SES/RAS 598: Space Robotics and AI

Lecture 2: State Estimation Techniques

Dr. Jnaneshwar Das

Arizona State University
School of Earth and Space Exploration

Spring 2025

Lecture Outline

Introduction to Kalman Filters

- **Key Concepts:**
  - Recursive state estimation
  - Optimal for linear systems
  - Handles Gaussian uncertainty

Introduction to Kalman Filters

- **Key Concepts:**
  - Recursive state estimation
  - Optimal for linear systems
  - Handles Gaussian uncertainty
- **Applications in Navigation:**
  - Spacecraft position tracking
  - Drone state estimation
  - Robot localization

Introduction to Kalman Filters

- **Key Concepts:**
  - Recursive state estimation
  - Optimal for linear systems
  - Handles Gaussian uncertainty
- **Applications in Navigation:**
  - Spacecraft position tracking
  - Drone state estimation
  - Robot localization
- **Advantages:**
  - Computationally efficient
  - Handles noisy measurements
  - Provides uncertainty estimates

Kalman Filter Algorithm

- **Prediction Step:**

$$\hat{x}_{k|k-1} = F_k \hat{x}_{k-1|k-1} + B_k u_k$$
$$P_{k|k-1} = F_k P_{k-1|k-1} F_k^T + Q_k$$

Kalman Filter Algorithm

- **Prediction Step:**

$$\hat{x}_{k|k-1} = F_k \hat{x}_{k-1|k-1} + B_k u_k$$
$$P_{k|k-1} = F_k P_{k-1|k-1} F_k^T + Q_k$$

- **Update Step:**

$$K_k = P_{k|k-1} H_k^T (H_k P_{k|k-1} H_k^T + R_k)^{-1}$$
$$\hat{x}_{k|k} = \hat{x}_{k|k-1} + K_k (z_k - H_k \hat{x}_{k|k-1})$$
$$P_{k|k} = (I - K_k H_k) P_{k|k-1}$$

Implementation in Python

def predict(self, x, P, u): $x_p red = self.F@x + self.B@u P_p red = self.F@P@self.F.T + self.Q return x$

def update(self, $x_p red, P_p red, z) : K = P_p red@self.H.T@np.linalg.inv(self.H@P_p red@self.H.T +$
$self.R) x = x_p red + K@(z - self.H@x_p red) P = (np.eye(len(x)) - K@self.H)@P_p red return x, P$

Introduction to Particle Filters

- **Key Features:**
  - Non-parametric estimation
  - Handles non-linear systems
  - Represents arbitrary distributions

Introduction to Particle Filters

- **Key Features:**
  - Non-parametric estimation
  - Handles non-linear systems
  - Represents arbitrary distributions
- **Working Principle:**
  - Particle representation
  - Sequential importance sampling
  - Resampling step

Introduction to Particle Filters

- **Key Features:**
  - Non-parametric estimation
  - Handles non-linear systems
  - Represents arbitrary distributions
- **Working Principle:**
  - Particle representation
  - Sequential importance sampling
  - Resampling step
- **Advantages:**
  - No linearity assumption
  - Handles multi-modal distributions
  - Robust to outliers

Particle Filter Algorithm

for each time step k **do**
  for each particle i **do**
    Sample new state: $x_k^i \sim p(x_k | x_{k-1}^i, u_k)$
    Update weight: $w_k^i = w_{k-1}^i p(z_k | x_k^i)$
  end for
  Normalize weights: $w_k^i = w_k^i / \sum_j w_k^j$
  if $N_{eff} < N_{threshold}$ **then**
    Resample particles
  end if
end for

Implementation Example

def predict(self, u): Propagate particles through motion model for i in range(len(self.particles)): self.particles[i] = motion$_m odel(self.particles[i], u)$
def update(self, z): Update weights based on measurement for i in range(len(self.particles)): self.weights[i] *= measurement$_m odel(self.particles[i], z) self.weights/ = np.sum(self.weights)$
Resample if needed if self.neff() ¡ self.n$_t hreshold : self.resample()$

Assignment Overview

- **Objectives:**
  - Implement 2D state estimation using ROS2
  - Compare Kalman and particle filter performance
  - Visualize results using RViz

Assignment Overview

- **Objectives:**
  - Implement 2D state estimation using ROS2
  - Compare Kalman and particle filter performance
  - Visualize results using RViz
- **Key Components:**
  - ROS2 node implementation
  - Filter implementation
  - Visualization and analysis

Assignment Overview

- **Objectives:**
  - Implement 2D state estimation using ROS2
  - Compare Kalman and particle filter performance
  - Visualize results using RViz
- **Key Components:**
  - ROS2 node implementation
  - Filter implementation
  - Visualization and analysis
- **Evaluation:**
  - Code quality and documentation
  - Filter performance metrics
  - Analysis and discussion

ROS2 Environment Setup

Create package ros2 pkg create –build-type ament$_{p}ython\ state_{e}stimation_{a}ssignment$
Build workspace cd /ros2$_{w}scolconbuild$
Source workspace source install/setup.bash
~

Basic ROS2 Node Structure

class StateEstimator(Node): def $_init_{(self):super().\_init_{(}}$

$'$state$_e stimator')self.subscription = self.create_s ubscription(Odometry, 'odom', self.odom_c allb$
$self.create_p ublisher(PoseStamped, 'estimated_p ose', 10)$
def odom$_c allback(self, msg): Implementstateestimationherepass$

Preparation for Next Week

- **Assignment 1:**
  - Review ROS2 basics
  - Study filter implementations
  - Start coding basic structure

Preparation for Next Week

- **Assignment 1:**
  - Review ROS2 basics
  - Study filter implementations
  - Start coding basic structure
- **Reading:**
  - Extended Kalman Filter
  - Unscented Kalman Filter
  - Advanced particle filter topics

Preparation for Next Week

- **Assignment 1:**
  - Review ROS2 basics
  - Study filter implementations
  - Start coding basic structure
- **Reading:**
  - Extended Kalman Filter
  - Unscented Kalman Filter
  - Advanced particle filter topics
- **Tools:**
  - Test ROS2 installation
  - Practice with RViz
  - Explore sensor fusion tutorial

Questions?

Thank you!

Contact: jdas5@asu.edu