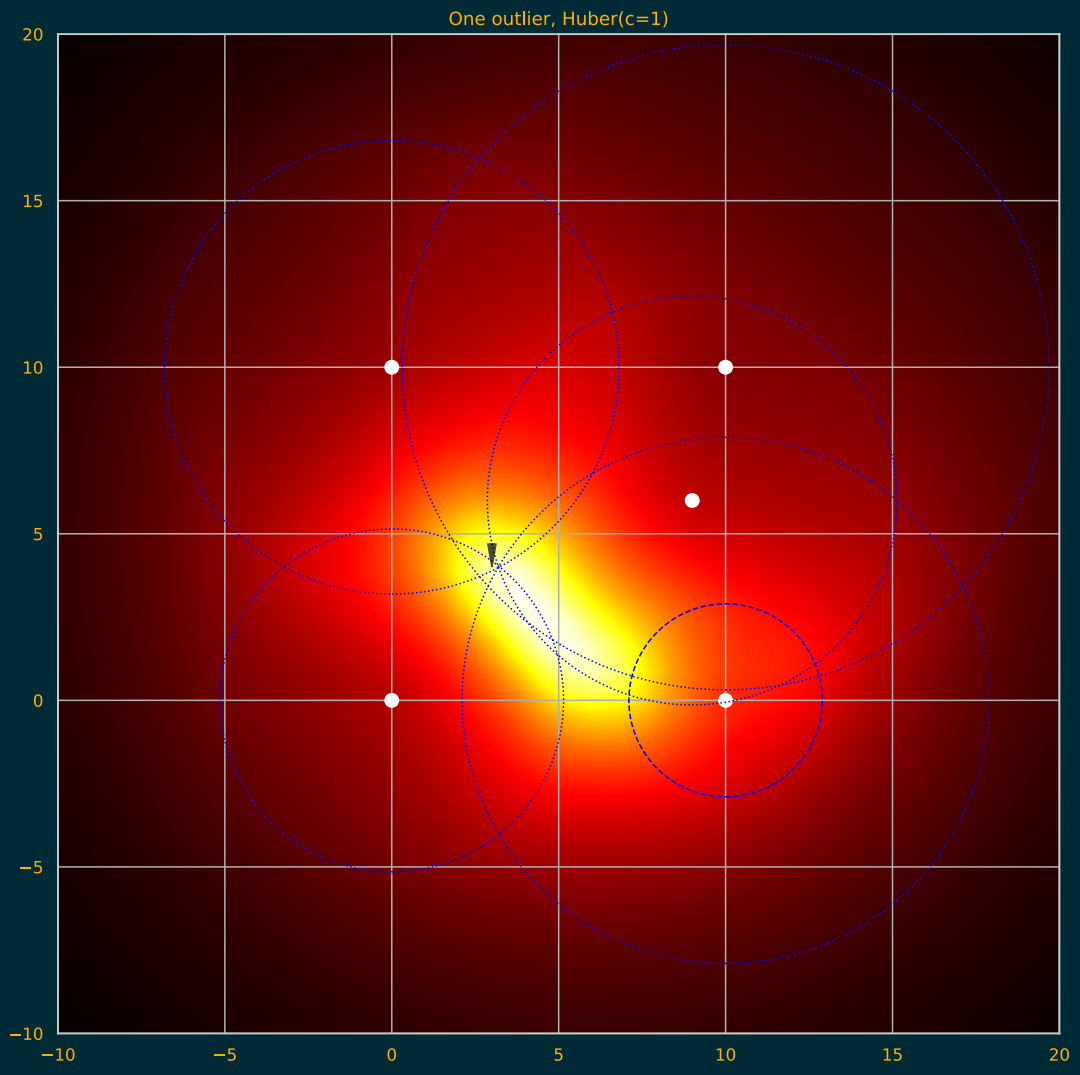
- No outliers
- L2 loss function
- White filled circles = anchors
- Black wedge = True location
- Blue circles = measured distances

FITNESS SPACE, ONE OUTLIER



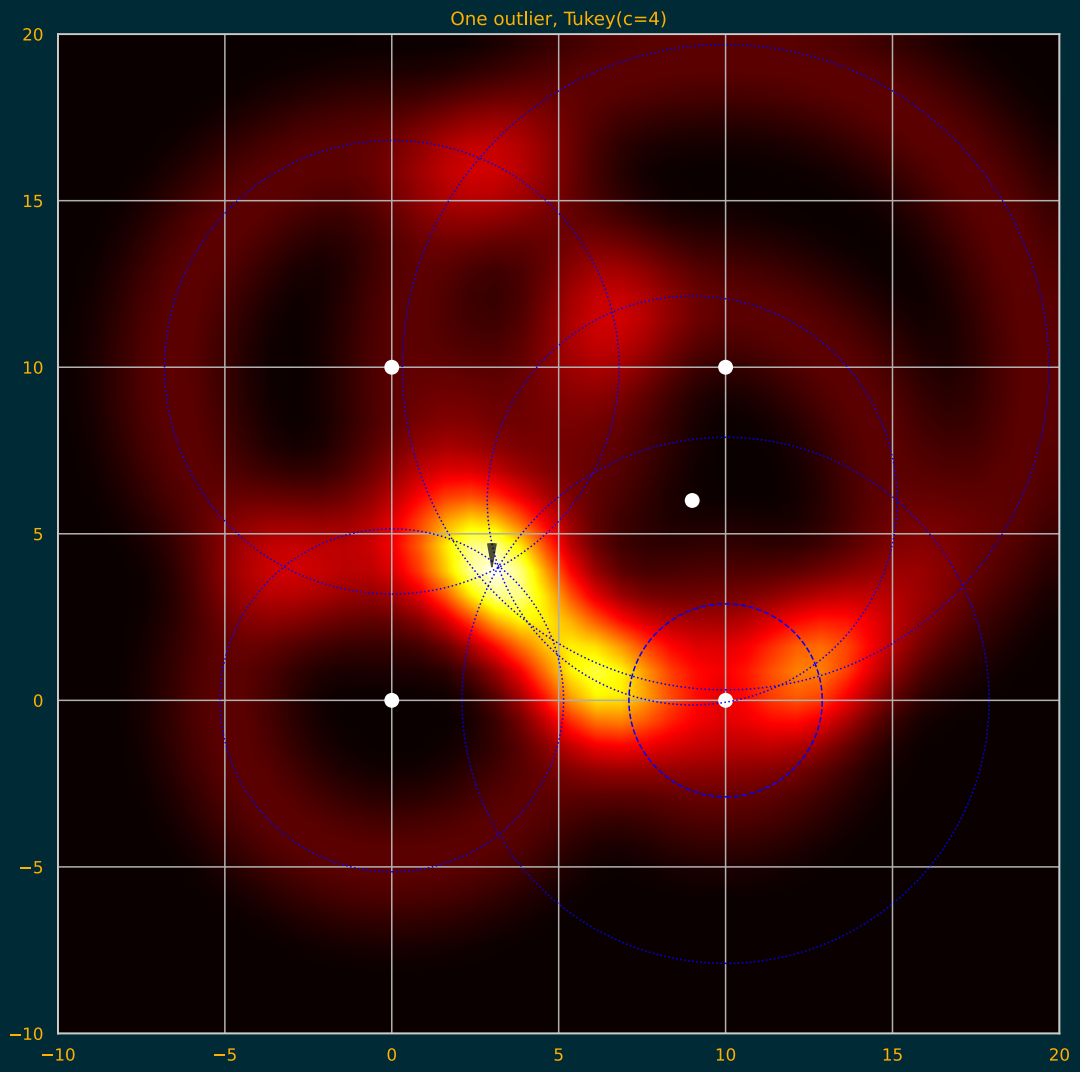
- One outlier
- L2 loss function
- White filled circles = anchors
- Black wedge = True location
- Blue cicles = measured distances

FITNESS SPACE, ONE OUTLIER



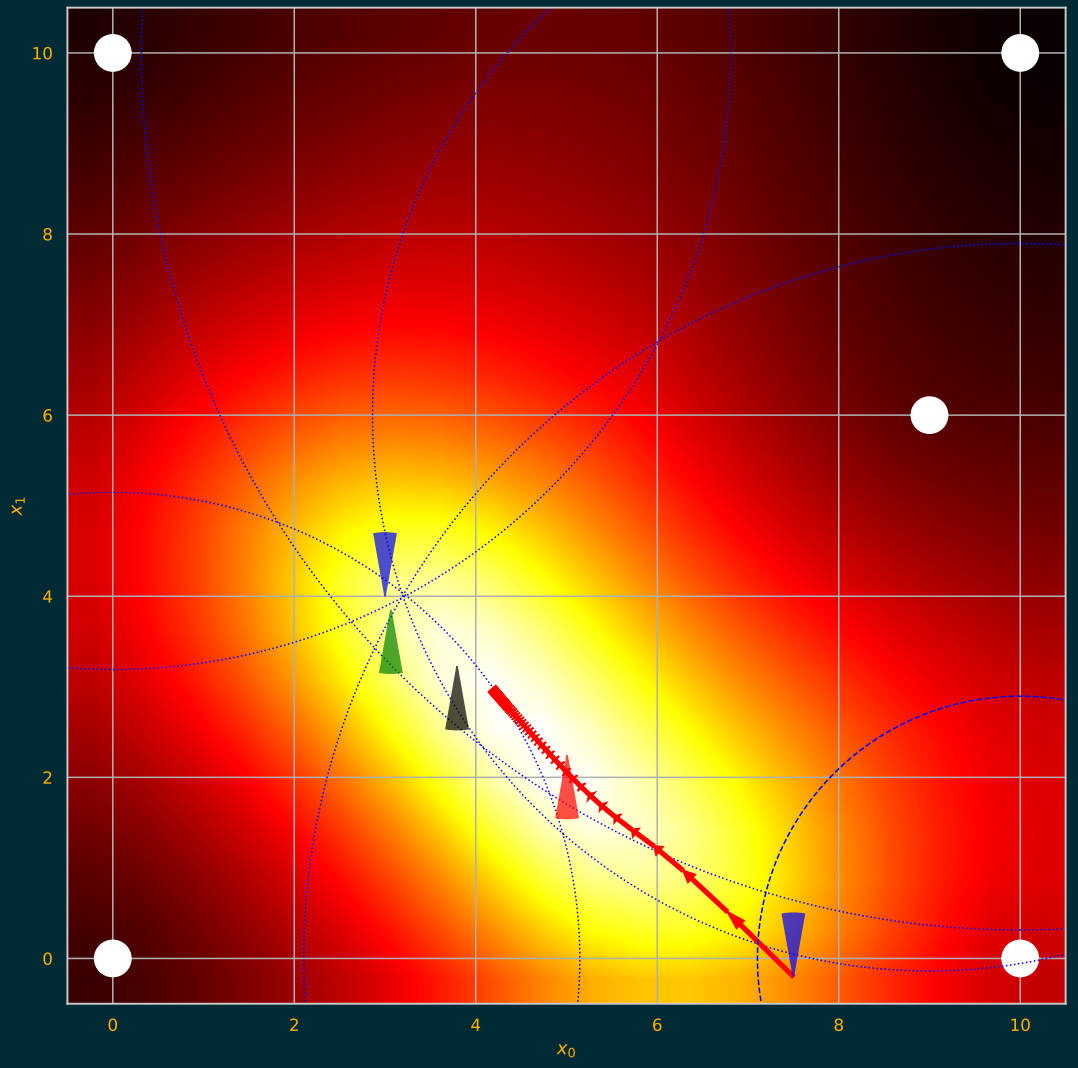
- One outlier
- Huber loss function
- White filled circles = anchors
- Black wedge = True location
- Blue cicles = measured distances

FITNESS SPACE, ONE OUTLIER



- One outlier
- Tukey loss function
- White filled circles = anchors
- Black wedge = True location
- Blue cicles = measured distances

GRADIENT DESCENT, WITH OUTLIER



- Blue wedge = starting point and true position
- Red path = gradient descent iteration with outlier
- Green wedge = with
- Red wedge = with and outlier
- Black wedge = with a robust loss

L_2

L_2

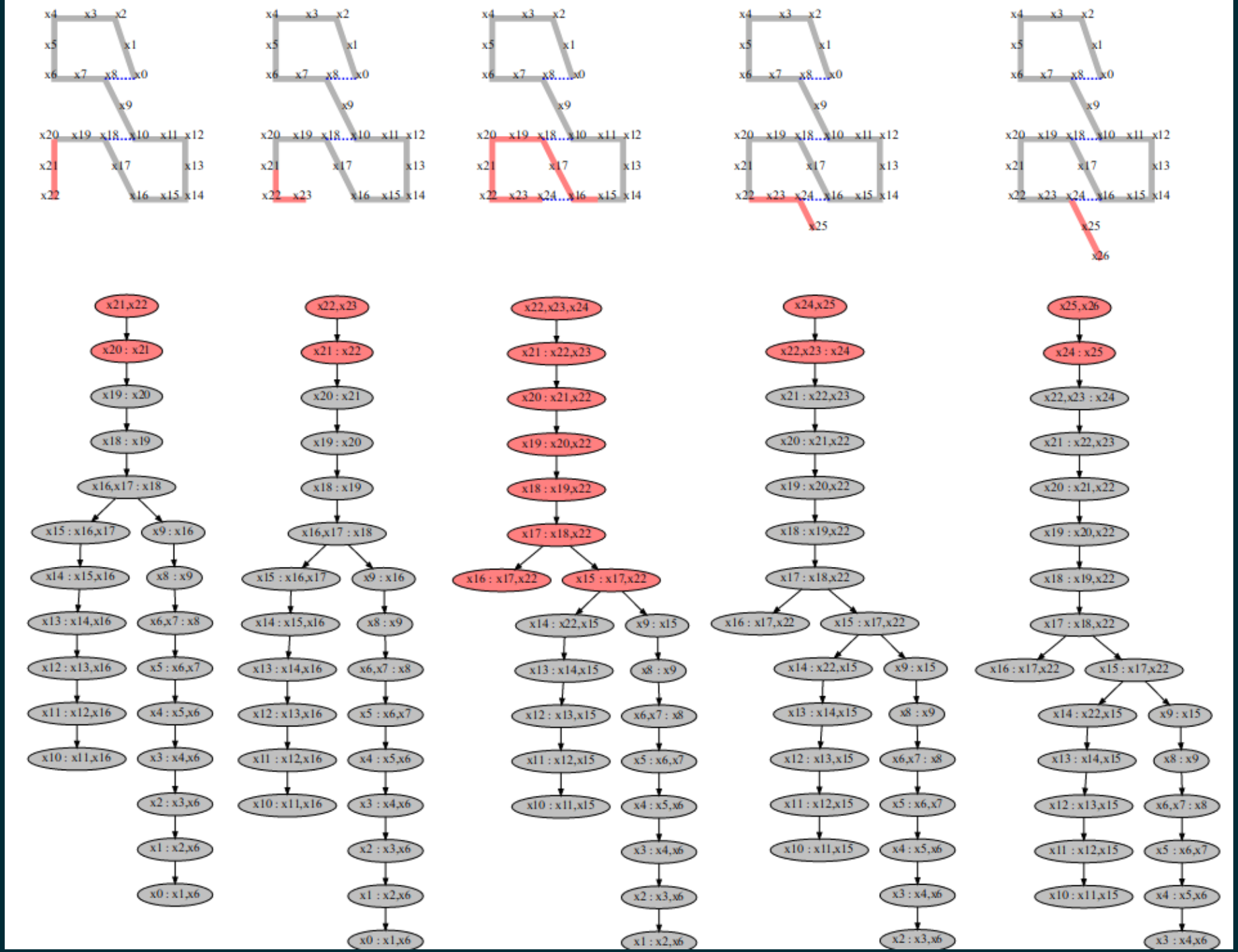
OUTLIER VS ROBUSTNESS

ROBOT POSITIONING WITH GTSAM

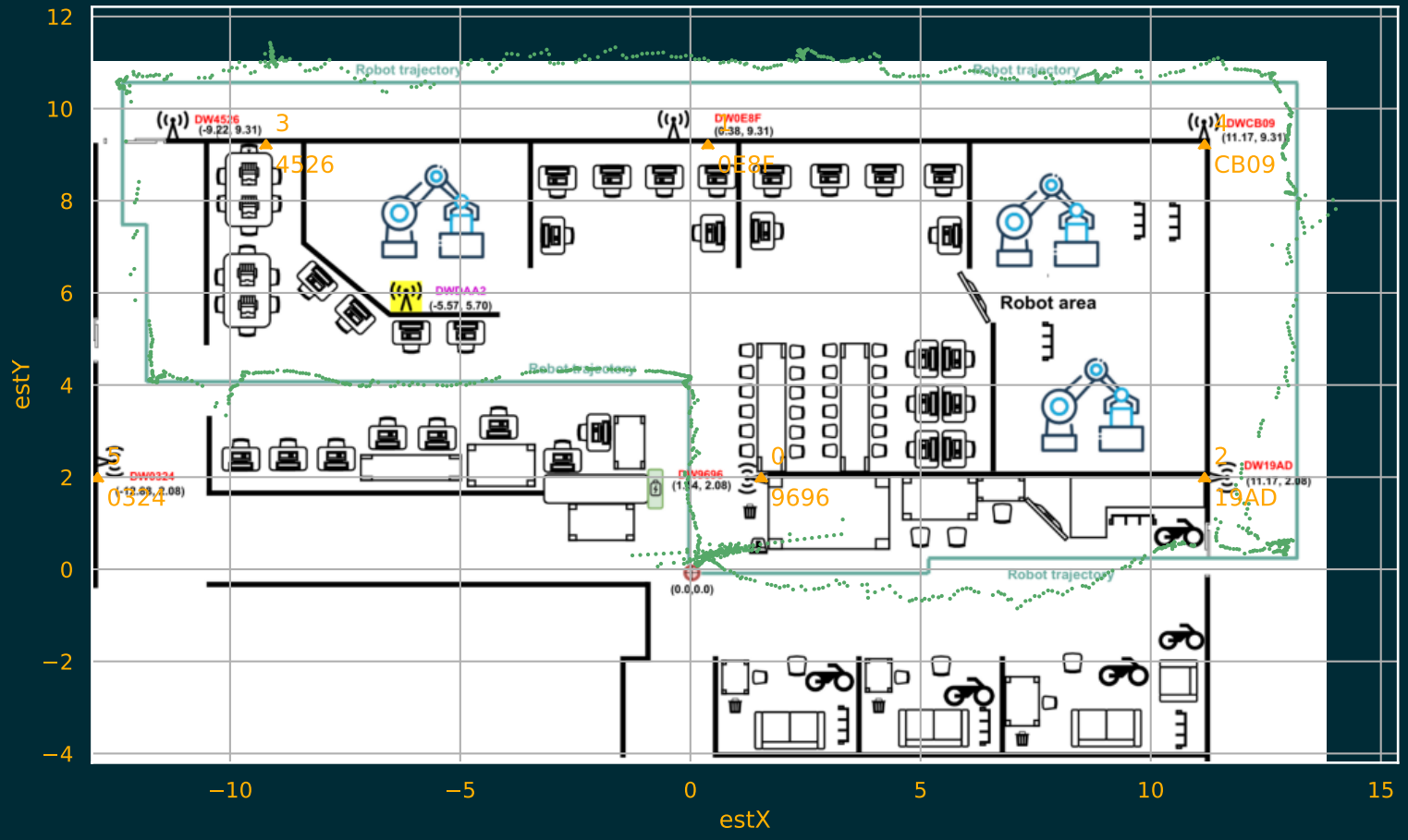
GT-SAM

- Georgia Tech Smoothing And Mapping.
- Open source, LGPL, somewhat documented
- A complete library for FG, particularly for positioning. SLAM + other methods
- Written in C/C++ but has interfaces to Python and MATLAB
- Supports Robust loss functions: Huber, Tukey, add more
- Fast non-linear solvers, including Gauss-Newton and Levenberg-Marquart

ITERATIVE SAM


x4 x3 x2
x5 x1
x6 x7 x8 x0
x9
x20 x19 x18 x10 x11 x12
x21 x17 x13
x22 x16 x15 x14
x23
x24
x25
x26
x21,x22
x20 : x21
x19 : x20
x18 : x19
x16,x17 : x18
x15 : x16,x17
x9 : x16
x14 : x15,x16
x8 : x9
x13 : x14,x16
x6,x7 : x8
x12 : x13,x16
x5 : x6,x7
x11 : x12,x16
x4 : x5,x6
x10 : x11,x16
x3 : x4,x6
x2 : x3,x6
x1 : x2,x6
x0 : x1,x6
x22,x23
x21 : x22
x20 : x21
x19 : x20
x18 : x19
x16,x17 : x18
x15 : x16,x17
x9 : x16
x14 : x15,x16
x8 : x9
x13 : x14,x16
x6,x7 : x8
x12 : x13,x16
x5 : x6,x7
x11 : x12,x16
x4 : x5,x6
x10 : x11,x16
x3 : x4,x6
x2 : x3,x6
x1 : x2,x6
x0 : x1,x6
x22,x23,x24
x21 : x22,x23
x20 : x21,x22
x19 : x20,x22
x18 : x19,x22
x17 : x18,x22
x16 : x17,x22
x15 : x17,x22
x14 : x22,x15
x9 : x15
x13 : x14,x15
x8 : x9
x12 : x13,x15
x6,x7 : x8
x11 : x12,x15
x5 : x6,x7
x10 : x11,x15
x4 : x5,x6
x3 : x4,x6
x2 : x3,x6
x1 : x2,x6
x24,x25
x22,x23 : x24
x21 : x22,x23
x20 : x21,x22
x19 : x20,x22
x18 : x19,x22
x17 : x18,x22
x16 : x17,x22
x15 : x17,x22
x14 : x22,x15
x9 : x15
x13 : x14,x15
x8 : x9
x12 : x13,x15
x6,x7 : x8
x11 : x12,x15
x5 : x6,x7
x10 : x11,x15
x4 : x5,x6
x3 : x4,x6
x2 : x3,x6
x25,x26
x24 : x25
x22,x23 : x24
x21 : x22,x23
x20 : x21,x22
x19 : x20,x22
x18 : x19,x22
x17 : x18,x22
x16 : x17,x22
x15 : x17,x22
x14 : x22,x15
x9 : x15
x13 : x14,x15
x8 : x9
x12 : x13,x15
x6,x7 : x8
x11 : x12,x15
x5 : x6,x7
x10 : x11,x15
x4 : x5,x6
x3 : x4,x6

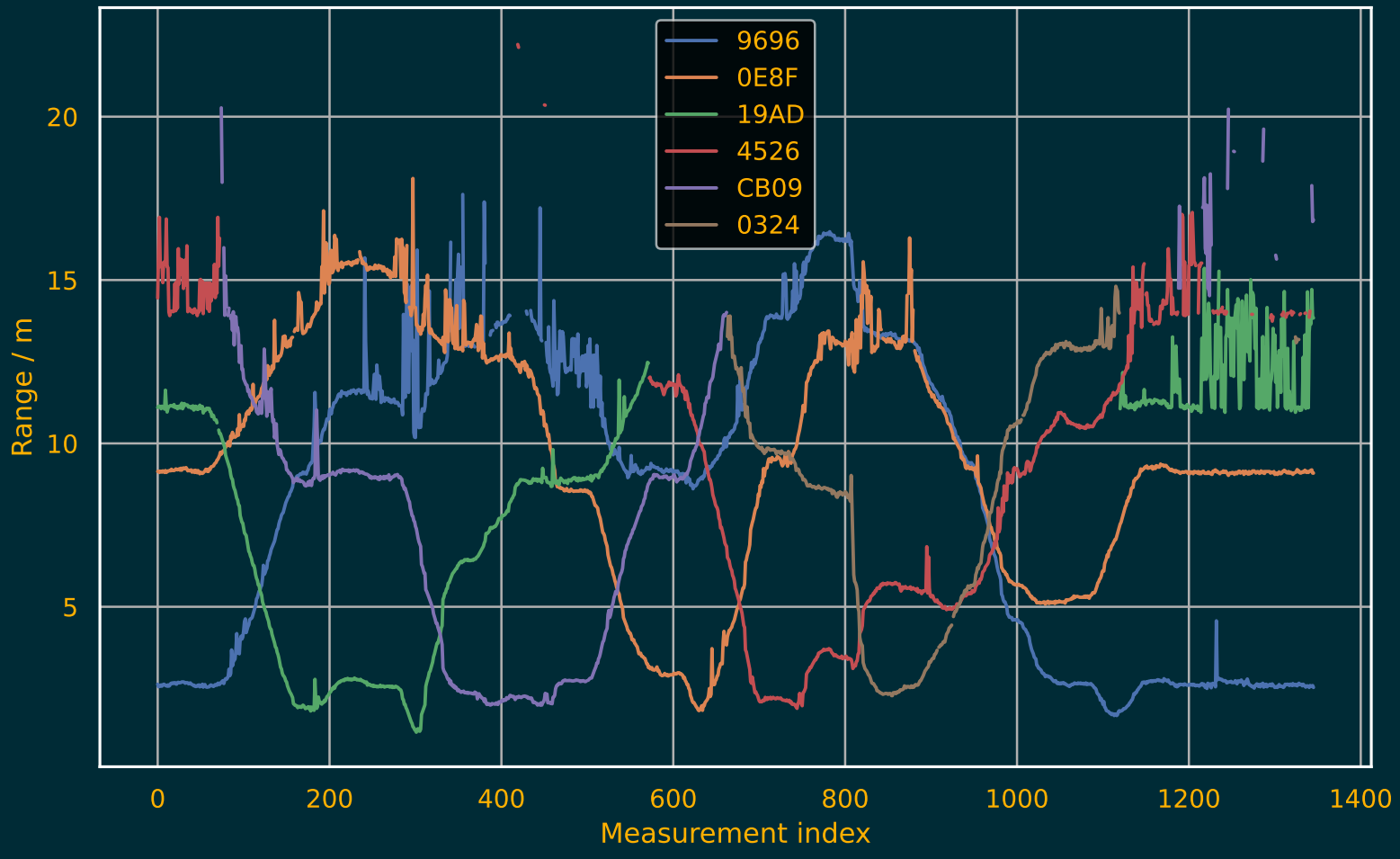
CASE STUDY: ROBOT POSITIONING



M. Elsanhoury et al., “Precise Indoor Positioning System for Mobile Robots via Smoothed UWB/IMU Sensor Fusion,” in 2023 13th International Conference on Indoor Positioning and Indoor Navigation (IPIN), Sep. 2023, pp. 1–6. doi: 10.1109/IPIN57070.2023.10332542.

LABORATORY

SENSOR DATA - RANGESS



GTSAM NOISE MODELS

```
# Set Noise parameters
prior_sigmas = gtsam.Point3(1, 1, math.pi)
odo_sigmas = gtsam.Point3(0.1, 0.1, math.pi)
sigmaR = 0.1        # range standard deviation

prior_noise = NM.Diagonal.Sigmas(prior_sigmas)  # prior
odo_noise = NM.Diagonal.Sigmas(odo_sigmas)     # odometry
gaussian = NM.Isotropic.Sigma(1, sigmaR)     # non-robust
tukey = NM.Robust.Create(NM.mEstimator.Tukey.Create(15), gaus
range_noise = tukey if robust else gaussian
```

INITIALIZATION

```
 1 def initialize_isam():
 2     # Initialize iSAM
 3     isam = gtsam.ISAM2()
 4     return(isam)
 5
 6 def initialize_fg():
 7     new_factors = gtsam.NonlinearFactorGraph()
 8     pose0 = Pose2(lms_solutions[0,0], lms_solutions[0,1], 0)
 9     new_factors.addPriorPose2(0, pose0, prior_noise)
10     initial = gtsam.Values()
11     initial.insert(0, pose0)
12     return new_factors, initial, pose0
```

INITIALIZATION

```
 1 def initialize_isam():
 2     # Initialize iSAM
 3     isam = gtsam.ISAM2()
 4     return(isam)
 5
 6 def initialize_fg():
 7     new_factors = gtsam.NonlinearFactorGraph()
 8     pose0 = Pose2(lms_solutions[0,0], lms_solutions[0,1], 0)
 9     new_factors.addPriorPose2(0, pose0, prior_noise)
10     initial = gtsam.Values()
11     initial.insert(0, pose0)
12     return new_factors, initial, pose0
```

INITIALIZATION

```
 1 def initialize_isam():
 2     # Initialize iSAM
 3     isam = gtsam.ISAM2()
 4     return(isam)
 5
 6 def initialize_fg():
 7     new_factors = gtsam.NonlinearFactorGraph()
 8     pose0 = Pose2(lms_solutions[0,0], lms_solutions[0,1], 0)
 9     new_factors.addPriorPose2(0, pose0, prior_noise)
10     initial = gtsam.Values()
11     initial.insert(0, pose0)
12     return new_factors, initial, pose0
```

INITIALIZATION

```
 1 def initialize_isam():
 2     # Initialize iSAM
 3     isam = gtsam.ISAM2()
 4     return(isam)
 5
 6 def initialize_fg():
 7     new_factors = gtsam.NonlinearFactorGraph()
 8     pose0 = Pose2(lms_solutions[0,0], lms_solutions[0,1], 0)
 9     new_factors.addPriorPose2(0, pose0, prior_noise)
10     initial = gtsam.Values()
11     initial.insert(0, pose0)
12     return new_factors, initial, pose0
```

INITIALIZATION

```
 1 def initialize_isam():
 2     # Initialize iSAM
 3     isam = gtsam.ISAM2()
 4     return(isam)
 5
 6 def initialize_fg():
 7     new_factors = gtsam.NonlinearFactorGraph()
 8     pose0 = Pose2(lms_solutions[0,0], lms_solutions[0,1], 0)
 9     new_factors.addPriorPose2(0, pose0, prior_noise)
10     initial = gtsam.Values()
11     initial.insert(0, pose0)
12     return new_factors, initial, pose0
```

INITIALIZATION

```
 1 def initialize_isam():
 2     # Initialize iSAM
 3     isam = gtsam.ISAM2()
 4     return(isam)
 5
 6 def initialize_fg():
 7     new_factors = gtsam.NonlinearFactorGraph()
 8     pose0 = Pose2(lms_solutions[0,0], lms_solutions[0,1], 0)
 9     new_factors.addPriorPose2(0, pose0, prior_noise)
10     initial = gtsam.Values()
11     initial.insert(0, pose0)
12     return new_factors, initial, pose0
```

LANDMARKS (UWB-ANCHORS)

```
landmark_noise = NM.Isotropic.Sigma(2, 0.05)  # accurate LM
for i, anchorid in enumerate(anchors['id']):
    landmark_key = gtsam.symbol('L', int(anchorid, 16))
    landmark=gtsam.PriorFactorPoint2(landmark_key,
                    Point2(anchors['pos'][i,:2]),
                    landmark_noise)
    new_factors.add(landmark)
    initial.insert(landmark_key, anchors['pos'][i,:2])
```

ADDING RANGE FACTORS

```
# Add range factors
ranges,variance=get_measurements(pos.distances, i+N0)
# {'9696': 12.3, '0E8F': 8.47, '19AD': 8.93, 'CB09': 2.78}
for anchor_id, pseudorange in ranges.items():
    landmark_key = gtsam.symbol('L', int(anchor_id, 16))
    sigma=np.sqrt(variance[anchor_id])
    gaussian = NM.Isotropic.Sigma(1, sigma)     # non-robust
    tukey = NM.Robust.Create(NM.mEstimator.Tukey.Create(15),
                             gaussian)
    range_noise = tukey if robust else gaussian
    range_factor = gtsam.RangeFactor2D(i, landmark_key,
                                       pseudorange, range_noise)
    new_factors.add(range_factor)
```

ADDING RANGE FACTORS

```
# Add range factors
ranges,variance=get_measurements(pos.distances, i+N0)
# {'9696': 12.3, '0E8F': 8.47, '19AD': 8.93, 'CB09': 2.78}
for anchor_id, pseudorange in ranges.items():
    landmark_key = gtsam.symbol('L', int(anchor_id, 16))
    sigma=np.sqrt(variance[anchor_id])
    gaussian = NM.Isotropic.Sigma(1, sigma)     # non-robust
    tukey = NM.Robust.Create(NM.mEstimator.Tukey.Create(15),
                             gaussian)
    range_noise = tukey if robust else gaussian
    range_factor = gtsam.RangeFactor2D(i, landmark_key,
                                       pseudorange, range_noise)
    new_factors.add(range_factor)
```

ADDING RANGE FACTORS

```python
# Add range factors
ranges,variance=get_measurements(pos.distances, i+N0)
# {'9696': 12.3, '0E8F': 8.47, '19AD': 8.93, 'CB09': 2.78}
for anchor_id, pseudorange in ranges.items():
    landmark_key = gtsam.symbol('L', int(anchor_id, 16))
    sigma=np.sqrt(variance[anchor_id])
    gaussian = NM.Isotropic.Sigma(1, sigma)     # non-robust
    tukey = NM.Robust.Create(NM.mEstimator.Tukey.Create(15),
                             gaussian)
    range_noise = tukey if robust else gaussian
    range_factor = gtsam.RangeFactor2D(i, landmark_key,
                                       pseudorange, range_noise)
    new_factors.add(range_factor)
```

ADDING RANGE FACTORS

```
# Add range factors
ranges,variance=get_measurements(pos.distances, i+N0)
# {'9696': 12.3, '0E8F': 8.47, '19AD': 8.93, 'CB09': 2.78}
for anchor_id, pseudorange in ranges.items():
    landmark_key = gtsam.symbol('L', int(anchor_id, 16))
    sigma=np.sqrt(variance[anchor_id])
    gaussian = NM.Isotropic.Sigma(1, sigma)     # non-robust
    tukey = NM.Robust.Create(NM.mEstimator.Tukey.Create(15),
                             gaussian)
    range_noise = tukey if robust else gaussian
    range_factor = gtsam.RangeFactor2D(i, landmark_key,
                                       pseudorange, range_noise)
    new_factors.add(range_factor)
```

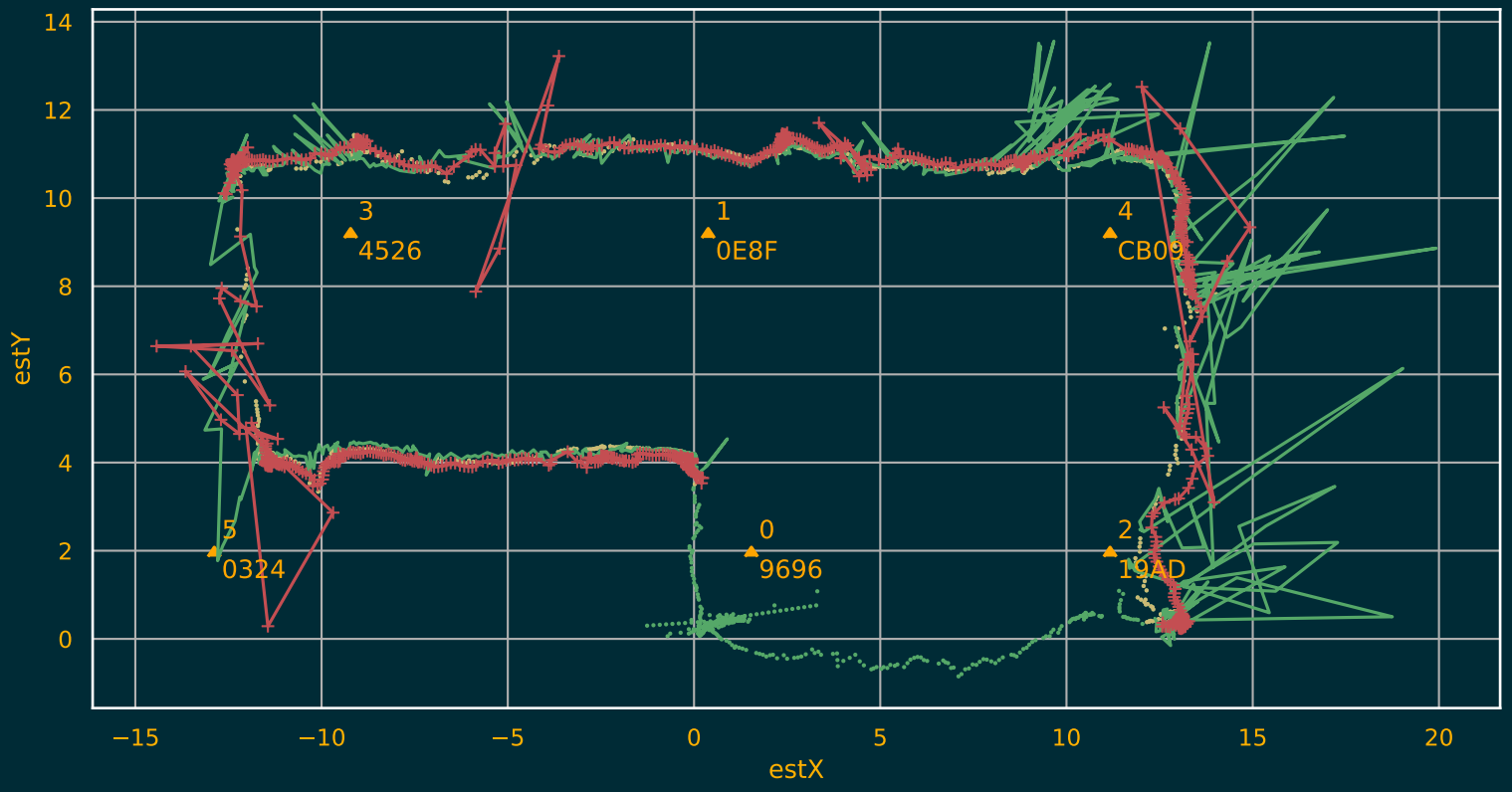
ADDING RANGE FACTORS

```
# Add range factors
ranges,variance=get_measurements(pos.distances, i+N0)
# {'9696': 12.3, '0E8F': 8.47, '19AD': 8.93, 'CB09': 2.78}
for anchor_id, pseudorange in ranges.items():
    landmark_key = gtsam.symbol('L', int(anchor_id, 16))
    sigma=np.sqrt(variance[anchor_id])
    gaussian = NM.Isotropic.Sigma(1, sigma)     # non-robust
    tukey = NM.Robust.Create(NM.mEstimator.Tukey.Create(15),
                             gaussian)
    range_noise = tukey if robust else gaussian
    range_factor = gtsam.RangeFactor2D(i, landmark_key,
                                       pseudorange, range_noise)
    new_factors.add(range_factor)
```

ADDING RANGE FACTORS

```python
# Add range factors
ranges,variance=get_measurements(pos.distances, i+N0)
# {'9696': 12.3, '0E8F': 8.47, '19AD': 8.93, 'CB09': 2.78}
for anchor_id, pseudorange in ranges.items():
    landmark_key = gtsam.symbol('L', int(anchor_id, 16))
    sigma=np.sqrt(variance[anchor_id])
    gaussian = NM.Isotropic.Sigma(1, sigma)     # non-robust
    tukey = NM.Robust.Create(NM.mEstimator.Tukey.Create(15),
                             gaussian)
    range_noise = tukey if robust else gaussian
    range_factor = gtsam.RangeFactor2D(i, landmark_key,
                                       pseudorange, range_noise)
    new_factors.add(range_factor)
```

THE TRACK



- Green = wLMS solution
- Red = FG solution with Tukey loss function

RESULTS

Method	Points	Time	RMSE
FG-Tukey	900	0.48 s	0.38 m
FG-LMS	900	0.47 s	0.48 m
wLMS	900	0.31 s	0.7 m
Pub*	?	?	0.1 m

*Kalman filter (EKF) with Sensor fusion Xsens IMU and Rauch-Tung-Striebel (RTS) Smoother

SUMMARY

- Robust methods are necessary in case of outliers or non-gaussian noise
- Some new methods are needed in the toolbox
  - Outlier detection
  - Robust statistics
  - RANSAC, FG
- Due to simplicity, Gradient descent is a great tool to study problems
- Factor graph is versatile and intuitive way to solve various state estimation problems with sensor fusion
- Python is excellent tool for machine learning, and it can be also fast if used wisely

END