CS 4610/5335 – Lecture 14
SLAM, Optimization, and Control

Lawson L.S. Wong
Northeastern University
3/9/22

Material adapted from:
1. Robert Platt, CS 4610/5335
2. Peter Corke, Robotics, Vision and Control
3. Sebastian Thrun, Wolfram Burgard, & Dieter Fox, Probabilistic Robotics
4. Marc Toussaint, U. Stuttgart Robotics Course
5. Russ Tedrake, MIT 6.832

Announcements

Ex3 update (see ex3_v2 and description on Canvas)

Ex4 will be posted on Friday / Saturday

After spring break, on 3/21 (lec15):

- First of 2 ethics lectures by Vance Ricks
- There will be a short ethics assignment (due ~4/1)

Announcements

Ex3 update (see ex3_v2 and description on Canvas)

Ex4 will be posted on Friday / Saturday

After spring break, on 3/21 (lec15):
- First of 2 ethics lectures by Vance Ricks
- There will be a short ethics assignment (due ~4/1)

Project:
- Some equipment will be available by Friday
  - RGB-D cameras (some D435s)
  - Turtlebots and Duckiebots
- Let us know if you want to borrow them this week (post unresolved followup on your project thread)

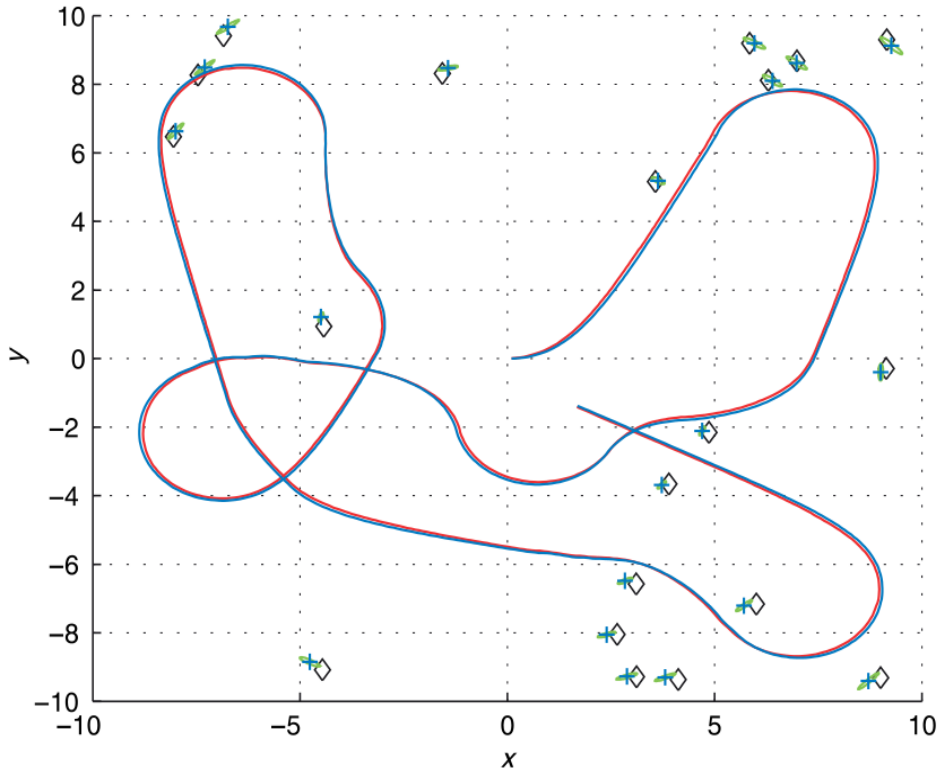
Outline

MOAR SLAM
- Rao-Blackwellized SLAM (e.g., FastSLAM)
  - with particle filtering / sequential Monte-Carlo
- Occupancy-grid mapping and SLAM
- RGB-D SLAM
- Pose-graph SLAM (e.g., GraphSLAM)

Trajectory optimization
- Direct transcription
- Direct shooting
- Model-predictive control (MPC)

Linear control
- Linear-quadratic regulator (LQR)
- Time-varying LQR
- Linear-quadratic-Gaussian (LQG)

Recap: SLAM using the EKF



Estimate both robot position and landmark positions:

$$\hat{\boldsymbol{x}} = (\underbrace{x_v, y_v, \theta_v}_{\text{robot position}}, \underbrace{x_1, y_1, x_2, y_2, \cdots x_M, y_M}_{\text{Landmark positions}})$$

Vehicle covariance

Vehicle/landmark covariance

$$\hat{\boldsymbol{P}} = \begin{pmatrix} \hat{\boldsymbol{P}}_{vv} & \hat{\boldsymbol{P}}_{vm} \\ \hat{\boldsymbol{P}}_{vm}^T & \hat{\boldsymbol{P}}_{mm} \end{pmatrix}$$

Landmark covariance

Recap: SLAM using the EKF

$$\hat{\boldsymbol{x}}^{+}\langle k+1\rangle = \boldsymbol{f}\left(\hat{\boldsymbol{x}}\langle k\rangle, \hat{\boldsymbol{u}}\langle k\rangle\right)$$ — predict state one step ahead

$$\hat{P}^{+}\langle k+1\rangle = F_x \hat{P}\langle k\rangle F_x^T + F_v V F_v^T$$ — project covariance one step ahead

prediction phase

$$\boldsymbol{\nu} = \boldsymbol{z}^{\#}\langle k+1\rangle - h\left(\hat{\boldsymbol{x}}^{+}\langle k+1\rangle, \boldsymbol{p}_i\right)$$ — new information – innovation

$$K = P^{+}\langle k+1\rangle H_x^T \left(H_x P^{+}\langle k+1\rangle H_x^T + H_w W H_w^T\right)^{-1}$$ — how to distribute the innovation to states – Kalman gain

$$\hat{\boldsymbol{x}}\langle k+1\rangle = \hat{\boldsymbol{x}}^{+}\langle k+1\rangle + K\boldsymbol{\nu}$$ — state updated with innovation

$$\hat{P}\langle k+1\rangle = \hat{P}^{+}\langle k+1\rangle - K H_x \hat{P}^{+}\langle k+1\rangle$$ — updated covariance

update phase

Fig. 6.6.
Summary of extended Kalman filter algorithm showing the prediction and update phases

The state vector comprises the vehicle configuration *and* the coordinates of the M landmarks that have been observed so far

$$\hat{\boldsymbol{x}} = \left(x_v, y_v, \theta_v, x_1, y_1, x_2, y_2, \cdots x_M, y_M\right)^T \in \mathbb{R}^{2M+3\times 1}$$

The estimated covariance is a $(2M+3)\times(2M+3)$ matrix and has the structure

$$\hat{\boldsymbol{P}} = \begin{pmatrix} \hat{\boldsymbol{P}}_{vv} & \hat{\boldsymbol{P}}_{vm} \\ \hat{\boldsymbol{P}}_{vm}^T & \hat{\boldsymbol{P}}_{mm} \end{pmatrix}$$

where $\hat{\boldsymbol{P}}_{vv}$ is the covariance of the vehicle pose, $\hat{\boldsymbol{P}}_{mm}$ the covariance of the map landmark positions, and $\hat{\boldsymbol{P}}_{vm}$ is the correlation between vehicle and landmark states.

Recap: SLAM using the EKF

Landmark positions and vehicle pose are now our states:

$$\hat{\boldsymbol{x}} = \left(x_v, y_v, \theta_v, x_1, y_1, x_2, y_2, \cdots x_M, y_M\right)^T \in \mathbb{R}^{2M+3\times 1}$$

High level idea: Combine localization with known map
and mapping with known vehicle pose

1. Write down joint motion model, derive Jacobian

2a. If new landmark detected, expand the state
"Insertion" Jacobian is slightly different now
Derive this from the joint "insertion" operator

2b. Else, update the appropriate landmark
Measurement Jacobian also slightly different now
Derive this from the joint measurement model

… exercise to the reader (see ~~Ex3 Q3~~) – now in Ex4

SLAM using the EKF

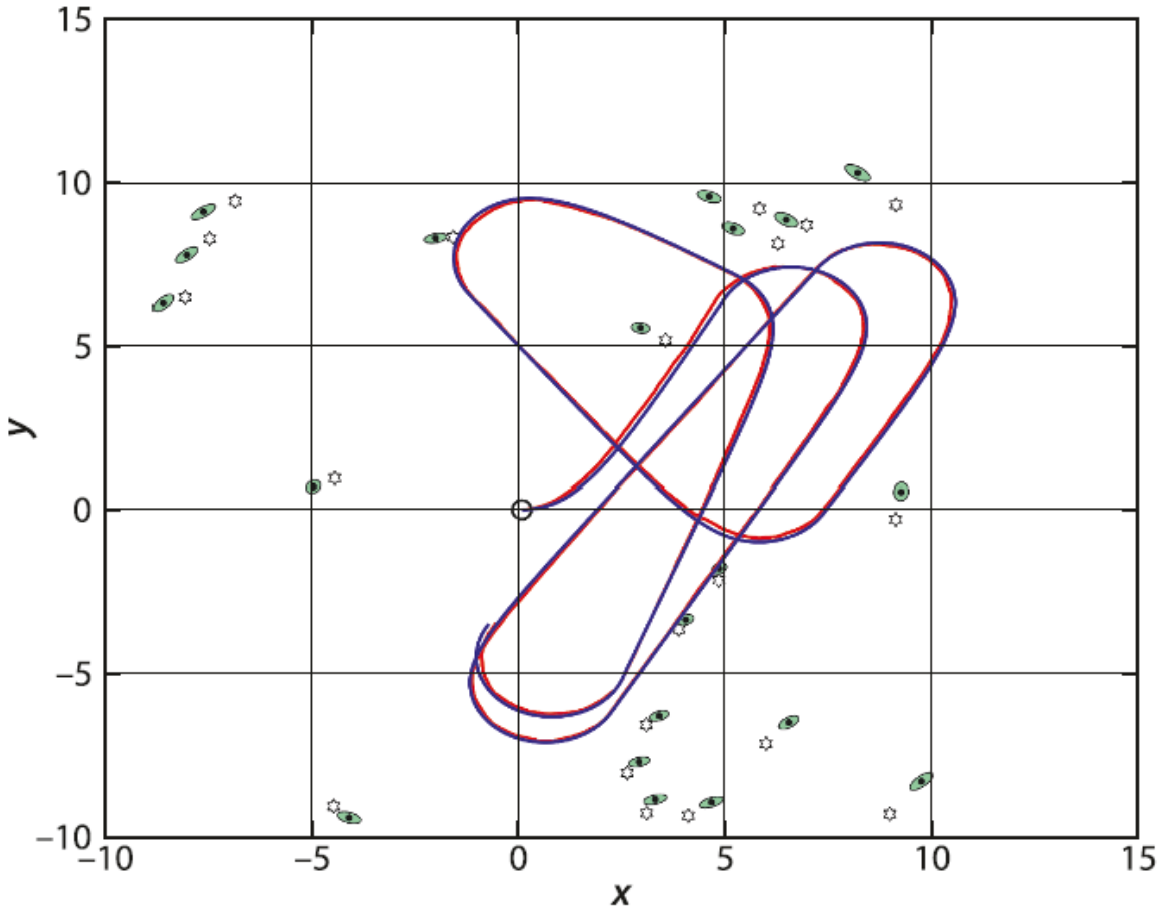
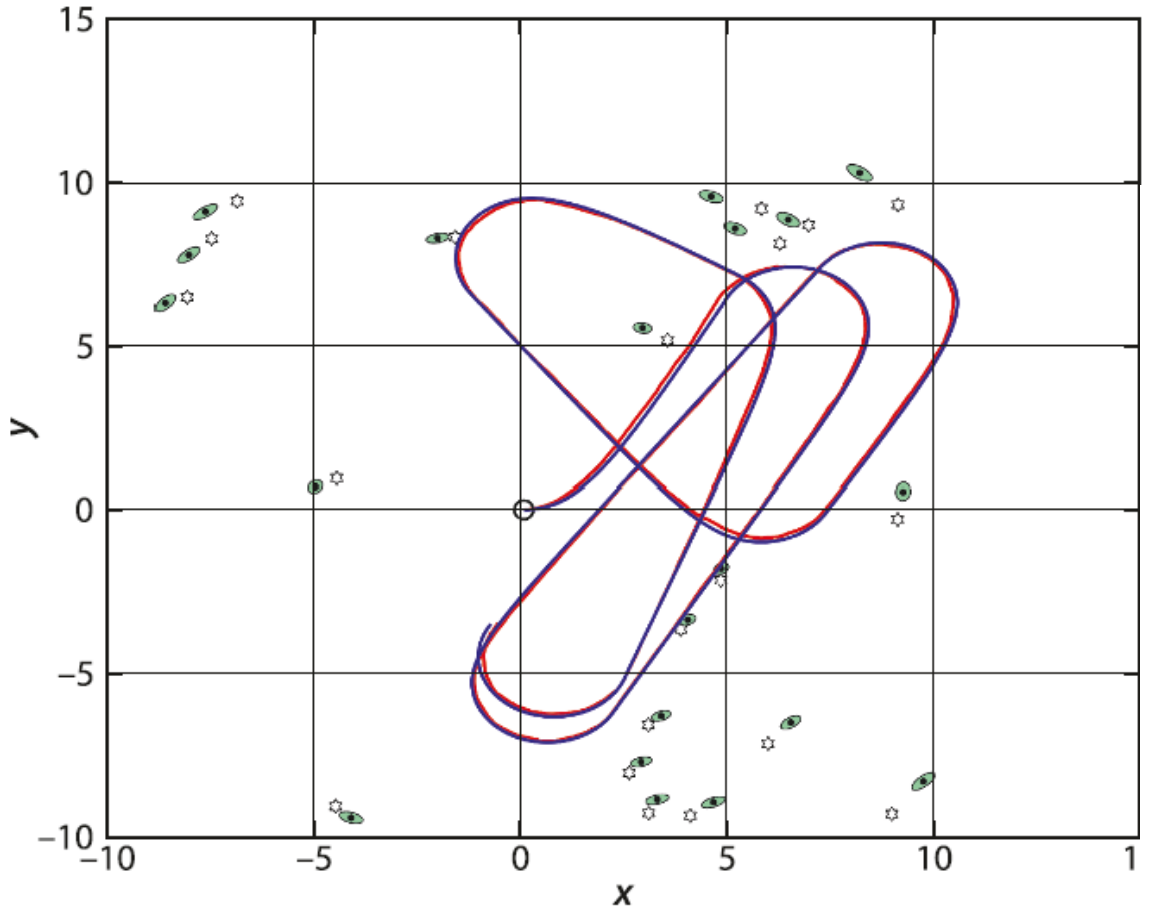
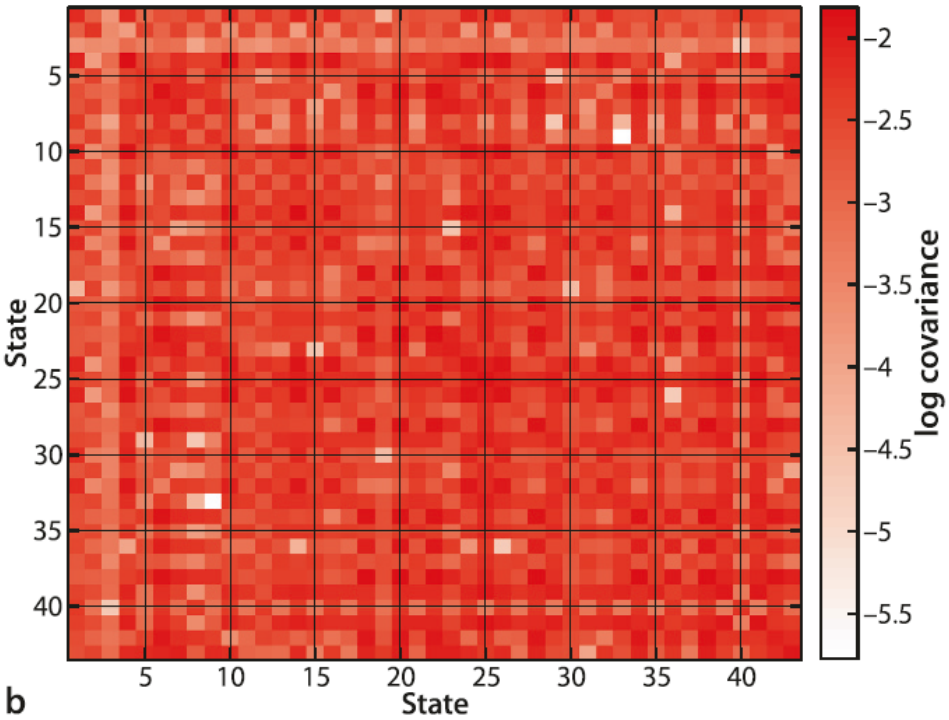


Fig. 6.11.
Simultaneous localization and mapping showing the true (*blue*) and estimated (*red*) robot path superimposed on the true map (*black ✡-marker*). The estimated map features are indicated by *black dots* with 95% confidence ellipses (*green*)

SLAM using the EKF





b

SLAM using the EKF

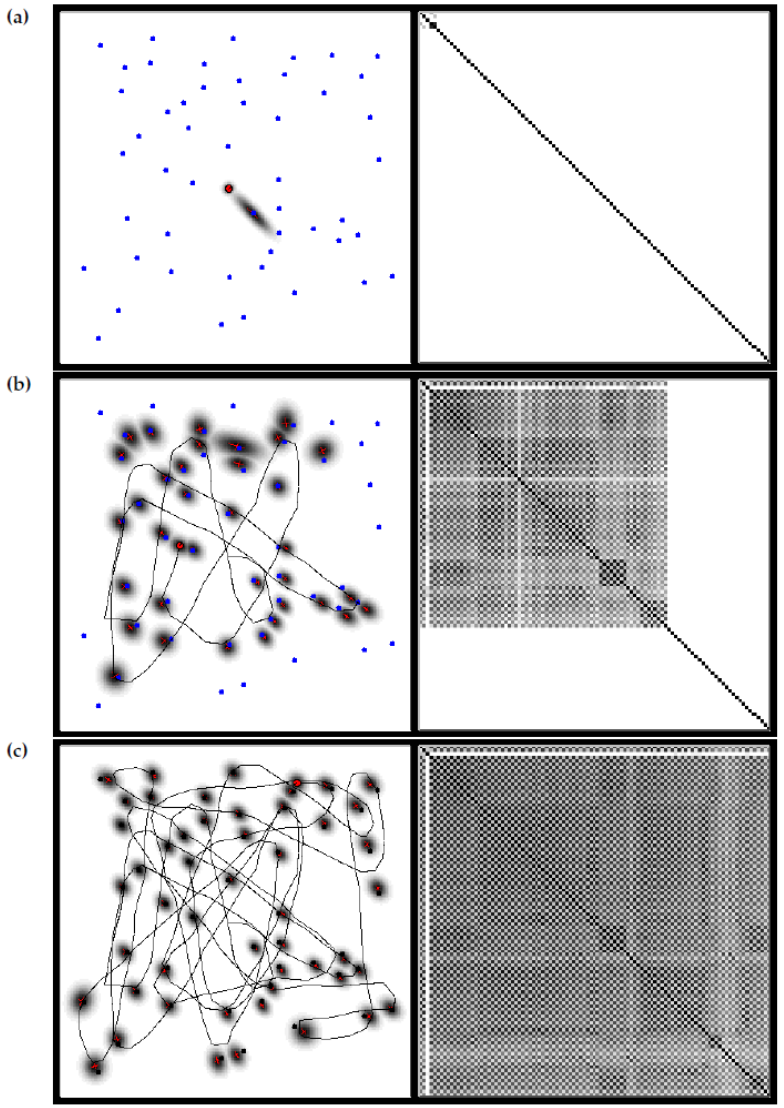


Figure 10.4 EKF SLAM with known data association in a simulated environment. The map is shown on the left, with the gray-level corresponding to the uncertainty of each landmark. The matrix on the right is the correlation matrix, which is the normalized covariance matrix of the posterior estimate. After some time, all x- and all y-coordinate estimates become fully correlated.

SLAM using the EKF

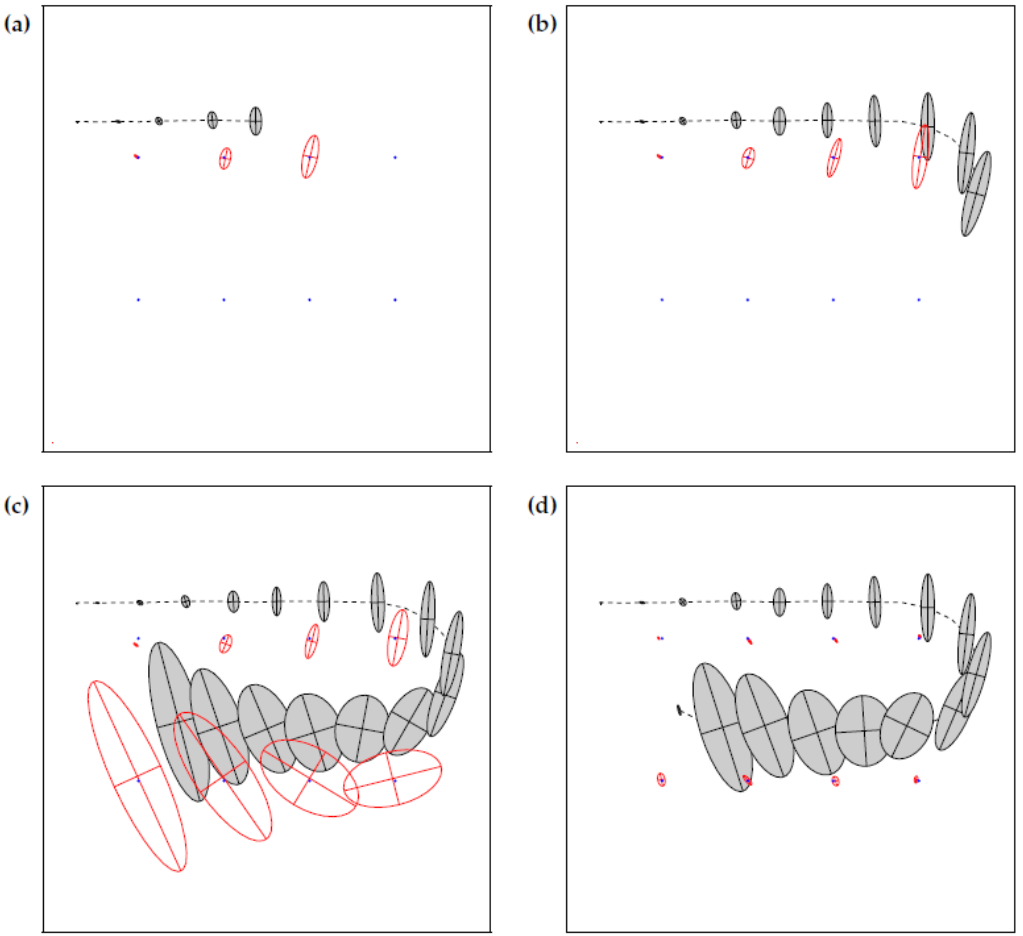
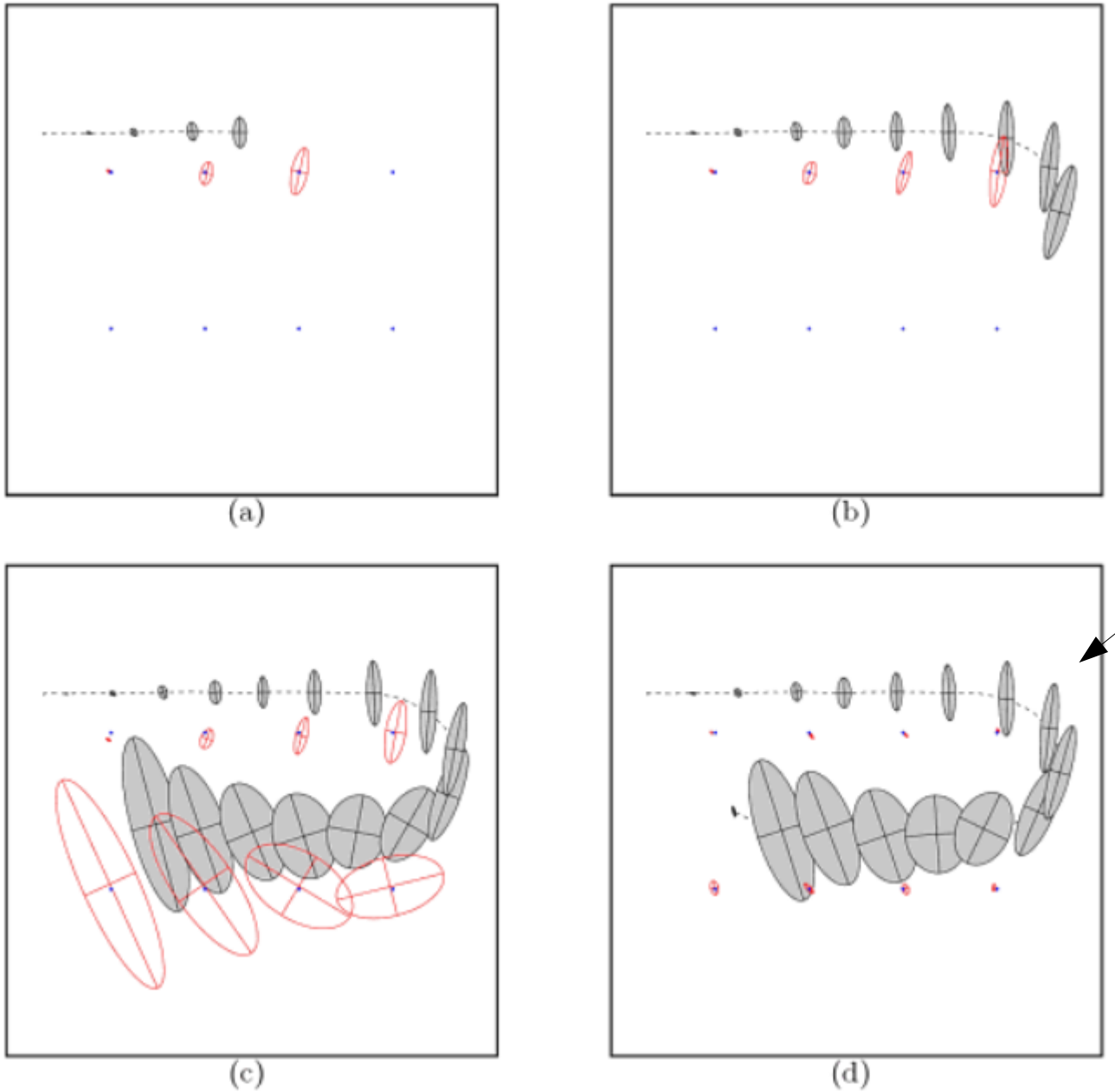


Figure 10.3 EKF applied to the online SLAM problem. The robot's path is a dotted line, and its estimates of its own position are shaded ellipses. Eight distinguishable landmarks of unknown location are shown as small dots, and their location estimates are shown as white ellipses. In (a)–(c) the robot's positional uncertainty is increasing, as is its uncertainty about the landmarks it encounters. In (d) the robot senses the first landmark again, and the uncertainty of *all* landmarks decreases, as does the uncertainty of its current pose. Image courtesy of Michael Montemerlo, Stanford University.

SLAM using the EKF

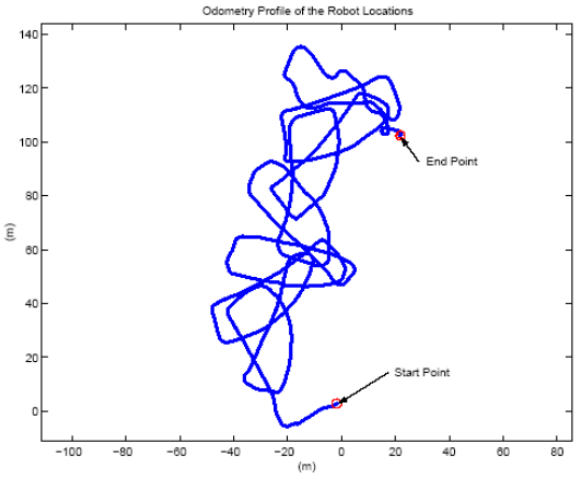


Landmark covariance drops significantly as soon as “loop closure” occurs.

SLAM using the EKF

(a) RWI B21 Mobile robot and testing environment

(b) Raw odometry



(c) Result of EKF SLAM with ground truth

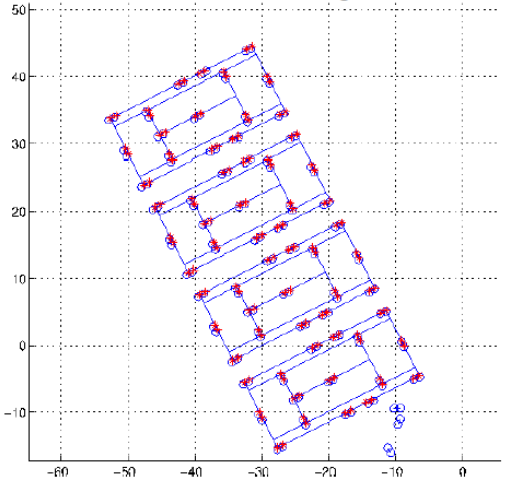


Figure 10.7 (a) The MIT B21 mobile robot in a calibrated testing facility. (b) Raw odometry of the robot, as it is manually driven through the environment. (c) The result of EKF SLAM is a highly accurate map. The image shows the estimated map overlayed on a manually constructed map. All images and results are courtesy of John Leonard and Matthew Walter, MIT.

Rao-Blackwellized SLAM

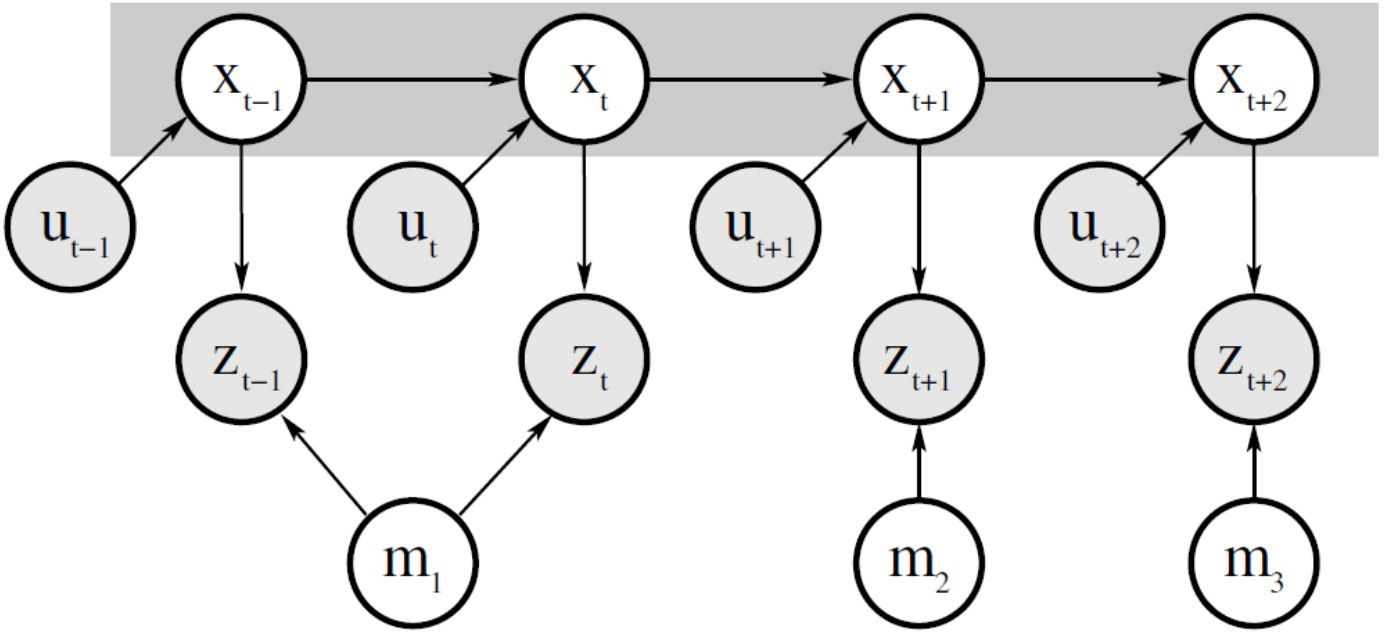
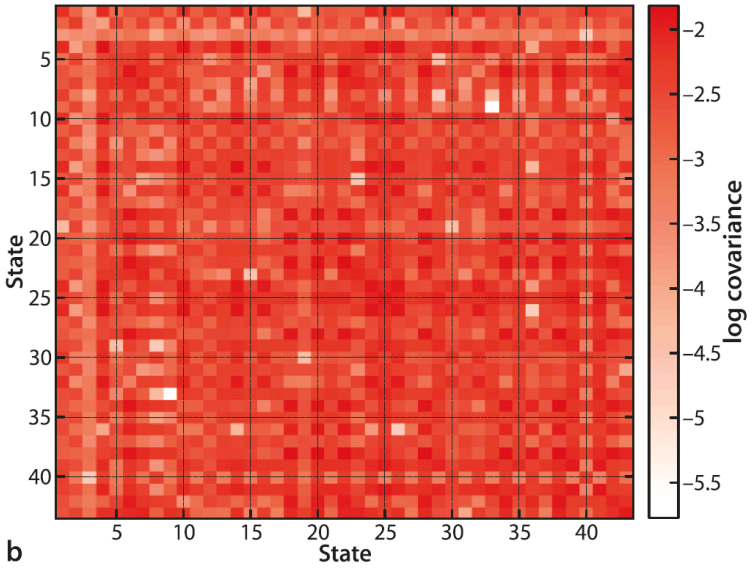


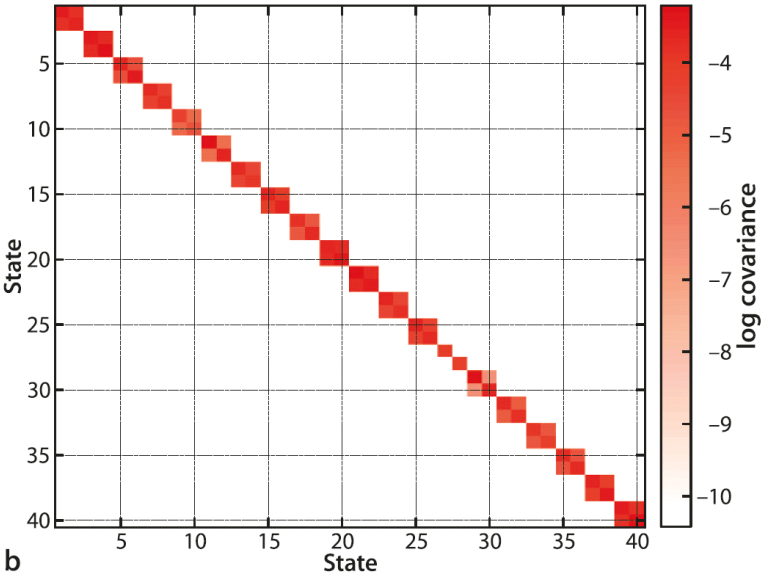
Figure 13.3 The SLAM problem depicted as Bayes network graph. The robot moves from pose x_{t-1} to pose x_{t+2}, driven by a sequence of controls. At each pose x_t it observes a nearby feature in the map $m = \{m_1, m_2, m_3\}$. This graphical network illustrates that the pose variables "separate" the individual features in the map from each other. If the poses are known, there remains no other path involving variables whose value is not known, between any two features in the map. This lack of a path renders the posterior of any two features in the map conditionally independent (given the poses).

Rao-Blackwellized SLAM

Covariance matrix
with unknown vehicle pose



Covariance matrix
with known vehicle pose



Aside: Particle filtering (sequential Monte-Carlo)

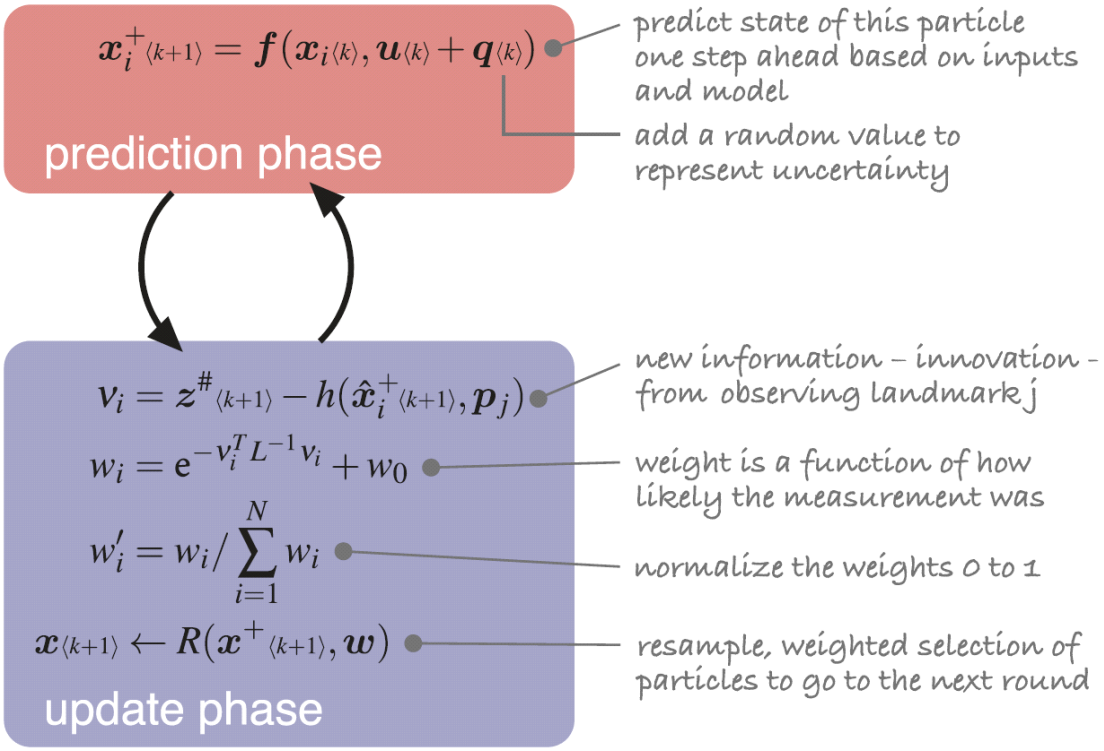


Fig. 6.19.
The particle filter estimator showing the prediction and update phases

Particle-filter localization (Monte-Carlo localization)

From
"Probabilistic Robotics"
(Ch. 7-8)
by
Sebastian Thrun,
Wolfram Burgard,
& Dieter Fox

Particle filter
(sequential Monte-Carlo);
Approximate HMM
inference

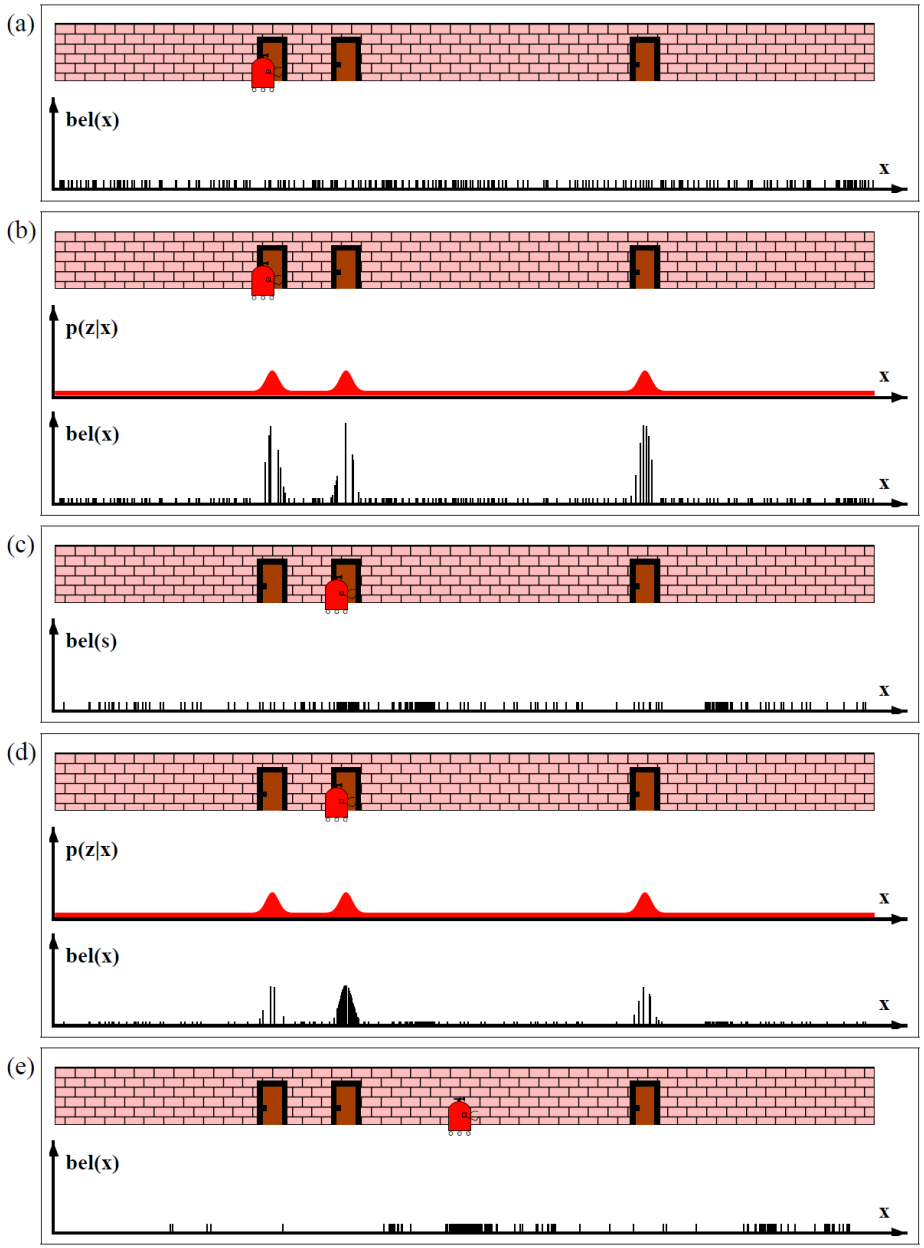
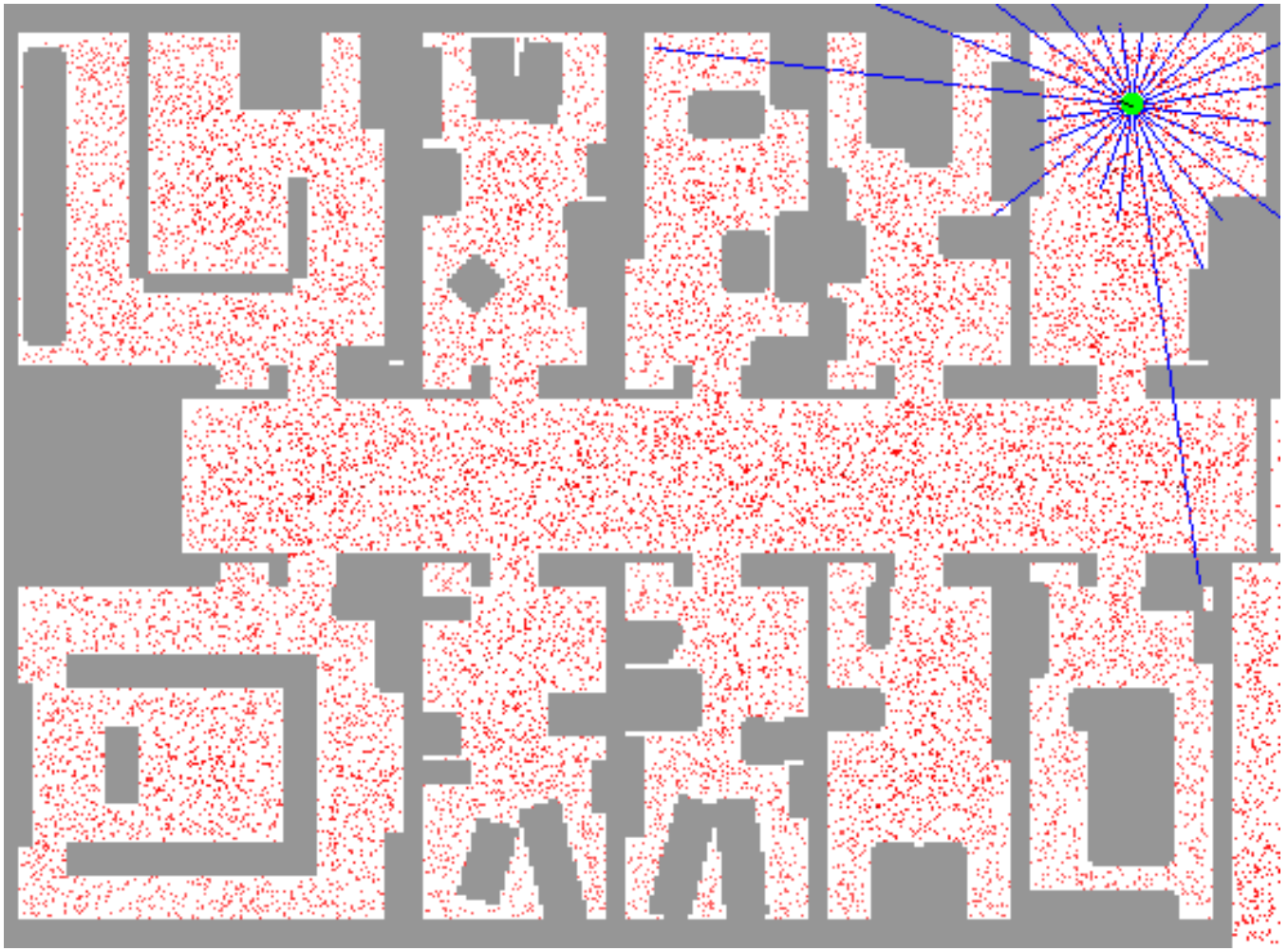


Figure 8.11 Monte Carlo Localization, a particle fi lter applied to mobile robot localization.

Particle-filter localization (Monte-Carlo localization)



Also see: https://www.youtube.com/watch?v=DZT-zNj61Jg

SLAM + particle filtering: FastSLAM

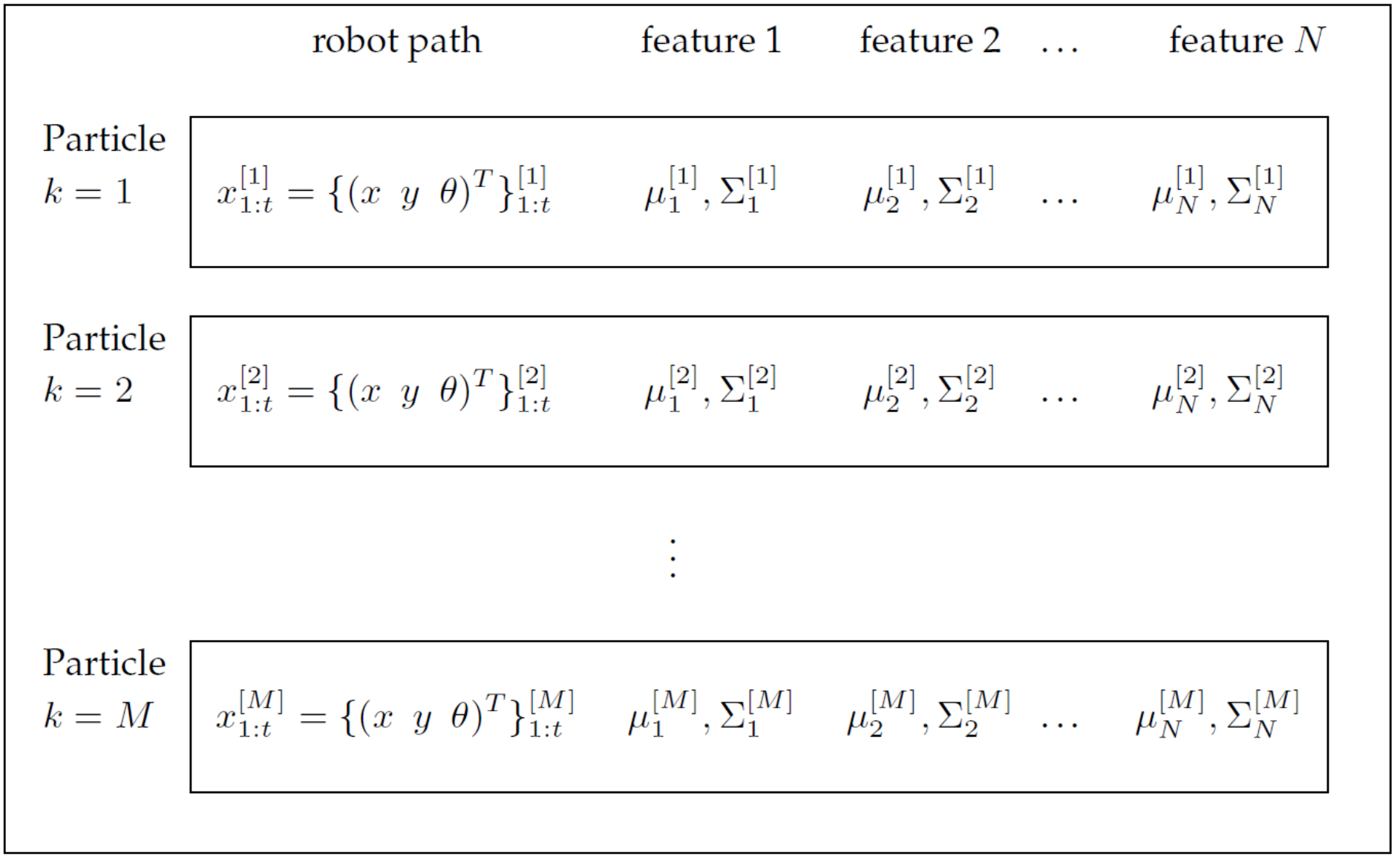


Figure 13.1 Particles in FastSLAM are composed of a path estimate and a set of estimators of individual feature locations with associated covariances.

SLAM + Particle Filtering: FastSLAM

- Do the following M times:
  - **Retrieval.** Retrieve a pose $x_{t-1}^{[k]}$ from the particle set Y_{t-1}.
  - **Prediction.** Sample a new pose $x_t^{[k]} \sim p(x_t \mid x_{t-1}^{[k]}, u_t)$.
  - **Measurement update.** For each observed feature z_t^i identify the correspondence j for the measurement z_t^i, and incorporate the measurement z_t^i into the corresponding EKF, by updating the mean $\mu_{j,t}^{[k]}$ and covariance $\Sigma_{j,t}^{[k]}$.
  - **Importance weight.** Calculate the importance weight $w^{[k]}$ for the new particle.
- **Resampling.** Sample, with replacement, M particles, where each particle is sampled with a probability proportional to $w^{[k]}$.

Figure 13.2 The basic steps of the FastSLAM algorithm.

SLAM + particle filtering: FastSLAM

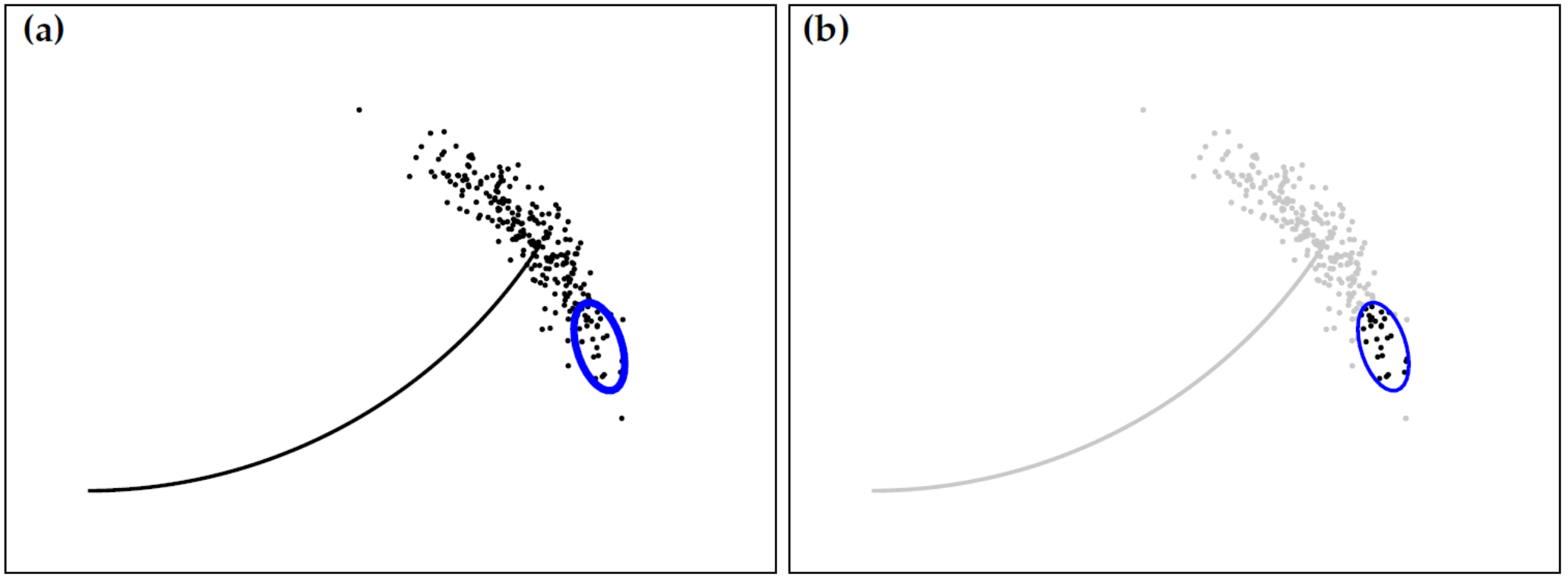


Figure 13.6 Mismatch between proposal and posterior distributions: (a) illustrates the forward samples generated by FastSLAM 1.0, and the posterior induced by the measurement (ellipse). Diagram (b) shows the sample set after the resampling step.

SLAM + particle filtering: FastSLAM

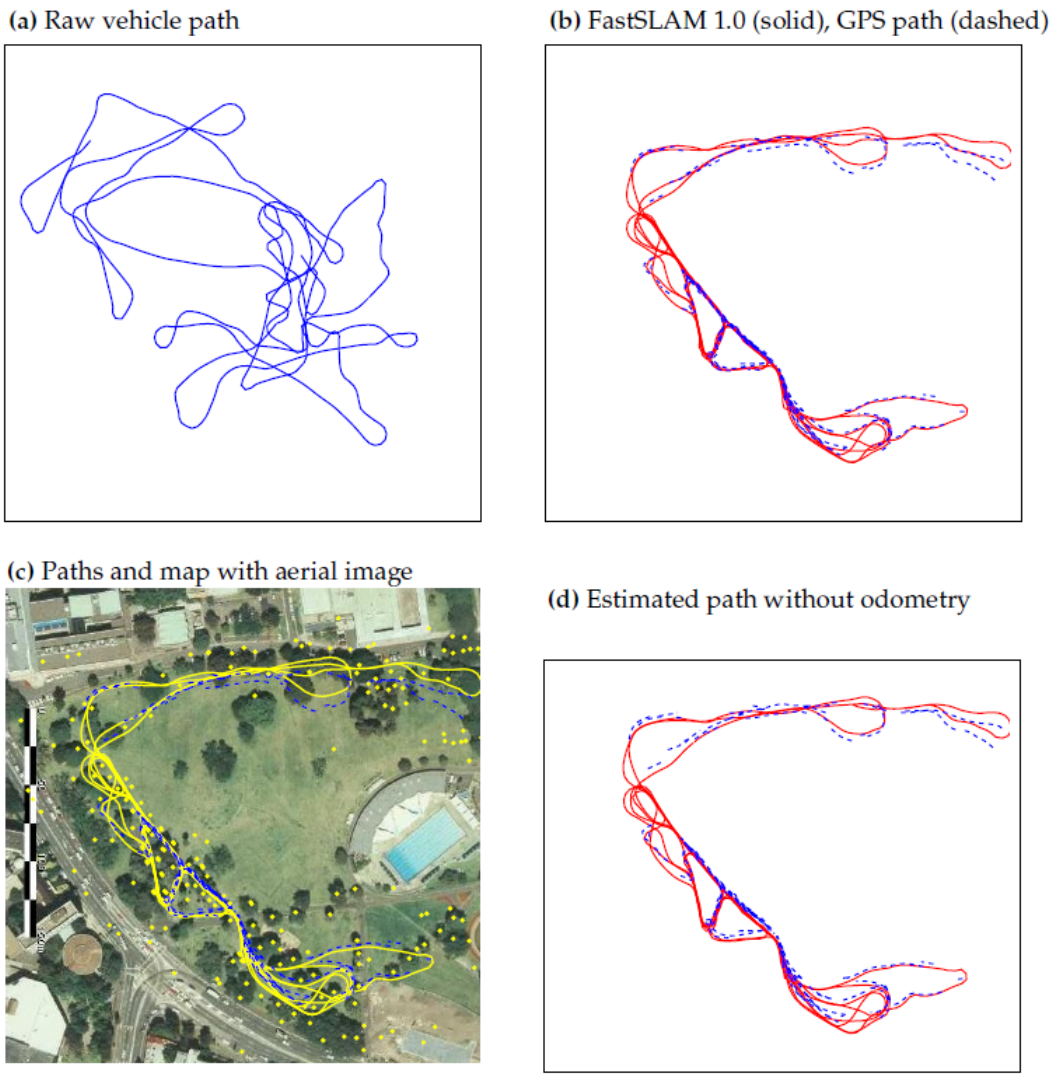


Figure 13.10 (a) Vehicle path predicted by the odometry; (b) True path (dashed line) and FastSLAM 1.0 path (solid line); (c) Victoria Park results overlayed on aerial imagery with the GPS path in blue (dashed), average FastSLAM 1.0 path in yellow (solid), and estimated features as yellow circles. (d) Victoria Park Map created without odometry information. Data and aerial image courtesy of José Guivant and Eduardo Nebot, Australian Centre for Field Robotics.

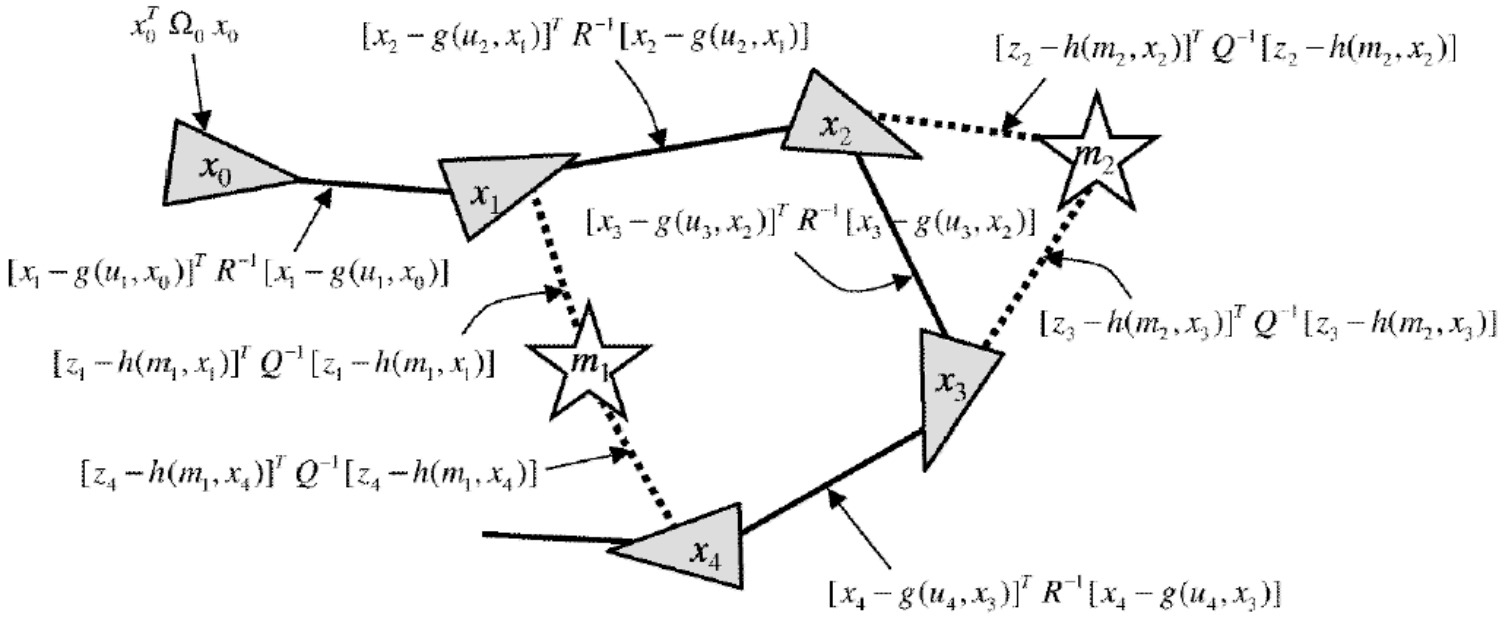
Occupancy-grid mapping / RGB-D SLAM

Fig. 6.24.
a Laser scans rendered into an occupancy grid, the area enclosed in the *green square* is dispayed in **b**. *White cells* are free space, *black cells* are occupied and *grey cells* are unknown. Grid cell size is 10 cm

Fig. 6.25.
3D point cloud created by integrating multiple scans from a vehicle-mounted scanning laser rangefinder, where the scans are in a vertical plane normal to the vehicle's forward axis. This is sometimes called a "2.5D" representation since only the front surfaces of objects are described – note the range shadows on the walls behind cars. Note also that the density of laser points is not constant across the map, for example the point density on the road surface is much greater than it is high on the walls of buildings (image courtesy Alex Stewart; Stewart 2014)

Also see: https://www.youtube.com/watch?v=58_xG8AkcaE

Pose-graph SLAM: GraphSLAM



Sum of all constraints:

$$J_{\text{GraphSLAM}} = x_0^T \Omega_0 x_0 + \sum_t [x_t - g(u_t, x_{t-1})]^T R^{-1} [x_t - g(u_t, x_{t-1})] + \sum_t [z_t - h(m_{c_t}, x_t)]^T Q^{-1} [z_t - h(m_{c_t}, x_t)]$$

Figure 11.1 GraphSLAM illustration, with 4 poses and two map features. Nodes in the graphs are robot poses and feature locations. The graph is populated by two types of edges: Solid edges link consecutive robot poses, and dashed edges link poses with features sensed while the robot assumes that pose. Each link in GraphSLAM is a non-linear quadratic constraint. Motion constraints integrate the motion model; measurement constraints the measurement model. The target function of GraphSLAM is sum of these constraints. Minimizing it yields the most likely map and the most likely robot path.

Pose-graph SLAM

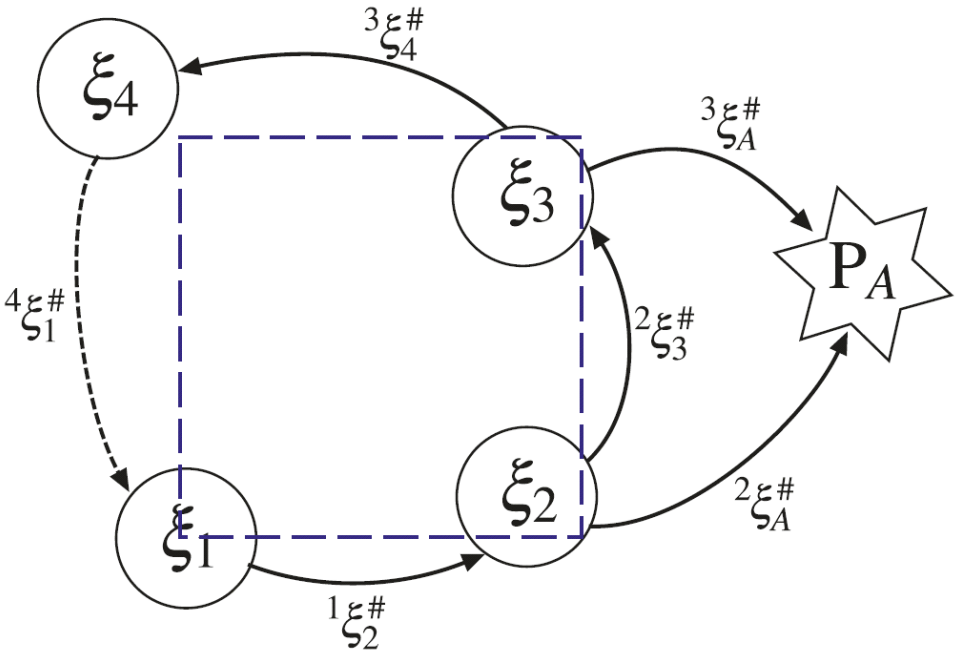


Fig. 6.14.
Pose-graph SLAM example. Places are shown as *circular nodes* and have an associated pose. Landmarks are shown as *star-shaped nodes* and have an associated position. Edges represent a measurement of a relative pose or position with respect to the node at the tail of the arrow

Pose-graph SLAM

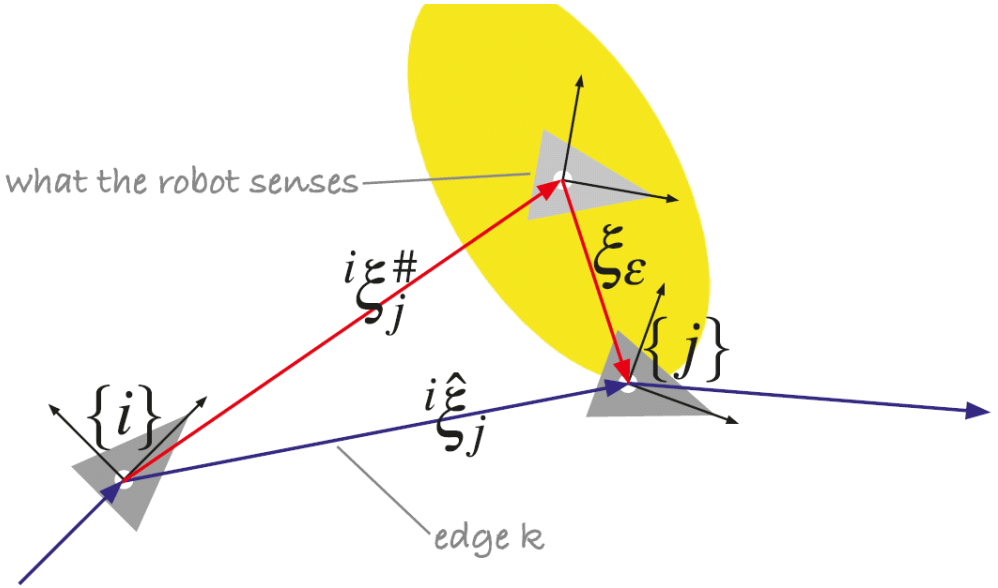


Fig. 6.15.
Pose graph notation. The *light grey robot* is the estimated pose of $\{j\}$ based on the sensor measurement ${}^{i}\xi_j^{\#}$. The *yellow ellipse* indicates uncertainty associated with that measurement

Pose-graph SLAM

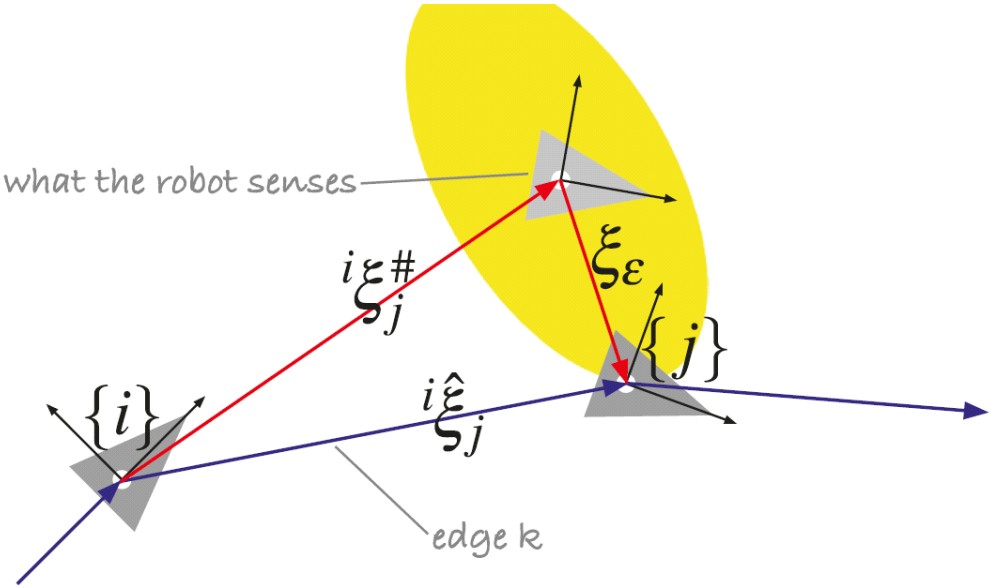


Fig. 6.15.
Pose graph notation. The *light grey robot* is the estimated pose of $\{j\}$ based on the sensor measurement ${}^i\xi_j^\#$. The *yellow ellipse* indicates uncertainty associated with that measurement

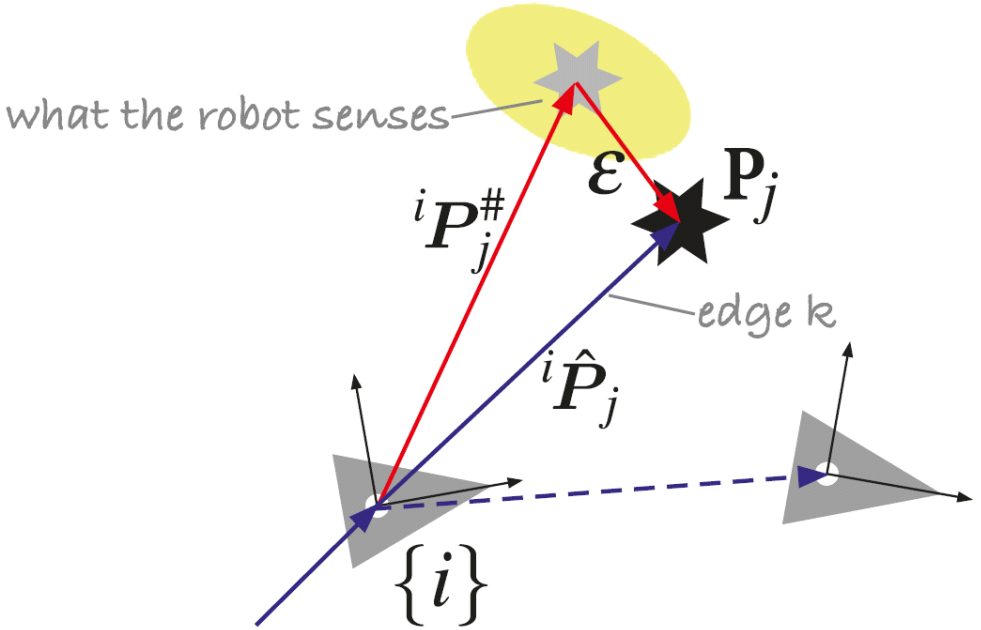
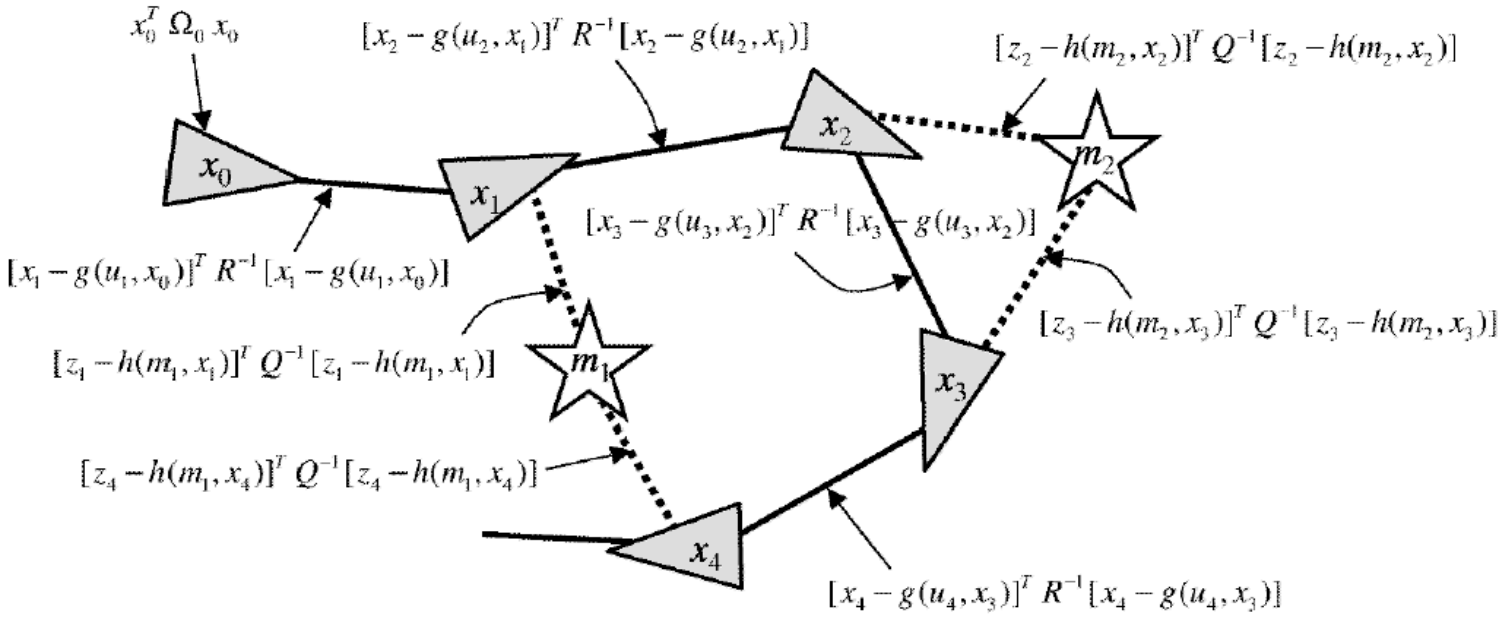


Fig. 6.18.
Notation for a pose graph with a landmark indicated by the *star-shaped symbol*. The measured position of landmark j with respect to robot pose i is ${}^i\boldsymbol{P}_j^\#$. The *yellow ellipse* indicates uncertainty associated with that measurement

Pose-graph SLAM: GraphSLAM



Sum of all constraints:

$$J_{\text{GraphSLAM}} = x_0^T \Omega_0 x_0 + \sum_t [x_t - g(u_t, x_{t-1})]^T R^{-1} [x_t - g(u_t, x_{t-1})] + \sum_t [z_t - h(m_{c_t}, x_t)]^T Q^{-1} [z_t - h(m_{c_t}, x_t)]$$

Figure 11.1 GraphSLAM illustration, with 4 poses and two map features. Nodes in the graphs are robot poses and feature locations. The graph is populated by two types of edges: Solid edges link consecutive robot poses, and dashed edges link poses with features sensed while the robot assumes that pose. Each link in GraphSLAM is a non-linear quadratic constraint. Motion constraints integrate the motion model; measurement constraints the measurement model. The target function of GraphSLAM is sum of these constraints. Minimizing it yields the most likely map and the most likely robot path.

… and more (MIT Embodied Intelligence seminar)

Upcoming Seminars

The Past, Present and Future of SLAM
10 March 2022: John Leonard (MIT)

Abstract: Simultaneous localization and mapping (SLAM) is the process of constructing a global model from local observations, acquired as a mobile robot moves through an environment. SLAM is a foundational capability for mobile robots, supporting such core functions as planning, navigation, and control, for a wide range of application domains. SLAM is one of the most deeply investigated fields in mobile robotics research, yet many open questions remain to enable the realization of robust, long-term autonomy. This talk will review the historical development of SLAM and will describe several current research projects in our group. Two key themes are increasing the expressive capacity of the environmental models used in SLAM systems (representation) and improving the performance of the algorithms used to estimate these models from data (inference). Our ultimate goal is to provide autonomous robots with a more comprehensive understanding of the world, facilitating life-long learning in complex dynamic environments.

Biography: John J. Leonard is Samuel C. Collins Professor of Mechanical and Ocean Engineering in the MIT Department of Mechanical Engineering. He is also a member of the MIT Computer Science and Artificial Intelligence Laboratory (CSAIL). His research addresses the problems of navigation and mapping for autonomous mobile robots. He holds the degrees of B.S.E.E. in Electrical Engineering and Science from the University of Pennsylvania (1987) and D.Phil. in Engineering Science from the University of Oxford (1994). He is an IEEE Fellow (2014) and an AAAS Fellow (2020). Prof. Leonard is also a Technical Advisor at Toyota Research Institute

https://ei.csail.mit.edu/seminars.html

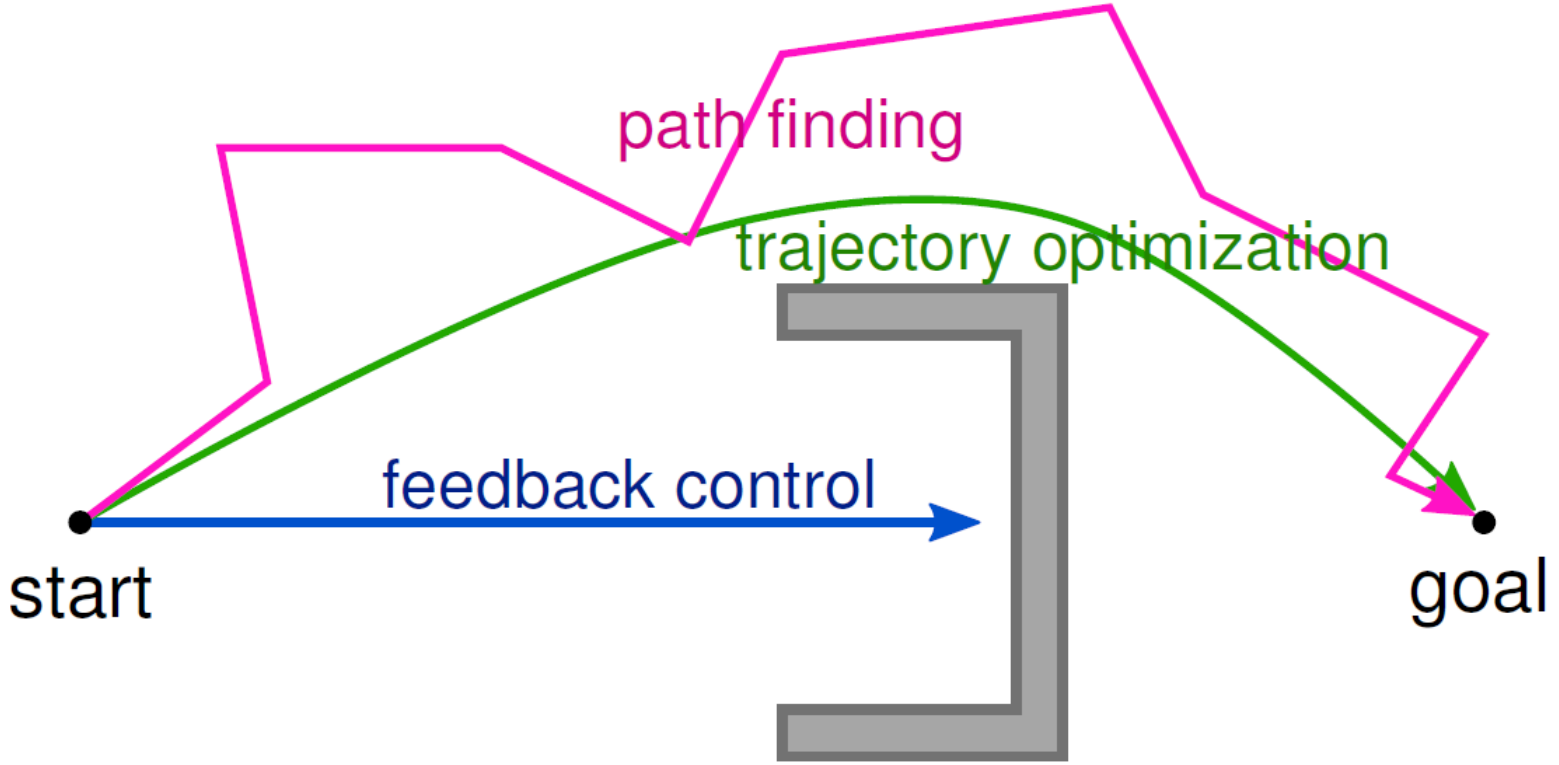
Outline

✔ MOAR SLAM
- Rao-Blackwellized SLAM (e.g., FastSLAM)
  - with particle filtering / sequential Monte-Carlo
- Occupancy-grid mapping and SLAM
- RGB-D SLAM
- Pose-graph SLAM (e.g., GraphSLAM)

Trajectory optimization
- Direct transcription
- Direct shooting
- Model-predictive control (MPC)

Linear control
- Linear-quadratic regulator (LQR)
- Time-varying LQR
- Linear-quadratic-Gaussian (LQG)

Feedback control, path finding, trajectory optim.



- Feedback Control: E.g., $q_{t+1} = q_t + J^\sharp(y^* - \phi(q_t))$
- Trajectory Optimization: $\text{argmin}_{q_{0:T}} f(q_{0:T})$
- Path Finding: Find some $q_{0:T}$ with only valid configurations

Trajectory optimization

$$\min_{\mathbf{x}[\cdot],\mathbf{u}[\cdot]} \quad \ell_f(\mathbf{x}[N]) + \sum_{n_0}^{N-1} \ell(\mathbf{x}[n], \mathbf{u}[n])$$

Trajectory optimization

10.2.1 Direct Transcription

For instance, let us start by writing both $\mathbf{u}[\cdot]$ and $\mathbf{x}[\cdot]$ as decision variables. Then we can write:

$$\begin{aligned} \min_{\mathbf{x}[\cdot],\mathbf{u}[\cdot]} \quad & \ell_f(\mathbf{x}[N]) + \sum_{n_0}^{N-1} \ell(\mathbf{x}[n], \mathbf{u}[n]) \\ \text{subject to} \quad & \mathbf{x}[n+1] = \mathbf{A}\mathbf{x}[n] + \mathbf{B}\mathbf{u}[n], \quad \forall n \in [0, N-1] \\ & \mathbf{x}[0] = \mathbf{x}_0 \\ & + \text{additional constraints.} \end{aligned}$$

We call this modeling choice -- adding $\mathbf{x}[\cdot]$ as decision variables and modeling the discrete dynamics as explicit constraints -- the "*direct transcription*".

Trajectory optimization

10.2.2 Direct Shooting

The savvy reader might have noticed that adding $\mathbf{x}[\cdot]$ as decision variables was not strictly necessary. If we know $\mathbf{x}[0]$ and we know $\mathbf{u}[\cdot]$, then we should be able to solve for $\mathbf{x}[n]$ using forward simulation. For our discrete-time linear systems, this is particularly nice:

$$\begin{aligned}\mathbf{x}[1] =& \mathbf{A}\mathbf{x}[0] + \mathbf{B}\mathbf{u}[0]\\ \mathbf{x}[2] =& \mathbf{A}(\mathbf{A}\mathbf{x}[0] + \mathbf{B}\mathbf{u}[0]) + \mathbf{B}\mathbf{u}[1]\\ \mathbf{x}[n] =& \mathbf{A}^n\mathbf{x}[0] + \sum_{k=0}^{n-1} \mathbf{A}^{n-1-k}\mathbf{B}\mathbf{u}[k].\end{aligned}$$

Trajectory optimization

10.2.2 Direct Shooting

The savvy reader might have noticed that adding $\mathbf{x}[\cdot]$ as decision variables was not strictly necessary. If we know $\mathbf{x}[0]$ and we know $\mathbf{u}[\cdot]$, then we should be able to solve for $\mathbf{x}[n]$ using forward simulation. For our discrete-time linear systems, this is particularly nice:

$$\begin{aligned}\mathbf{x}[1] =& \mathbf{A}\mathbf{x}[0] + \mathbf{B}\mathbf{u}[0] \\ \mathbf{x}[2] =& \mathbf{A}(\mathbf{A}\mathbf{x}[0] + \mathbf{B}\mathbf{u}[0]) + \mathbf{B}\mathbf{u}[1] \\ \mathbf{x}[n] =& \mathbf{A}^n\mathbf{x}[0] + \sum_{k=0}^{n-1} \mathbf{A}^{n-1-k}\mathbf{B}\mathbf{u}[k].\end{aligned}$$

Seems like a good idea, but can be numerically unstable

Trajectory optimization

10.3.1 Direct Transcription and Direct Shooting

The formulations that we wrote for direct transcription and direct shooting above are still valid when the dynamics are nonlinear, it's just that the resulting problem is nonconvex. For instance, the direct transcription for discrete-time systems becomes the more general:

$$\begin{aligned} \min_{\mathbf{x}[\cdot],\mathbf{u}[\cdot]} \quad & \ell_f(\mathbf{x}[N]) + \sum_{n_0}^{N-1} \ell(\mathbf{x}[n], \mathbf{u}[n]) \\ \text{subject to} \quad & \mathbf{x}[n+1] = \mathbf{f}(\mathbf{x}[n], \mathbf{u}[n]), \quad \forall n \in [0, N-1] \\ & \mathbf{x}[0] = \mathbf{x}_0 \\ & + \text{additional constraints.} \end{aligned}$$

Direct shooting still works, too, since on each iteration of the algorithm we can compute $\mathbf{x}[n]$ given $\mathbf{x}[0]$ and $\mathbf{u}[\cdot]$ by forward simulation.

Trajectory optimization

10.3.1 Direct Transcription and Direct Shooting

The formulations that we wrote for direct transcription and direct shooting above are still valid when the dynamics are nonlinear, it's just that the resulting problem is nonconvex. For instance, the direct transcription for discrete-time systems becomes the more general:

$$\begin{aligned} \min_{\mathbf{x}[\cdot],\mathbf{u}[\cdot]} \quad & \ell_f(\mathbf{x}[N]) + \sum_{n_0}^{N-1} \ell(\mathbf{x}[n], \mathbf{u}[n]) \\ \text{subject to} \quad & \mathbf{x}[n+1] = \mathbf{f}(\mathbf{x}[n], \mathbf{u}[n]), \quad \forall n \in [0, N-1] \\ & \mathbf{x}[0] = \mathbf{x}_0 \\ & + \text{additional constraints.} \end{aligned}$$

Direct shooting still works, too, since on each iteration of the algorithm we can compute $\mathbf{x}[n]$ given $\mathbf{x}[0]$ and $\mathbf{u}[\cdot]$ by forward simulation.

MATLAB: fmincon

Trajectory optimization

10.4.2 Model-Predictive Control

The maturity, robustness, and speed of solving trajectory optimization using convex optimization leads to a beautiful idea: if we can optimize trajectories quickly enough, then we can use our trajectory optimization as a feedback policy. The recipe is simple: (1) measure the current state, (2) optimize a trajectory from the current state, (3) execute the first action from the optimized trajectory, (4) let the dynamics evolve for one step and repeat. This recipe is known as *model-predictive control* (MPC).

Trajectory optimization

10.4.2 Model-Predictive Control

The maturity, robustness, and speed of solving trajectory optimization using convex optimization leads to a beautiful idea: if we can optimize trajectories quickly enough, then we can use our trajectory optimization as a feedback policy. The recipe is simple: (1) measure the current state, (2) optimize a trajectory from the current state, (3) execute the first action from the optimized trajectory, (4) let the dynamics evolve for one step and repeat. This recipe is known as *model-predictive control* (MPC).

Receding-horizon MPC:

1. Optimize for next T time steps
2. Take first control / action
3. Repeat (go to step 1)

Outline

✔ MOAR SLAM
- Rao-Blackwellized SLAM (e.g., FastSLAM)
  - with particle filtering / sequential Monte-Carlo
- Occupancy-grid mapping and SLAM
- RGB-D SLAM
- Pose-graph SLAM (e.g., GraphSLAM)

✔ Trajectory optimization
- Direct transcription
- Direct shooting
- Model-predictive control (MPC)

Linear control
- Linear-quadratic regulator (LQR)
- Time-varying LQR
- Linear-quadratic-Gaussian (LQG)

Linear-quadratic regulator (LQR)

Consider the discrete time dynamics:

$$\mathbf{x}[n+1] = \mathbf{A}\mathbf{x}[n] + \mathbf{B}\mathbf{u}[n],$$

Linear-quadratic regulator (LQR)

Consider the discrete time dynamics:

$$\mathbf{x}[n+1] = \mathbf{A}\mathbf{x}[n] + \mathbf{B}\mathbf{u}[n],$$

and we wish to minimize

$$\min \sum_{n=0}^{N-1} \mathbf{x}^T[n]\mathbf{Q}\mathbf{x}[n] + \mathbf{u}^T[n]\mathbf{R}\mathbf{u}[n], \qquad \mathbf{Q} = \mathbf{Q}^T \succeq 0, \mathbf{R} = \mathbf{R}^T \succ 0.$$

Linear-quadratic regulator (LQR)

Consider the discrete time dynamics:

$$\mathbf{x}[n+1] = \mathbf{A}\mathbf{x}[n] + \mathbf{B}\mathbf{u}[n],$$

and we wish to minimize

$$\min \sum_{n=0}^{N-1} \mathbf{x}^T[n]\mathbf{Q}\mathbf{x}[n] + \mathbf{u}^T[n]\mathbf{R}\mathbf{u}[n], \qquad \mathbf{Q} = \mathbf{Q}^T \succeq 0, \mathbf{R} = \mathbf{R}^T \succ 0.$$

The cost-to-go is given by

$$J(\mathbf{x}, n-1) = \min_{\mathbf{u}} \mathbf{x}^T\mathbf{Q}\mathbf{x} + \mathbf{u}^T\mathbf{R}\mathbf{u} + J(\mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u}, n).$$

Linear-quadratic regulator (LQR)

Consider the discrete time dynamics:

$$\mathbf{x}[n+1] = \mathbf{A}\mathbf{x}[n] + \mathbf{B}\mathbf{u}[n],$$

and we wish to minimize

$$\min \sum_{n=0}^{N-1} \mathbf{x}^T[n]\mathbf{Q}\mathbf{x}[n] + \mathbf{u}^T[n]\mathbf{R}\mathbf{u}[n], \qquad \mathbf{Q} = \mathbf{Q}^T \succeq 0, \mathbf{R} = \mathbf{R}^T \succ 0.$$

The cost-to-go is given by

$$J(\mathbf{x}, n-1) = \min_{\mathbf{u}} \mathbf{x}^T\mathbf{Q}\mathbf{x} + \mathbf{u}^T\mathbf{R}\mathbf{u} + J(\mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u}, n).$$

If you are familiar with reinforcement learning,
this is the Bellman equation
(min vs. max, cost vs. reward,
cost-to-go vs. value function)

Linear-quadratic regulator (LQR)

Consider the discrete time dynamics:

$$\mathbf{x}[n+1] = \mathbf{A}\mathbf{x}[n] + \mathbf{B}\mathbf{u}[n],$$

and we wish to minimize

$$\min \sum_{n=0}^{N-1} \mathbf{x}^T[n]\mathbf{Q}\mathbf{x}[n] + \mathbf{u}^T[n]\mathbf{R}\mathbf{u}[n], \qquad \mathbf{Q} = \mathbf{Q}^T \succeq 0, \mathbf{R} = \mathbf{R}^T \succ 0.$$

The cost-to-go is given by

$$J(\mathbf{x}, n-1) = \min_{\mathbf{u}} \mathbf{x}^T\mathbf{Q}\mathbf{x} + \mathbf{u}^T\mathbf{R}\mathbf{u} + J(\mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u}, n).$$

If we once again take

$$J(\mathbf{x}, n) = \mathbf{x}^T\mathbf{S}[\mathbf{n}]\mathbf{x}, \quad \mathbf{S}[n] = \mathbf{S}^T[n] \succ 0,$$

(it is possible to show that the optimal cost-to-go function has this quadratic form)

Linear-quadratic regulator (LQR)

Consider the discrete time dynamics:

$$\mathbf{x}[n+1] = \mathbf{A}\mathbf{x}[n] + \mathbf{B}\mathbf{u}[n],$$

and we wish to minimize

$$\min \sum_{n=0}^{N-1} \mathbf{x}^T[n]\mathbf{Q}\mathbf{x}[n] + \mathbf{u}^T[n]\mathbf{R}\mathbf{u}[n], \qquad \mathbf{Q} = \mathbf{Q}^T \succeq 0, \mathbf{R} = \mathbf{R}^T \succ 0.$$

The cost-to-go is given by

$$J(\mathbf{x}, n-1) = \min_{\mathbf{u}} \mathbf{x}^T\mathbf{Q}\mathbf{x} + \mathbf{u}^T\mathbf{R}\mathbf{u} + J(\mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u}, n).$$

If we once again take

$$J(\mathbf{x}, n) = \mathbf{x}^T\mathbf{S}[\mathbf{n}]\mathbf{x}, \quad \mathbf{S}[n] = \mathbf{S}^T[n] \succ 0,$$

then we have

$$\mathbf{u}^*[n] = -\mathbf{K}[\mathbf{n}]\mathbf{x}[n] = -(\mathbf{R} + \mathbf{B}^T\mathbf{S}[n]\mathbf{B})^{-1}\mathbf{B}^T\mathbf{S}[n]\mathbf{A}\mathbf{x}[n],$$

Linear-quadratic regulator (LQR)

Consider the discrete time dynamics:

$$\mathbf{x}[n+1] = \mathbf{A}\mathbf{x}[n] + \mathbf{B}\mathbf{u}[n],$$

and we wish to minimize

$$\min \sum_{n=0}^{N-1} \mathbf{x}^T[n]\mathbf{Q}\mathbf{x}[n] + \mathbf{u}^T[n]\mathbf{R}\mathbf{u}[n], \qquad \mathbf{Q} = \mathbf{Q}^T \succeq 0, \mathbf{R} = \mathbf{R}^T \succ 0.$$

The cost-to-go is given by

$$J(\mathbf{x}, n-1) = \min_{\mathbf{u}} \mathbf{x}^T\mathbf{Q}\mathbf{x} + \mathbf{u}^T\mathbf{R}\mathbf{u} + J(\mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u}, n).$$

If we once again take

$$J(\mathbf{x}, n) = \mathbf{x}^T\mathbf{S}[\mathbf{n}]\mathbf{x}, \quad \mathbf{S}[n] = \mathbf{S}^T[n] \succ 0,$$

then we have

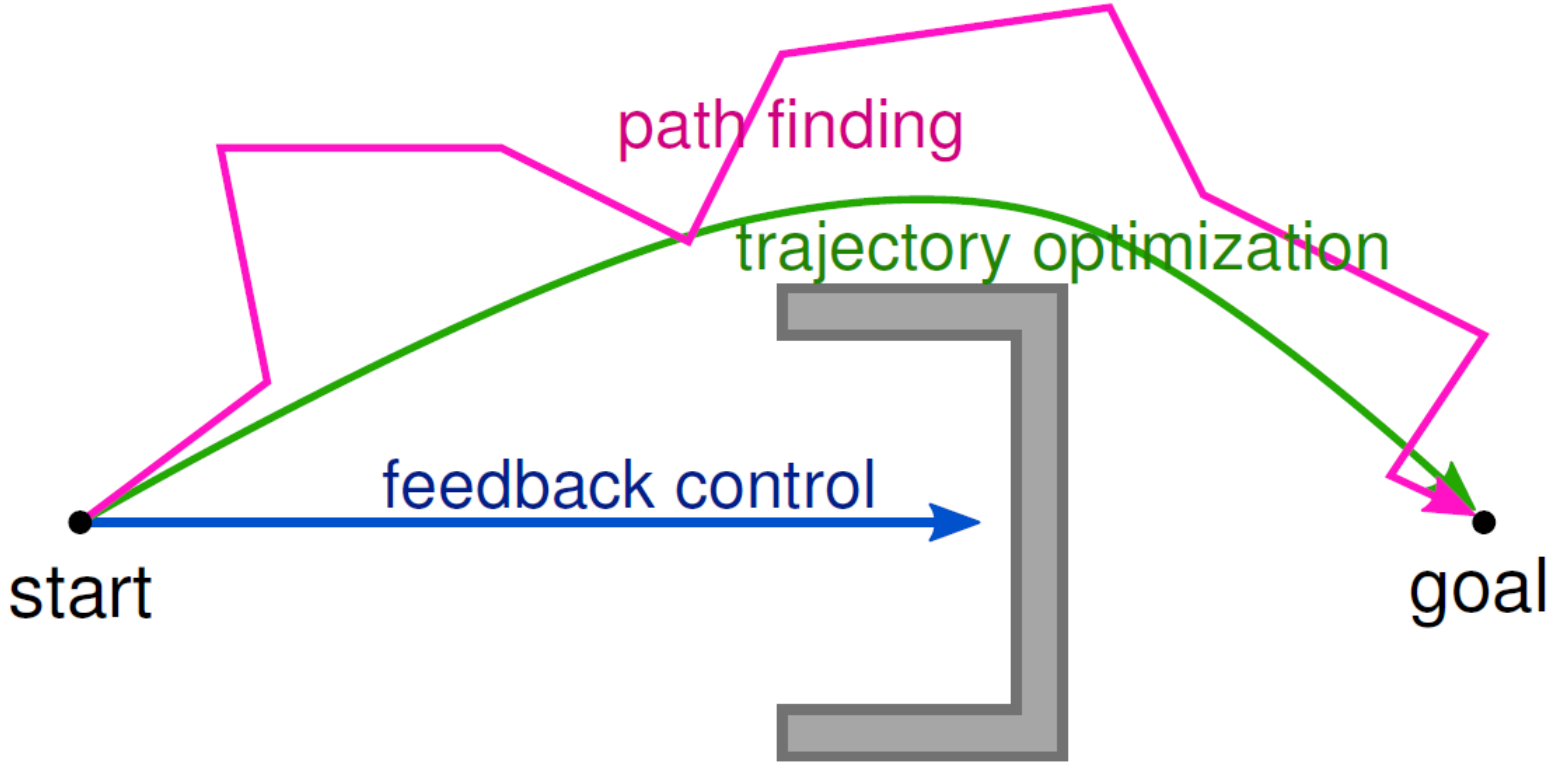
$$\mathbf{u}^*[n] = -\mathbf{K}[\mathbf{n}]\mathbf{x}[n] = -(\mathbf{R} + \mathbf{B}^T\mathbf{S}[n]\mathbf{B})^{-1}\mathbf{B}^T\mathbf{S}[n]\mathbf{A}\mathbf{x}[n],$$

yielding

$$\mathbf{S}[n-1] = \mathbf{Q} + \mathbf{A}^T\mathbf{S}[n]\mathbf{A} - (\mathbf{A}^T\mathbf{S}[n]\mathbf{B})(\mathbf{R} + \mathbf{B}^T\mathbf{S}[n]\mathbf{B})^{-1}(\mathbf{B}^T\mathbf{S}[n]\mathbf{A}), \quad \mathbf{S}[N] = 0,$$

which is the famous *Riccati difference equation*.

Feedback control, path finding, trajectory optim.



- Feedback Control: E.g., $q_{t+1} = q_t + J^\sharp(y^* - \phi(q_t))$
- Trajectory Optimization: $\text{argmin}_{q_{0:T}} f(q_{0:T})$
- Path Finding: Find some $q_{0:T}$ with only valid configurations

LQR variants

- “The LQR algorithm is essentially an automated way of finding an appropriate state-feedback controller.”

- Common application: Stabilize about a fixed point (e.g., upright configuration of inverted pendulum)

LQR variants

- “The LQR algorithm is essentially an automated way of finding an appropriate state-feedback controller.”

- Common application: Stabilize about a fixed point (e.g., upright configuration of inverted pendulum)

- Only applicable for linear systems without constraints
  - Nonlinear: Linearize!

LQR variants

- “The LQR algorithm is essentially an automated way of finding an appropriate state-feedback controller.”

- Common application: Stabilize about a fixed point (e.g., upright configuration of inverted pendulum)

- Only applicable for linear systems without constraints
  - Nonlinear: Linearize!
  - Constraints:
    - Use trajectory optimization to find trajectory
    - Stabilize along sequence of trajectory points
    - Time-varying LQR

Linear-quadratic Gaussian control (LQG)

The discrete-time linear system equations are

$$\mathbf{x}_{i+1} = A_i \mathbf{x}_i + B_i \mathbf{u}_i + \mathbf{v}_i,$$

$$\mathbf{y}_i = C_i \mathbf{x}_i + \mathbf{w}_i.$$

Here i represents the discrete time index and $\mathbf{v}_i, \mathbf{w}_i$ represent discrete-time Gaussian white noise processes with covariance matrices V_i, W_i, respectively, and are independent of each other.

Linear-quadratic Gaussian control (LQG)

The discrete-time linear system equations are

$$\mathbf{x}_{i+1} = A_i \mathbf{x}_i + B_i \mathbf{u}_i + \mathbf{v}_i,$$

$$\mathbf{y}_i = C_i \mathbf{x}_i + \mathbf{w}_i.$$

Here i represents the discrete time index and $\mathbf{v}_i, \mathbf{w}_i$ represent discrete-time Gaussian white noise processes with covariance matrices V_i, W_i, respectively, and are independent of each other.

The quadratic cost function to be minimized is

$$J = \mathbb{E}\left[\mathbf{x}_N^{\mathrm{T}} F \mathbf{x}_N + \sum_{i=0}^{N-1} (\mathbf{x}_i^{\mathrm{T}} Q_i \mathbf{x}_i + \mathbf{u}_i^{\mathrm{T}} R_i \mathbf{u}_i)\right],$$

$$F \geq 0, Q_i \geq 0, R_i > 0.$$

Linear-quadratic Gaussian control (LQG)

The discrete-time linear system equations are

$$\mathbf{x}_{i+1} = A_i \mathbf{x}_i + B_i \mathbf{u}_i + \mathbf{v}_i,$$

$$\mathbf{y}_i = C_i \mathbf{x}_i + \mathbf{w}_i.$$

Here i represents the discrete time index and $\mathbf{v}_i, \mathbf{w}_i$ represent discrete-time Gaussian white noise processes with covariance matrices V_i, W_i, respectively, and are independent of each other.

The quadratic cost function to be minimized is

$$J = \mathbb{E}\left[\mathbf{x}_N^{\mathrm{T}} F \mathbf{x}_N + \sum_{i=0}^{N-1} (\mathbf{x}_i^{\mathrm{T}} Q_i \mathbf{x}_i + \mathbf{u}_i^{\mathrm{T}} R_i \mathbf{u}_i)\right],$$

$$F \geq 0, Q_i \geq 0, R_i > 0.$$

The discrete-time LQG controller is

$$\hat{\mathbf{x}}_{i+1} = A_i \hat{\mathbf{x}}_i + B_i \mathbf{u}_i + L_{i+1}\left(\mathbf{y}_{i+1} - C_{i+1}\left\{A_i \hat{\mathbf{x}}_i + B_i \mathbf{u}_i\right\}\right), \qquad \hat{\mathbf{x}}_0 = \mathbb{E}[\mathbf{x}_0],$$

$$\mathbf{u}_i = -K_i \hat{\mathbf{x}}_i.$$

and $\hat{\mathbf{x}}_i$ corresponds to the predictive estimate $\hat{\mathbf{x}}_i = \mathbb{E}[\mathbf{x}_i | \mathbf{y}^i, \mathbf{u}^{i-1}]$.

Linear-quadratic Gaussian control (LQG)

The discrete-time linear system equations are

$$\mathbf{x}_{i+1} = A_i \mathbf{x}_i + B_i \mathbf{u}_i + \mathbf{v}_i,$$

$$\mathbf{y}_i = C_i \mathbf{x}_i + \mathbf{w}_i.$$

Here i represents the discrete time index and $\mathbf{v}_i, \mathbf{w}_i$ represent discrete-time Gaussian white noise processes with covariance matrices V_i, W_i, respectively, and are independent of each other.

The quadratic cost function to be minimized is

$$J = \mathbb{E}\left[\mathbf{x}_N^{\mathrm{T}} F \mathbf{x}_N + \sum_{i=0}^{N-1} (\mathbf{x}_i^{\mathrm{T}} Q_i \mathbf{x}_i + \mathbf{u}_i^{\mathrm{T}} R_i \mathbf{u}_i)\right],$$

$$F \geq 0, Q_i \geq 0, R_i > 0.$$

The discrete-time LQG controller is

$$\hat{\mathbf{x}}_{i+1} = A_i \hat{\mathbf{x}}_i + B_i \mathbf{u}_i + L_{i+1}\left(\mathbf{y}_{i+1} - C_{i+1}\left\{A_i \hat{\mathbf{x}}_i + B_i \mathbf{u}_i\right\}\right), \qquad \hat{\mathbf{x}}_0 = \mathbb{E}[\mathbf{x}_0],$$

$$\mathbf{u}_i = -K_i \hat{\mathbf{x}}_i.$$

and $\hat{\mathbf{x}}_i$ corresponds to the predictive estimate $\hat{\mathbf{x}}_i = \mathbb{E}[\mathbf{x}_i | \mathbf{y}^i, \mathbf{u}^{i-1}]$.

The Kalman gain equals

$$L_i = P_i C_i^{\mathrm{T}} (C_i P_i C_i^{\mathrm{T}} + W_i)^{-1},$$

where P_i is determined by the following matrix Riccati difference equation that runs forward in time:

$$P_{i+1} = A_i \left(P_i - P_i C_i^{\mathrm{T}} \left(C_i P_i C_i^{\mathrm{T}} + W_i\right)^{-1} C_i P_i\right) A_i^{\mathrm{T}} + V_i, \qquad P_0 = \mathbb{E}[(\mathbf{x}_0 - \hat{\mathbf{x}}_0)(\mathbf{x}_0 - \hat{\mathbf{x}}_0)^{\mathrm{T}}].$$

Linear-quadratic Gaussian control (LQG)

The discrete-time linear system equations are

$$\mathbf{x}_{i+1} = A_i \mathbf{x}_i + B_i \mathbf{u}_i + \mathbf{v}_i,$$

$$\mathbf{y}_i = C_i \mathbf{x}_i + \mathbf{w}_i.$$

Here i represents the discrete time index and $\mathbf{v}_i, \mathbf{w}_i$ represent discrete-time Gaussian white noise processes with covariance matrices V_i, W_i, respectively, and are independent of each other.

The quadratic cost function to be minimized is

$$J = \mathbb{E}\left[\mathbf{x}_N^{\mathrm{T}} F \mathbf{x}_N + \sum_{i=0}^{N-1} (\mathbf{x}_i^{\mathrm{T}} Q_i \mathbf{x}_i + \mathbf{u}_i^{\mathrm{T}} R_i \mathbf{u}_i)\right],$$

$$F \geq 0, Q_i \geq 0, R_i > 0.$$

The discrete-time LQG controller is

$$\hat{\mathbf{x}}_{i+1} = A_i \hat{\mathbf{x}}_i + B_i \mathbf{u}_i + L_{i+1}\left(\mathbf{y}_{i+1} - C_{i+1}\left\{A_i \hat{\mathbf{x}}_i + B_i \mathbf{u}_i\right\}\right), \qquad \hat{\mathbf{x}}_0 = \mathbb{E}[\mathbf{x}_0],$$

$$\mathbf{u}_i = -K_i \hat{\mathbf{x}}_i.$$

and $\hat{\mathbf{x}}_i$ corresponds to the predictive estimate $\hat{\mathbf{x}}_i = \mathbb{E}[\mathbf{x}_i | \mathbf{y}^i, \mathbf{u}^{i-1}]$.

The feedback gain matrix equals

$$K_i = (B_i^{\mathrm{T}} S_{i+1} B_i + R_i)^{-1} B_i^{\mathrm{T}} S_{i+1} A_i$$

where S_i is determined by the following matrix Riccati difference equation that runs backward in time:

$$S_i = A_i^{\mathrm{T}}\left(S_{i+1} - S_{i+1} B_i \left(B_i^{\mathrm{T}} S_{i+1} B_i + R_i\right)^{-1} B_i^{\mathrm{T}} S_{i+1}\right) A_i + Q_i, \quad S_N = F.$$

Linear-quadratic Gaussian control (LQG)

The Kalman gain equals

$$L_i = P_i C_i^{\mathrm{T}} (C_i P_i C_i^{\mathrm{T}} + W_i)^{-1},$$

where P_i is determined by the following matrix Riccati difference equation that runs forward in time:

$$P_{i+1} = A_i \left(P_i - P_i C_i^{\mathrm{T}} \left(C_i P_i C_i^{\mathrm{T}} + W_i \right)^{-1} C_i P_i \right) A_i^{\mathrm{T}} + V_i, \qquad P_0 = \mathbb{E}[(\mathbf{x}_0 - \hat{\mathbf{x}}_0)(\mathbf{x}_0 - \hat{\mathbf{x}}_0)^{\mathrm{T}}].$$

The feedback gain matrix equals

$$K_i = (B_i^{\mathrm{T}} S_{i+1} B_i + R_i)^{-1} B_i^{\mathrm{T}} S_{i+1} A_i$$

where S_i is determined by the following matrix Riccati difference equation that runs backward in time:

$$S_i = A_i^{\mathrm{T}} \left(S_{i+1} - S_{i+1} B_i \left(B_i^{\mathrm{T}} S_{i+1} B_i + R_i \right)^{-1} B_i^{\mathrm{T}} S_{i+1} \right) A_i + Q_i, \quad S_N = F.$$

Linear-quadratic Gaussian control (LQG)

The Kalman gain equals

$$L_i = P_i C_i^{\mathrm{T}} (C_i P_i C_i^{\mathrm{T}} + W_i)^{-1},$$

where P_i is determined by the following matrix Riccati difference equation that runs forward in time:

$$P_{i+1} = A_i \left(P_i - P_i C_i^{\mathrm{T}} \left(C_i P_i C_i^{\mathrm{T}} + W_i \right)^{-1} C_i P_i \right) A_i^{\mathrm{T}} + V_i, \qquad P_0 = \mathbb{E}[(\mathbf{x}_0 - \hat{\mathbf{x}}_0)(\mathbf{x}_0 - \hat{\mathbf{x}}_0)^{\mathrm{T}}].$$

The feedback gain matrix equals

$$K_i = (B_i^{\mathrm{T}} S_{i+1} B_i + R_i)^{-1} B_i^{\mathrm{T}} S_{i+1} A_i$$

where S_i is determined by the following matrix Riccati difference equation that runs backward in time:

$$S_i = A_i^{\mathrm{T}} \left(S_{i+1} - S_{i+1} B_i \left(B_i^{\mathrm{T}} S_{i+1} B_i + R_i \right)^{-1} B_i^{\mathrm{T}} S_{i+1} \right) A_i + Q_i, \quad S_N = F.$$

Wikipedia: “Observe the similarity of the two matrix Riccati differential equations,
the first one running forward in time, the second one running backward in time.
This similarity is called **duality**.
The first matrix Riccati differential equation solves the linear–quadratic estimation problem (LQE).
The second matrix Riccati differential equation solves the linear–quadratic regulator problem (LQR).
These problems are dual and together they solve the linear–quadratic–Gaussian control problem (LQG).
So the LQG problem separates into the LQE and LQR problem that can be solved independently.
Therefore, the LQG problem is called **separable**.”

Linear-quadratic Gaussian control (LQG)

The Kalman gain equals

$$L_i = P_i C_i^{\mathrm{T}} (C_i P_i C_i^{\mathrm{T}} + W_i)^{-1},$$

where P_i is determined by the following matrix Riccati difference equation that runs forward in time:

$$P_{i+1} = A_i \left(P_i - P_i C_i^{\mathrm{T}} \left(C_i P_i C_i^{\mathrm{T}} + W_i \right)^{-1} C_i P_i \right) A_i^{\mathrm{T}} + V_i, \qquad P_0 = \mathbb{E}[(\mathbf{x}_0 - \hat{\mathbf{x}}_0)(\mathbf{x}_0 - \hat{\mathbf{x}}_0)^{\mathrm{T}}].$$

The feedback gain matrix equals

$$K_i = (B_i^{\mathrm{T}} S_{i+1} B_i + R_i)^{-1} B_i^{\mathrm{T}} S_{i+1} A_i$$

where S_i is determined by the following matrix Riccati difference equation that runs backward in time:

$$S_i = A_i^{\mathrm{T}} \left(S_{i+1} - S_{i+1} B_i \left(B_i^{\mathrm{T}} S_{i+1} B_i + R_i \right)^{-1} B_i^{\mathrm{T}} S_{i+1} \right) A_i + Q_i, \quad S_N = F.$$

Wikipedia: “Observe the similarity of the two matrix Riccati differential equations,
the first one running forward in time, the second one running backward in time.
This similarity is called **duality**.
The first matrix Riccati differential equation solves the linear–quadratic estimation problem (LQE).
The second matrix Riccati differential equation solves the linear–quadratic regulator problem (LQR).
These problems are dual and together they solve the linear–quadratic–Gaussian control problem (LQG).
So the LQG problem separates into the LQE and LQR problem that can be solved independently.
Therefore, the LQG problem is called **separable**.”

Separation principle

Feedback

Piazza thread: 3/9 Lec 14 Feedback

Please post your answers to the following anonymously.

1. What did you like today?

2. What was unclear?

3. Any fun spring break plans?

4. Any additional feedback / comments?