#include <iostream>

#include <vector>

#include <cmath>

// Define the measurement and track struct

struct Measurement {

double x;

double y;

};

struct Track {

double x;

double y;

};

// Define a function to calculate the probability density function (PDF) of a Gaussian

double gaussianPDF(double x, double mean, double variance) {

return (1.0 / sqrt(2 \* M\_PI \* variance)) \* exp(-0.5 \* pow((x - mean) / sqrt(variance), 2));

}

// Define the main JPDA function

std::vector<double> jointProbabilisticDataAssociation(const std::vector<Measurement>& measurements, const std::vector<Track>& tracks, double measurementNoise, double gateThreshold) {

std::vector<double> probabilities;

for (const auto& measurement : measurements) {

double totalProbability = 0.0;

for (const auto& track : tracks) {

double dx = measurement.x - track.x;

double dy = measurement.y - track.y;

double distance = sqrt(dx \* dx + dy \* dy);

double measurementProbability = gaussianPDF(distance, 0.0, measurementNoise);

if (measurementProbability > gateThreshold) {

totalProbability += measurementProbability;

}

}

probabilities.push\_back(totalProbability);

}

return probabilities;

}

int main() {

// Example usage

std::vector<Measurement> measurements = {{1.0, 2.0}, {3.0, 4.0}};

std::vector<Track> tracks = {{1.5, 2.5}, {2.5, 3.5}, {3.5, 4.5}};

double measurementNoise = 0.1;

double gateThreshold = 0.01;

std::vector<double> probabilities = jointProbabilisticDataAssociation(measurements, tracks, measurementNoise, gateThreshold);

// Output probabilities

std::cout << "Probabilities:\n";

for (size\_t i = 0; i < probabilities.size(); ++i) {

std::cout << "Measurement " << i+1 << ": " << probabilities[i] << std::endl;

}

return 0;

}

#include <iostream>

#include <vector>

#include <cmath>

// Define the measurement and track struct

struct Measurement {

double x;

double y;

};

struct Track {

double x;

double y;

};

// Define a function to calculate the probability density function (PDF) of a Gaussian

double gaussianPDF(double x, double mean, double variance) {

return (1.0 / sqrt(2 \* M\_PI \* variance)) \* exp(-0.5 \* pow((x - mean) / sqrt(variance), 2));

}

// Define the main JPDA function

std::vector<double> jointProbabilisticDataAssociation(const std::vector<Measurement>& measurements, const std::vector<Track>& tracks, double measurementNoise, double gateThreshold) {

std::vector<double> probabilities;

for (const auto& measurement : measurements) { // iterate over each measurement

double totalProbability = 0.0;

for (const auto& track : tracks) { // iterate over each track

double dx = measurement.x - track.x; // calculate the distance in x direction

double dy = measurement.y - track.y; // calculate the distance in y direction

double distance = sqrt(dx \* dx + dy \* dy); // calculate the distance between measurement and track

double measurementProbability = gaussianPDF(distance, 0.0, measurementNoise); // calculate the probability of the measurement given the track

if (measurementProbability > gateThreshold) { // check if the probability is above the gate threshold

totalProbability += measurementProbability; // add the probability to the total if it is within the gate

}

}

probabilities.push\_back(totalProbability); // store the total probability for this measurement

}

return probabilities; // return the vector of total probabilities for all measurements

}

int main() {

// Example usage

std::vector<Measurement> measurements = {{1.0, 2.0}, {3.0, 4.0}}; // example measurements

std::vector<Track> tracks = {{1.5, 2.5}, {2.5, 3.5}, {3.5, 4.5}}; // example tracks

double measurementNoise = 0.1; // example measurement noise

double gateThreshold = 0.01; // example gate threshold

std::vector<double> probabilities = jointProbabilisticDataAssociation(measurements, tracks, measurementNoise, gateThreshold); // calculate probabilities

// Output probabilities

std::cout << "Probabilities:\n";

for (size\_t i = 0; i < probabilities.size(); ++i) {

std::cout << "Measurement " << i+1 << ": " << probabilities[i] << std::endl; // output the total probabilities for each measurement

}

return 0;

}

Probabilistic Data Association (PDA)

#include <iostream>

#include <vector>

#include <cmath>

// Define the measurement and track struct

struct Measurement {

double x;

double y;

};

struct Track {

double x;

double y;

};

// Define a function to calculate the probability density function (PDF) of a Gaussian

double gaussianPDF(double x, double mean, double variance) {

return (1.0 / sqrt(2 \* M\_PI \* variance)) \* exp(-0.5 \* pow((x - mean) / sqrt(variance), 2));

}

// Define the main PDA function

std::vector<double> probabilisticDataAssociation(const std::vector<Measurement>& measurements, const std::vector<Track>& tracks, double measurementNoise) {

std::vector<double> probabilities;

for (const auto& measurement : measurements) { // iterate over each measurement

double totalProbability = 0.0;

for (const auto& track : tracks) { // iterate over each track

double dx = measurement.x - track.x; // calculate the distance in x direction

double dy = measurement.y - track.y; // calculate the distance in y direction

double distance = sqrt(dx \* dx + dy \* dy); // calculate the distance between measurement and track

double measurementProbability = gaussianPDF(distance, 0.0, measurementNoise); // calculate the probability of the measurement given the track

totalProbability += measurementProbability; // add the probability to the total

}

probabilities.push\_back(totalProbability); // store the total probability for this measurement

}

return probabilities; // return the vector of total probabilities for all measurements

}

int main() {

// Example usage

std::vector<Measurement> measurements = {{1.0, 2.0}, {3.0, 4.0}}; // example measurements

std::vector<Track> tracks = {{1.5, 2.5}, {2.5, 3.5}, {3.5, 4.5}}; // example tracks

double measurementNoise = 0.1; // example measurement noise

std::vector<double> probabilities = probabilisticDataAssociation(measurements, tracks, measurementNoise); // calculate probabilities

// Output probabilities

std::cout << "Probabilities:\n";

for (size\_t i = 0; i < probabilities.size(); ++i) {

std::cout << "Measurement " << i+1 << ": " << probabilities[i] << std::endl; // output the total probabilities for each measurement

}

return 0;

}

```