

SES/RAS 598: Space Robotics and AI

Lecture 1: Course Introduction & State Estimation Overview

Dr. Jnaneshwar Das

Arizona State University
School of Earth and Space Exploration

Spring 2025

Lecture Outline

- 1 Course Overview
- 2 State Estimation Fundamentals
- 3 Linear Dynamical Systems
- 4 Next Steps

Course Structure

- **Meeting Times:** Tu/Th 10:30-11:45am

Course Structure

- **Meeting Times:** Tu/Th 10:30-11:45am
- **Location:** PSF 647

Course Structure

- **Meeting Times:** Tu/Th 10:30-11:45am
- **Location:** PSF 647
- **Course Components:**
 - Assignments (20%)
 - Midterm Project (20%)
 - Final Project (50%)
 - Class Participation (10%)

Course Structure

- **Meeting Times:** Tu/Th 10:30-11:45am
- **Location:** PSF 647
- **Course Components:**
 - Assignments (20%)
 - Midterm Project (20%)
 - Final Project (50%)
 - Class Participation (10%)
- **Prerequisites:**
 - Linear algebra, calculus, probability theory
 - Python programming with NumPy, SciPy
 - Basic computer vision concepts
 - Linux/Unix systems experience

- **Recommended Books:**

- Probabilistic Robotics (Thrun, Burgard, Fox)
- Optimal State Estimation (Simon)
- Pattern Recognition and Machine Learning (Bishop)

- **Recommended Books:**

- Probabilistic Robotics (Thrun, Burgard, Fox)
- Optimal State Estimation (Simon)
- Pattern Recognition and Machine Learning (Bishop)

- **Interactive Tutorials:**

- Sensor Fusion
- Parameter Estimation
- Gaussian Processes

- **Recommended Books:**

- Probabilistic Robotics (Thrun, Burgard, Fox)
- Optimal State Estimation (Simon)
- Pattern Recognition and Machine Learning (Bishop)

- **Interactive Tutorials:**

- Sensor Fusion
- Parameter Estimation
- Gaussian Processes

- **Required Software:**

- Linux OS
- ROS2
- Python with scientific computing libraries

Why State Estimation?

- **Real-World Applications:**

- Mars rover navigation
- Drone flight control
- Satellite attitude determination

Why State Estimation?

- **Real-World Applications:**

- Mars rover navigation
- Drone flight control
- Satellite attitude determination

- **Key Challenges:**

- Sensor noise and uncertainty
- Environmental dynamics
- Resource constraints

Why State Estimation?

- **Real-World Applications:**

- Mars rover navigation
- Drone flight control
- Satellite attitude determination

- **Key Challenges:**

- Sensor noise and uncertainty
- Environmental dynamics
- Resource constraints

- **Impact on Space Exploration:**

- Autonomous navigation
- Precision landing
- Sample collection

- **Mathematical Foundation:**

$$\hat{\theta} = \arg \min_{\theta} \sum_{i=1}^n (y_i - h(\theta))^2$$

- **Mathematical Foundation:**

$$\hat{\theta} = \arg \min_{\theta} \sum_{i=1}^n (y_i - h(\theta))^2$$

- **Key Properties:**

- Minimizes squared error
- Optimal for Gaussian noise
- Computationally efficient

- **Mathematical Foundation:**

$$\hat{\theta} = \arg \min_{\theta} \sum_{i=1}^n (y_i - h(\theta))^2$$

- **Key Properties:**

- Minimizes squared error
- Optimal for Gaussian noise
- Computationally efficient

- **Applications:**

- Sensor calibration
- Trajectory estimation
- Parameter identification

Implementation Example: Least Squares Estimation

```
1 import numpy as np
2 from scipy.optimize import minimize
3
4 class LeastSquaresEstimator:
5     def __init__(self, measurements, measurement_model):
6         self.y = measurements          # Measurement vector
7         self.h = measurement_model     # Measurement model function
8
9     def cost_function(self, theta):
10         """Compute sum of squared errors."""
11         residuals = self.y - self.h(theta)
12         return np.sum(residuals**2)
13
14     def estimate(self, theta_init):
15         """Find parameters that minimize squared error."""
16         result = minimize(self.cost_function, theta_init,
17                           method='Nelder-Mead')
18         return result.x # Return optimal parameters
```


Maximum Likelihood Estimation

- **Principle:**

$$\hat{\theta}_{\text{MLE}} = \arg \max_{\theta} \prod_{i=1}^n p(y_i | \theta)$$

Maximum Likelihood Estimation

- **Principle:**

$$\hat{\theta}_{\text{MLE}} = \arg \max_{\theta} \prod_{i=1}^n p(y_i | \theta)$$

- **Connection to Least Squares:**

- Equivalent under Gaussian assumptions
- More general framework
- Handles different noise models

Maximum Likelihood Estimation

- **Principle:**

$$\hat{\theta}_{\text{MLE}} = \arg \max_{\theta} \prod_{i=1}^n p(y_i | \theta)$$

- **Connection to Least Squares:**

- Equivalent under Gaussian assumptions
- More general framework
- Handles different noise models

- **Space Applications:**

- Orbit determination
- Attitude estimation
- Sensor fusion

Implementation Example: Maximum Likelihood Estimation

```
1 import numpy as np
2 from scipy.stats import norm
3 from scipy.optimize import minimize
4
5 class MLEstimator:
6     def __init__(self, measurements, measurement_model):
7         self.y = measurements      # Measurement vector
8         self.h = measurement_model # Measurement model function
9
10    def neg_log_likelihood(self, theta):
11        """Compute negative log-likelihood."""
12        residuals = self.y - self.h(theta) # Assuming Gaussian noise model
13        return -np.sum(norm.logpdf(residuals))
14
15    def estimate(self, theta_init):
16        """Find parameters that maximize likelihood."""
17        result = minimize(self.neg_log_likelihood, theta_init,
18                          method='Nelder-Mead')
19        return result.x # Return optimal parameters
```

- **System Dynamics:**

$$\begin{aligned}x_{k+1} &= Ax_k + Bu_k + w_k \\ y_k &= Cx_k + v_k\end{aligned}$$

- **System Dynamics:**

$$\begin{aligned}x_{k+1} &= Ax_k + Bu_k + w_k \\ y_k &= Cx_k + v_k\end{aligned}$$

- **Components:**

- State vector x_k
- Input vector u_k
- Measurement vector y_k
- Process noise w_k
- Measurement noise v_k

Case Study: Mars Rover Navigation

- **State Variables:**

- Position (x, y, z)
- Orientation (roll, pitch, yaw)
- Velocities

Case Study: Mars Rover Navigation

- **State Variables:**

- Position (x, y, z)
- Orientation (roll, pitch, yaw)
- Velocities

- **Sensors:**

- Visual odometry
- Inertial measurement unit (IMU)
- Sun sensors

Case Study: Mars Rover Navigation

- **State Variables:**

- Position (x, y, z)
- Orientation (roll, pitch, yaw)
- Velocities

- **Sensors:**

- Visual odometry
- Inertial measurement unit (IMU)
- Sun sensors

- **Challenges:**

- Wheel slippage
- Varying terrain
- Limited computational resources

Implementation Example: State-Space Model

```
1 import numpy as np
2 from scipy.stats import multivariate_normal
3
4 class LinearStateSpaceModel:
5     def __init__(self, A, B, C, Q, R):
6         self.A = A # State transition matrix
7         self.B = B # Input matrix
8         self.C = C # Measurement matrix
9         self.Q = Q # Process noise covariance
10        self.R = R # Measurement noise covariance
11
12    def propagate_state(self, x, u=None):
13        """Propagate state forward one step."""
14        w = multivariate_normal.rvs(mean=np.zeros(x.shape), cov=self.Q)
15        if u is not None:
16            return self.A @ x + self.B @ u + w
17        return self.A @ x + w
18
19    def get_measurement(self, x):
20        """Get noisy measurement of current state."""
21        v = multivariate_normal.rvs(mean=np.zeros(self.C.shape[0]), cov=self.R)
22        return self.C @ x + v
```

Preparation for Next Lecture

- **Review:**

- Matrix operations
- Probability concepts
- Basic Python programming

Preparation for Next Lecture

- **Review:**

- Matrix operations
- Probability concepts
- Basic Python programming

- **Setup:**

- Install Linux if needed
- Configure ROS2 environment
- Test Python scientific libraries

Preparation for Next Lecture

- **Review:**

- Matrix operations
- Probability concepts
- Basic Python programming

- **Setup:**

- Install Linux if needed
- Configure ROS2 environment
- Test Python scientific libraries

- **Reading:**

- Skim Kalman filter basics
- Review assigned papers
- Explore interactive tutorials

Thank you!

Contact: jdas5@asu.edu

Course Assessment: Quiz Components

- **Part 1: Robotics and AI Fundamentals**

- Tests foundational knowledge in robotics and AI
- Key topics: SLAM, LiDAR, occupancy grids, GPS challenges, path planning
- Required score: 60% to proceed to Part 2

- **Part 2: Advanced Concepts**

- Tests understanding of tutorial concepts
- Topics covered:
 - Multi-view geometry and stereo vision
 - SLAM systems and loop closure
 - Kalman filter parameters and tuning
 - Sampling strategies for exploration
 - Error analysis in rock mapping

- **Foundational Topics (Part 1)**

- SLAM and LiDAR: Core to robot perception and mapping
- Occupancy grids: Probabilistic environment representation
- GPS challenges: Motivation for robust state estimation
- Path planning: Essential for autonomous navigation

- **Advanced Applications (Part 2)**

- Stereo vision: Depth estimation for rock mapping
- Loop closure: Improving map consistency
- Kalman filters: State estimation for robot localization
- Sampling strategies: Efficient exploration algorithms
- Error analysis: Ensuring reliable measurements