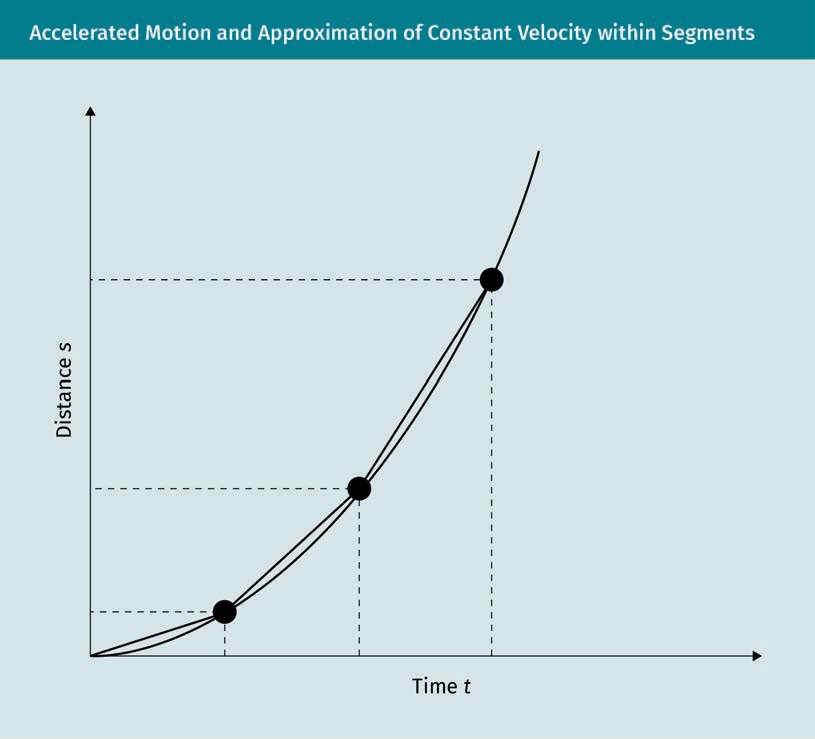
**Segments to Calculate Velocity**

**[Title]**



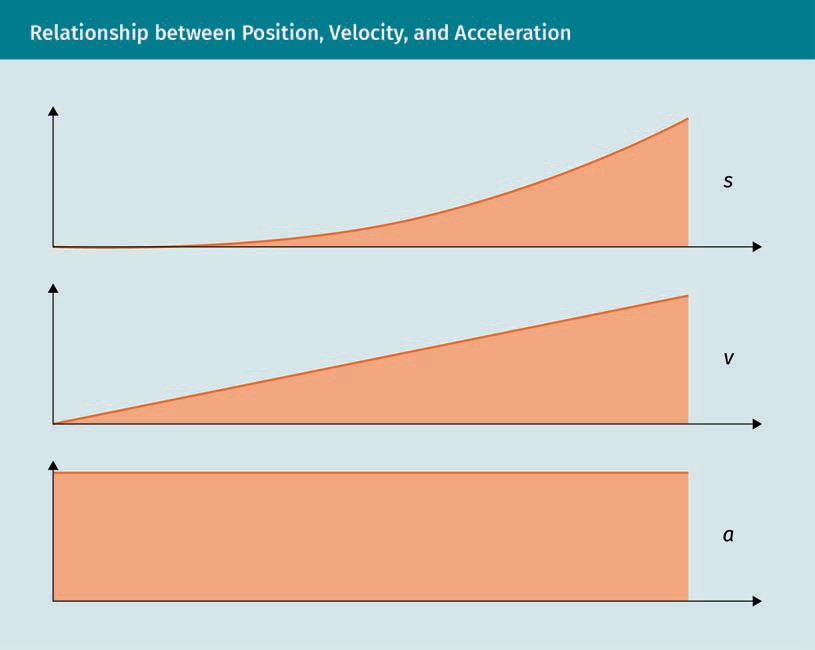
**Accelerated Motion and Approximation of Constant Velocity within Segements**



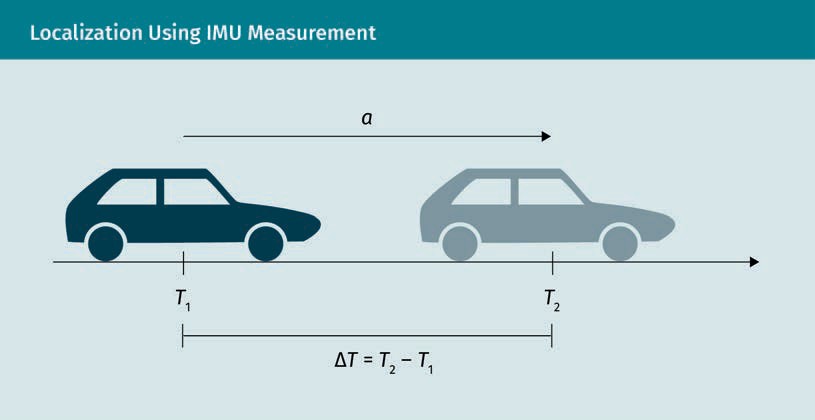
Time

Distance

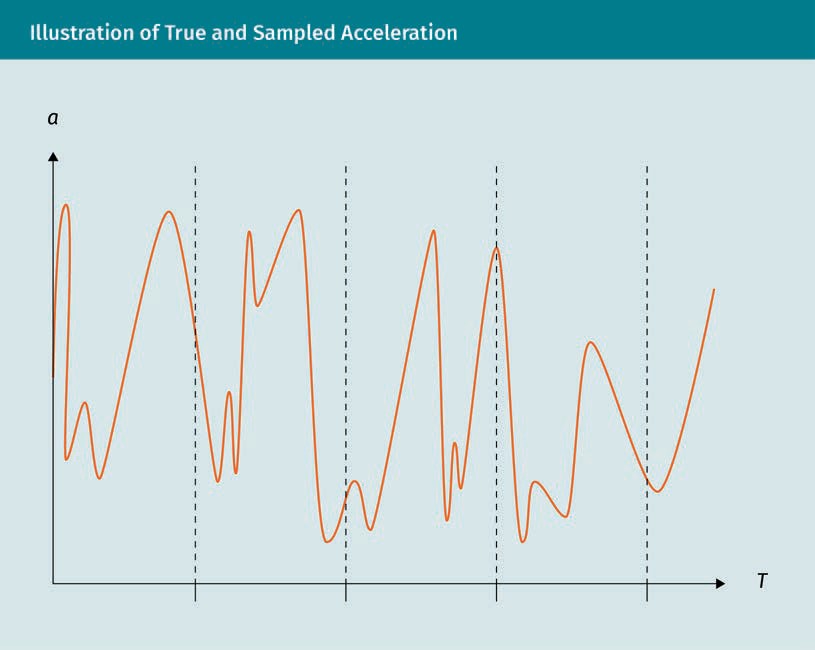
**Relationship between Position, Velocity, and Acceleration**



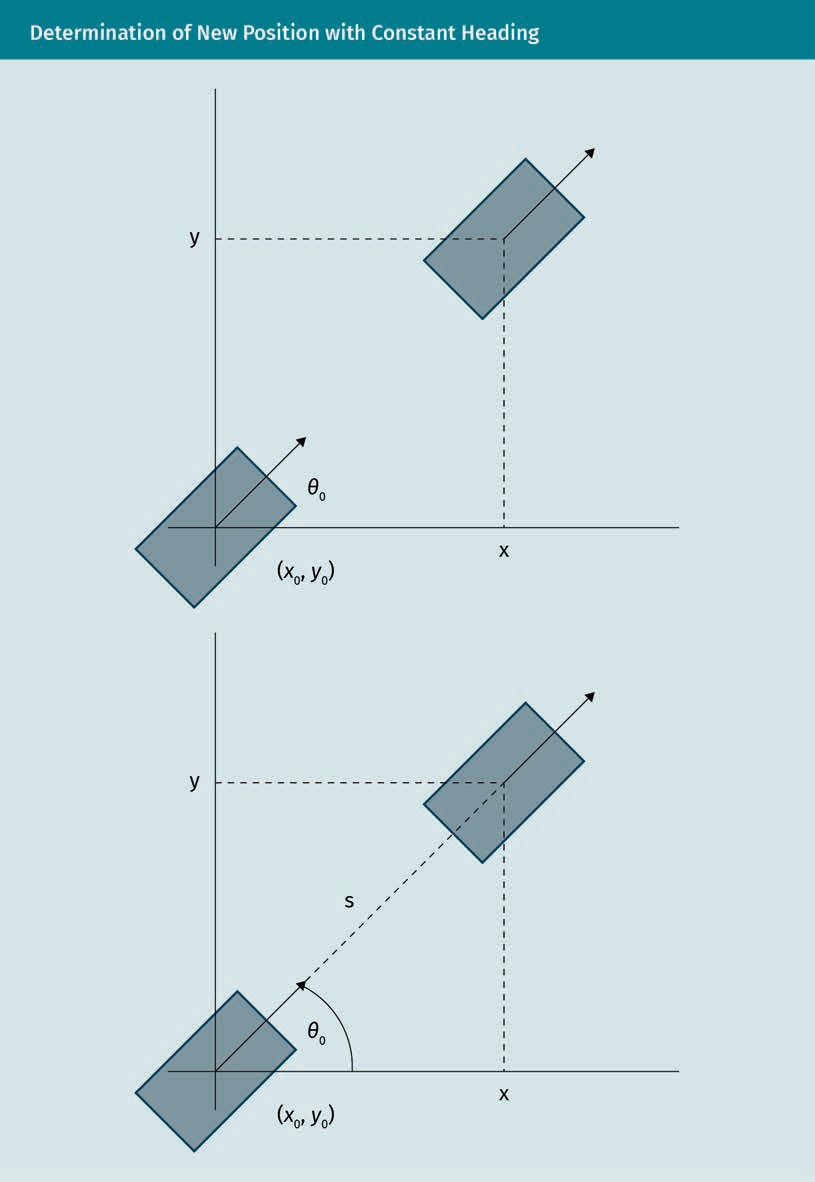
**Localization Using IMU Measurement**



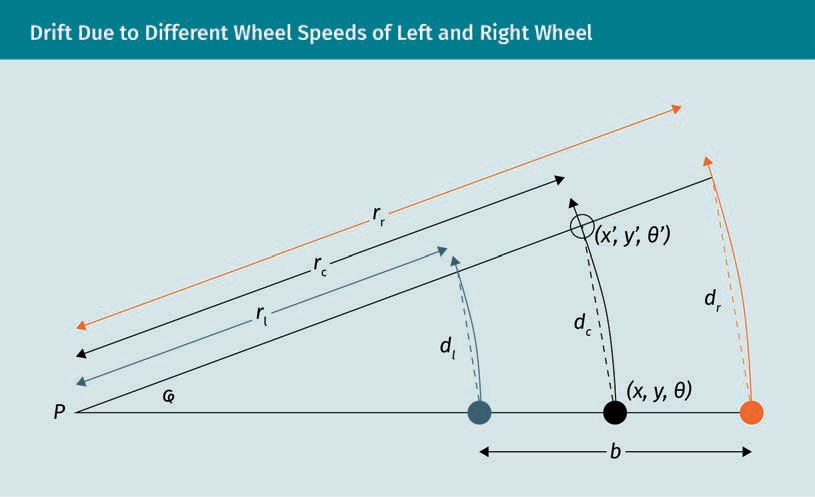
**Illustration of True and Sampled Acceleration**

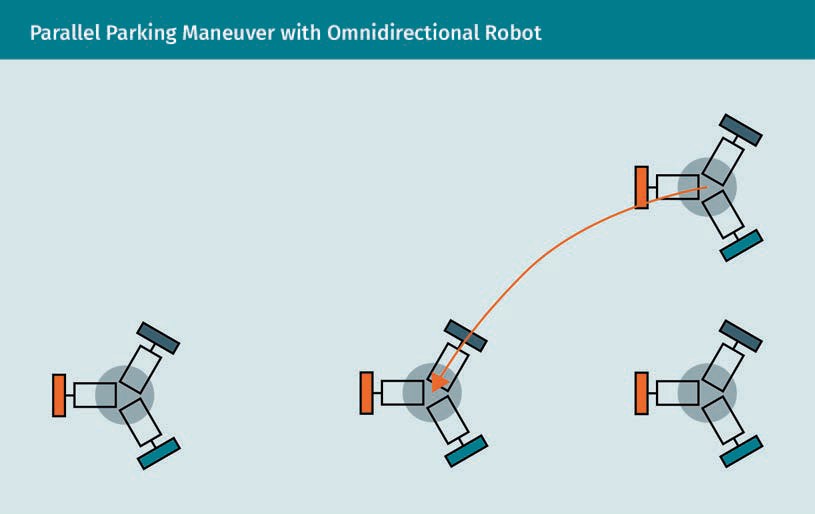


**Determination of New Position with Constant Heading**

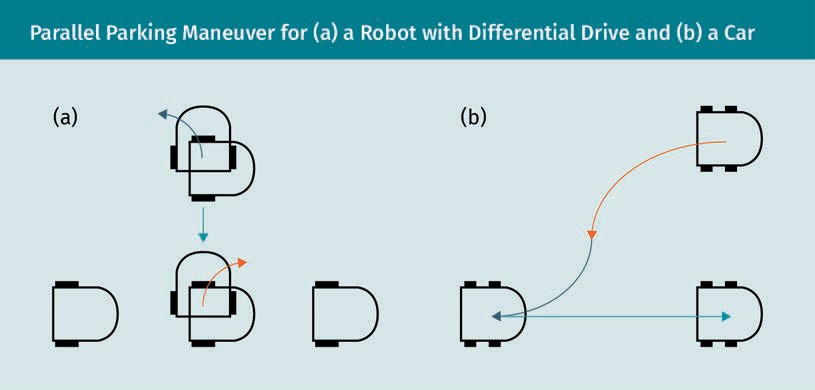


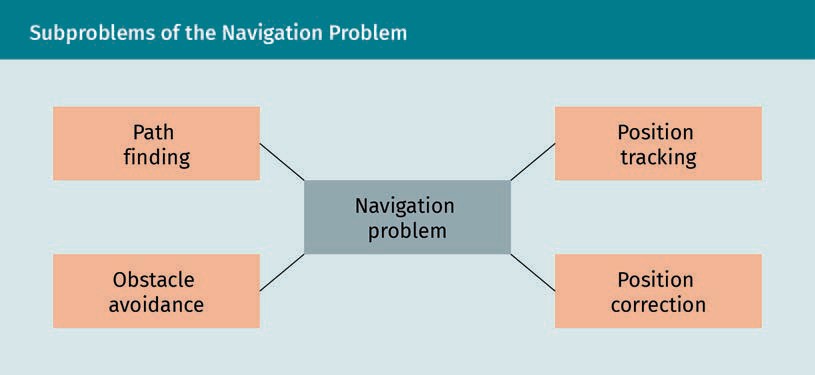
**Drift Due to Different Wheel Speeds of Left and Right Wheel**



**Parallel Parking Maneuver with Omnidirectional Robot**

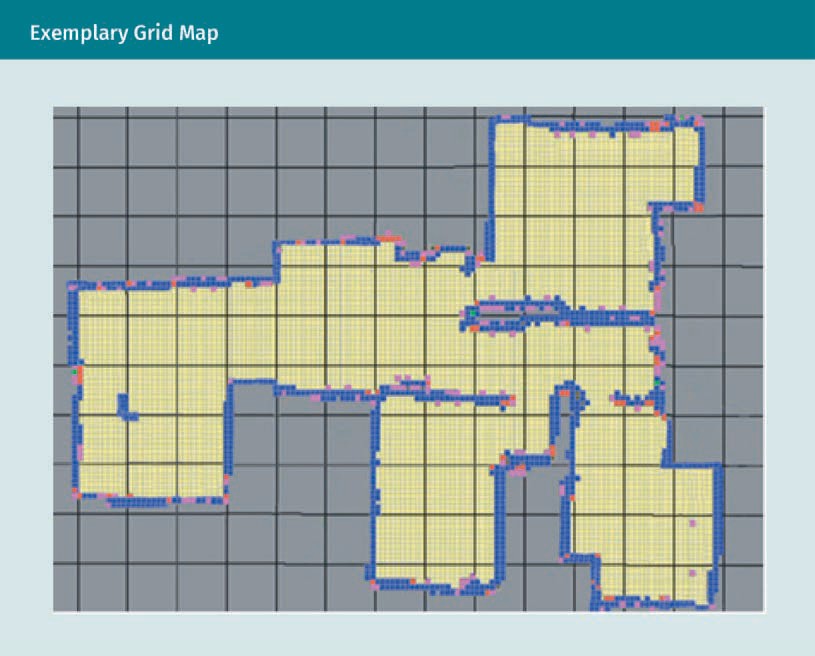
**Parallel Parking Maneuver for (a) a Robot with Differential Drive and (b) a Car**

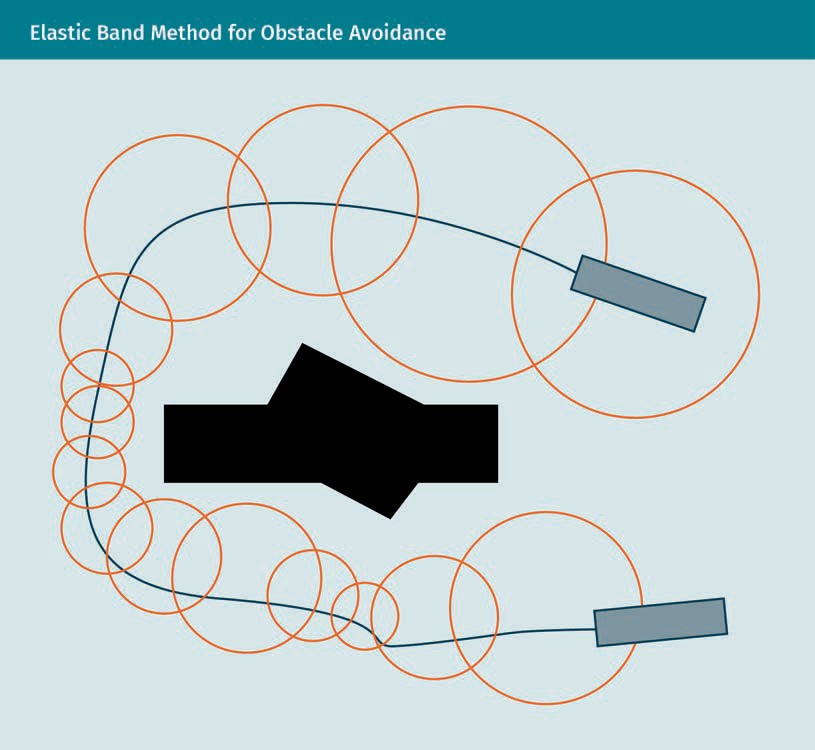


**Subproblems of the Navigation Problem**

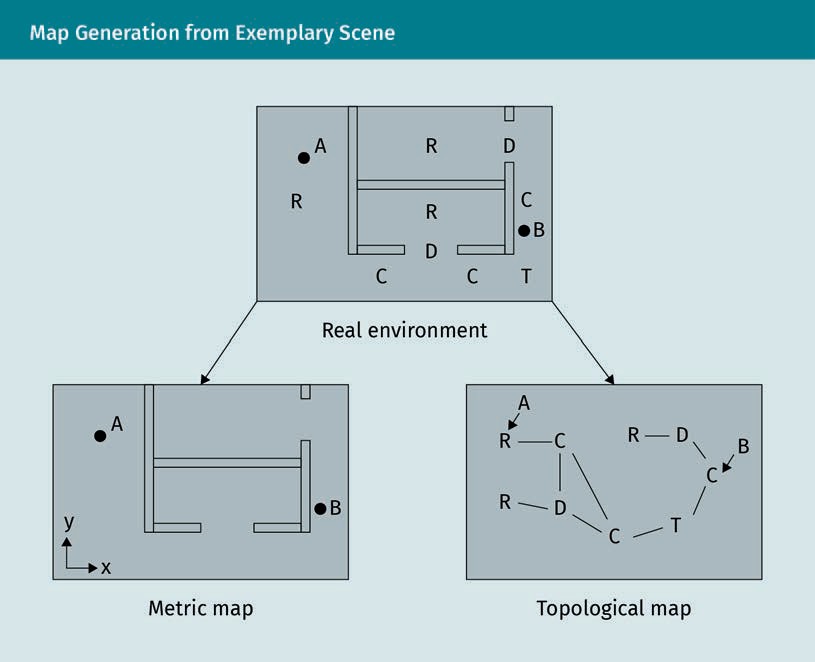
|  |  |
| --- | --- |
| Path finding |  |
| Obstacle avoidance |  |
| Navigation problem |  |
| Position tracking |  |
| Position correction |  |

**Exemplary Grid Map**



**Elastic Band Method for Obstacle Avoidance**

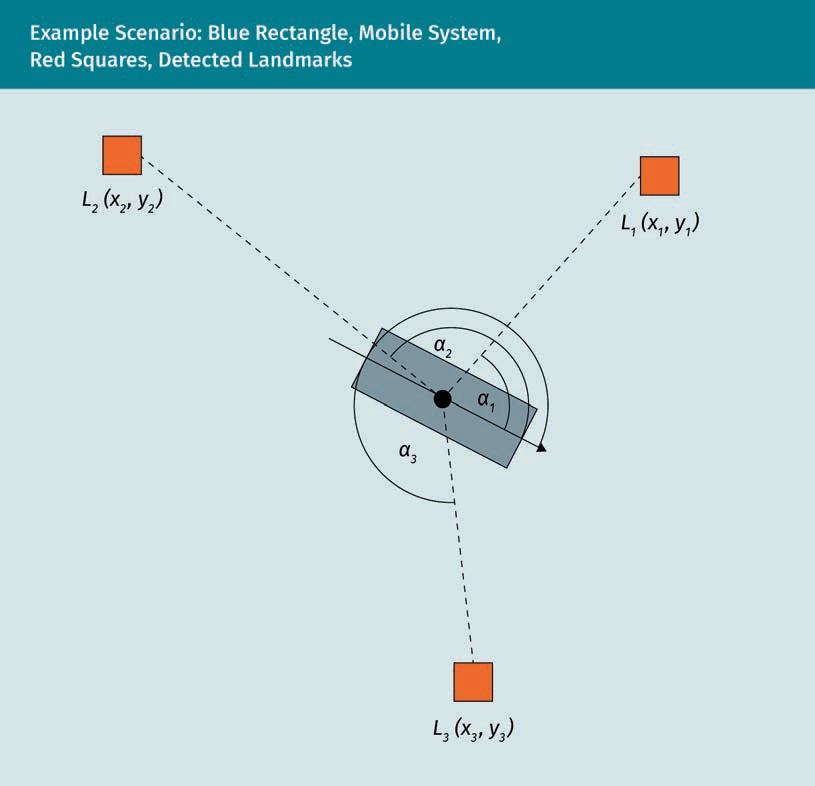
**Map Generation from Exemplary Scene**

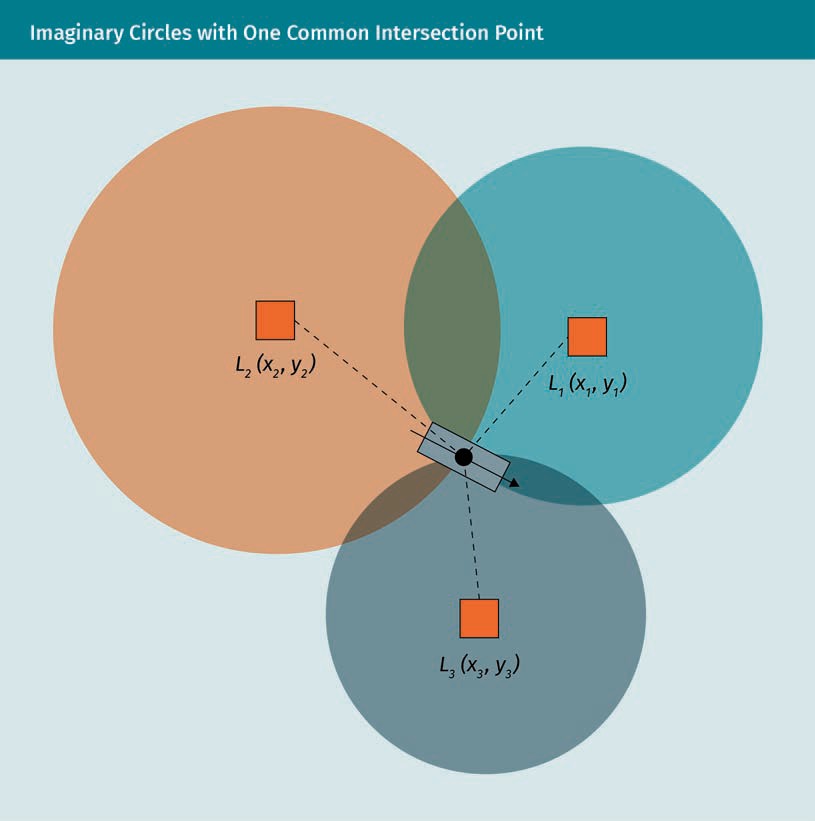


Real environment

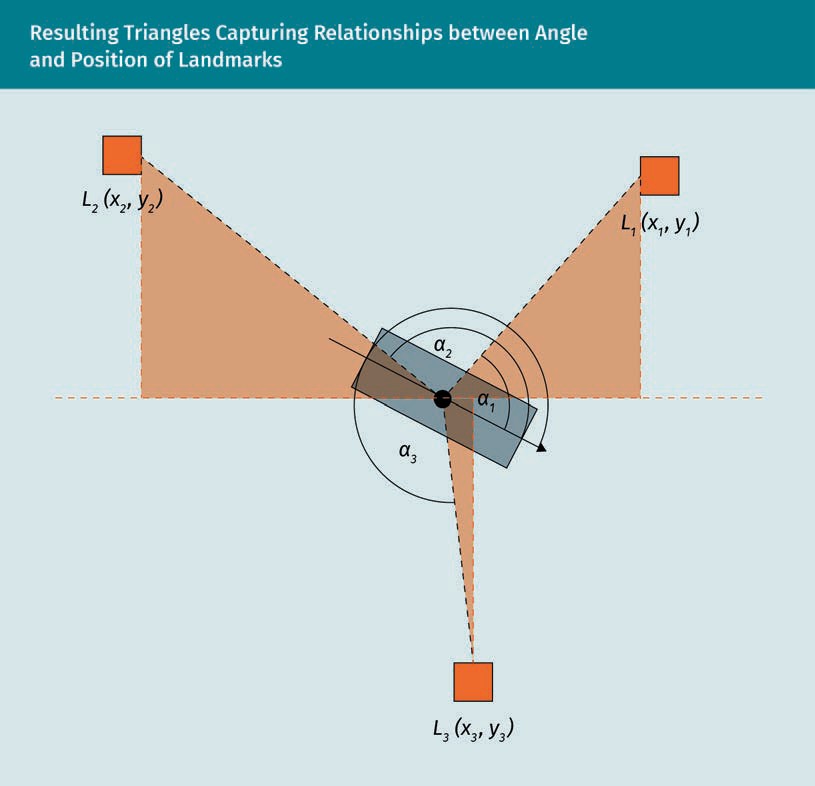
Metric map

Topological map

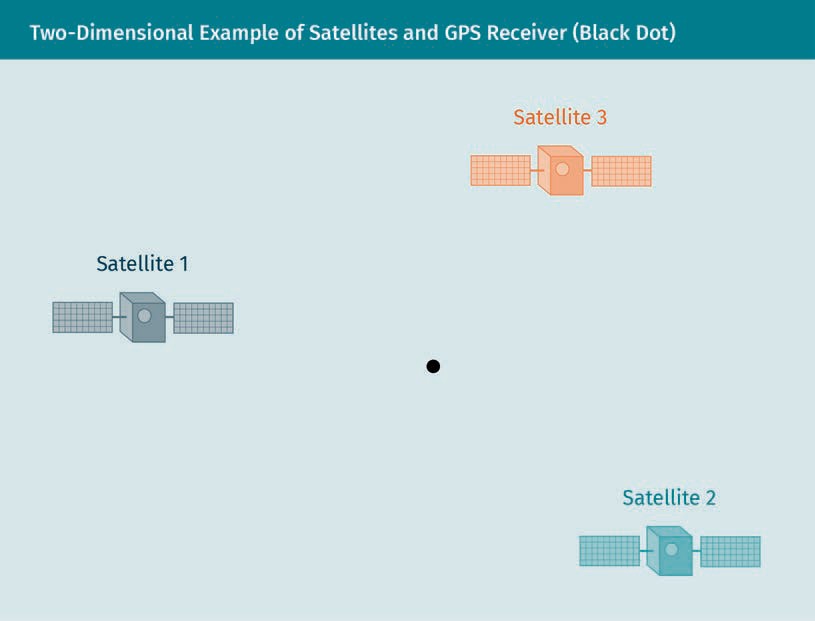
**Example scenario: Blue Rectangle, Mobile System, Red Squares, Detected Landmarks**

**Imaginary Circles with One common Intersection Point**

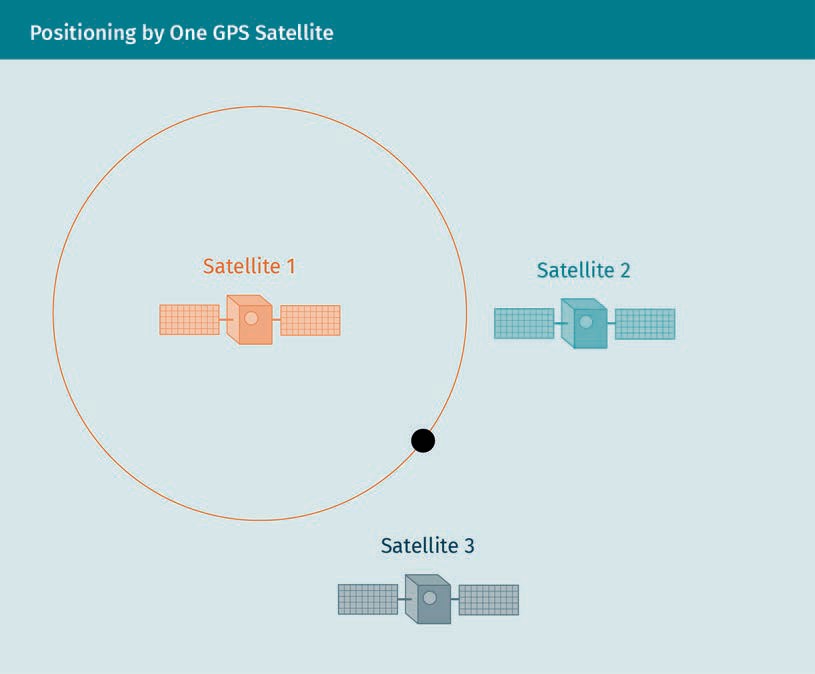
**Resulting Triangles Capturing Relationships between Angle and Position of Landmarks**



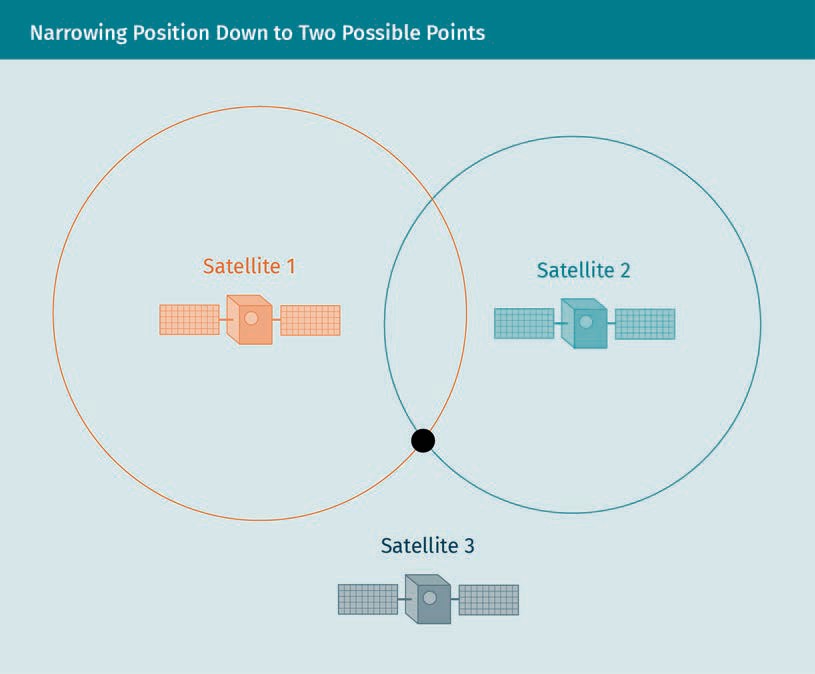
**Two-Dimentional Example of Satellites and GPS Receiver (Black Dot)**

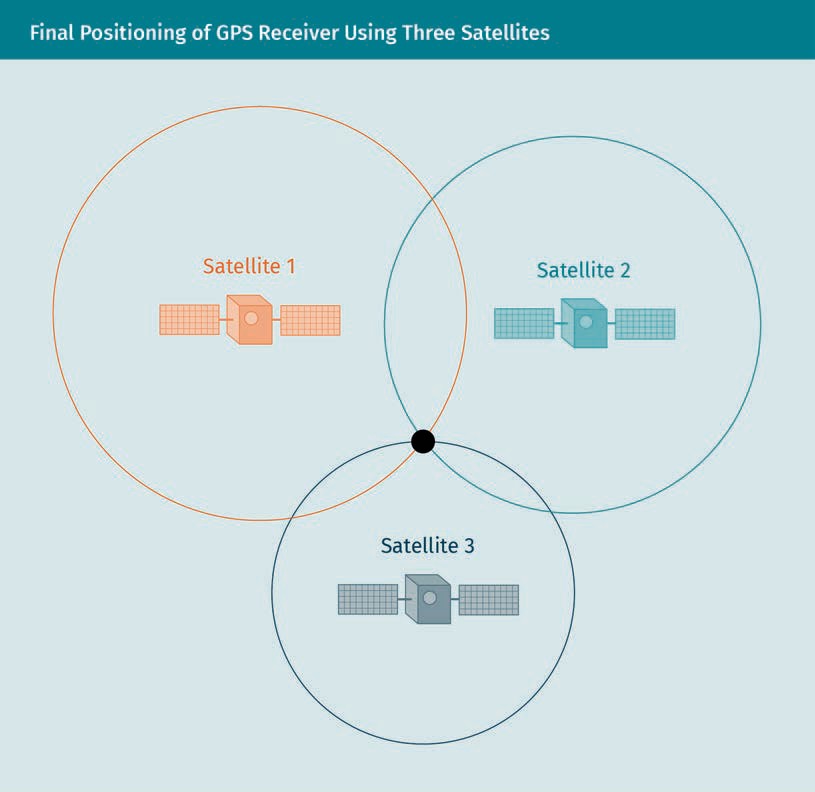


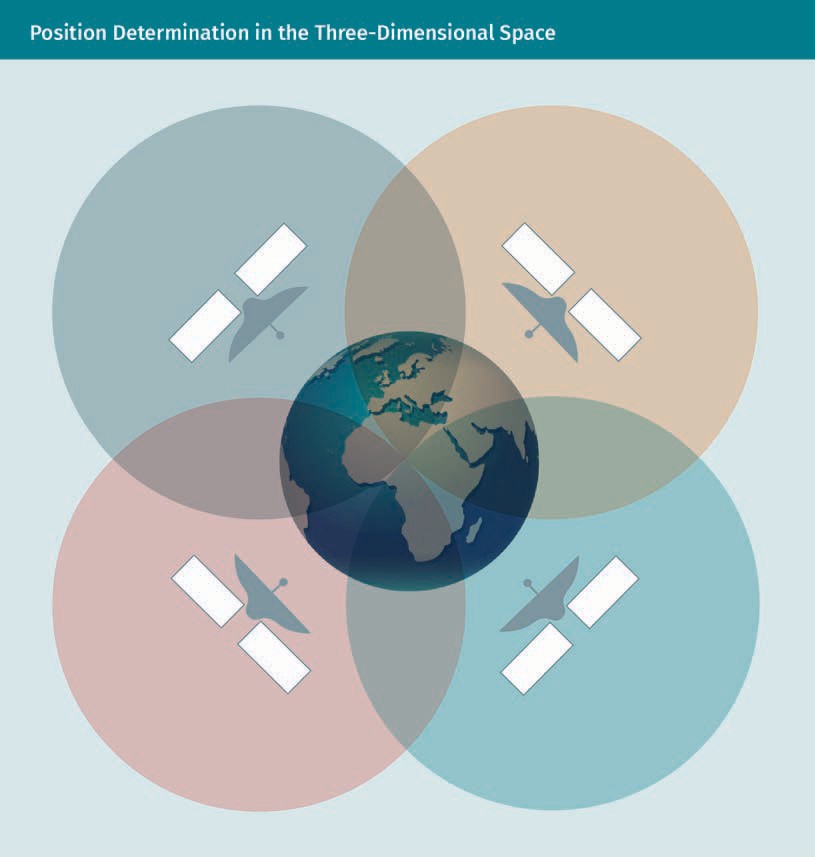
|  |  |
| --- | --- |
| Satellite |  |

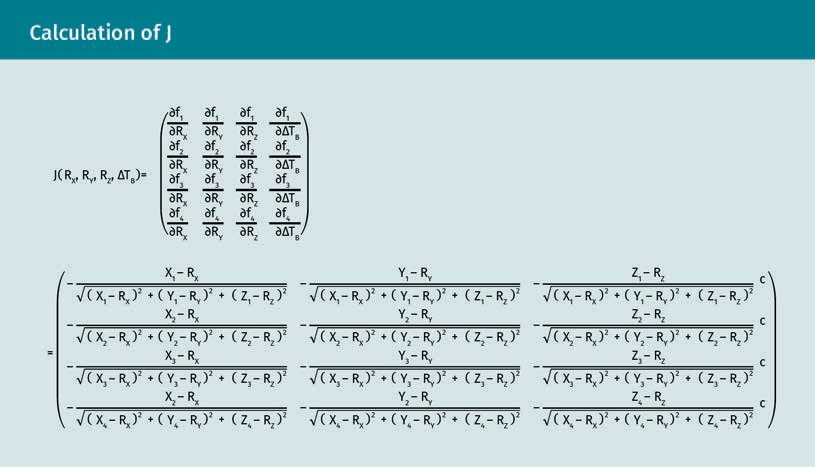
**Positioning by One GPS Satellite**

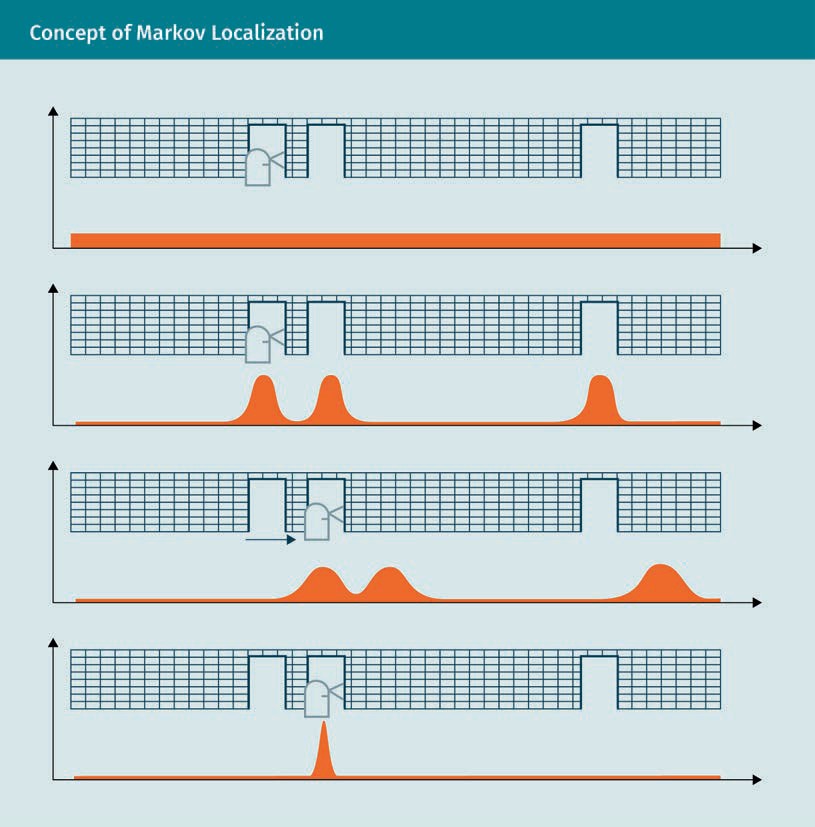
**Narrowing Position Down to Two Possible Points**

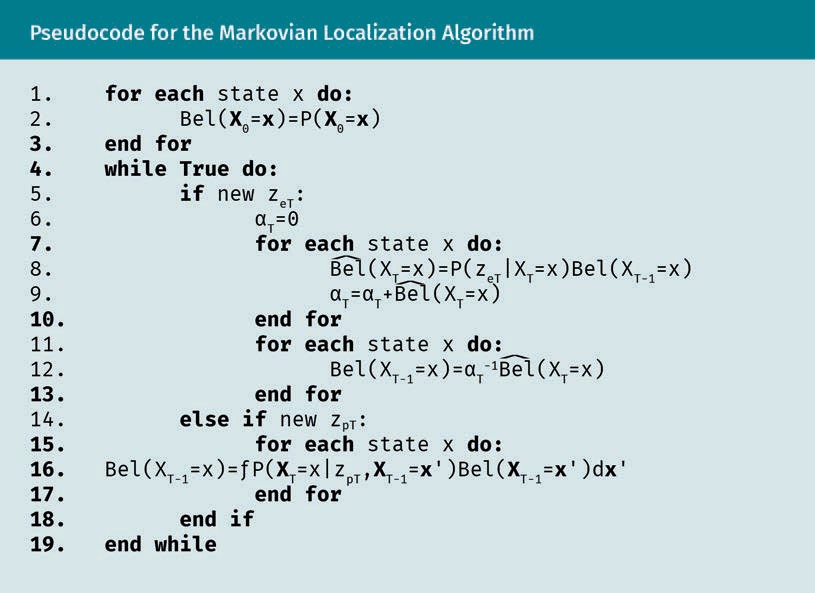


**Final Positioning of GPS Receiver Using Three Satellites**

**Position Determionation in the Three-Dimensional Space**

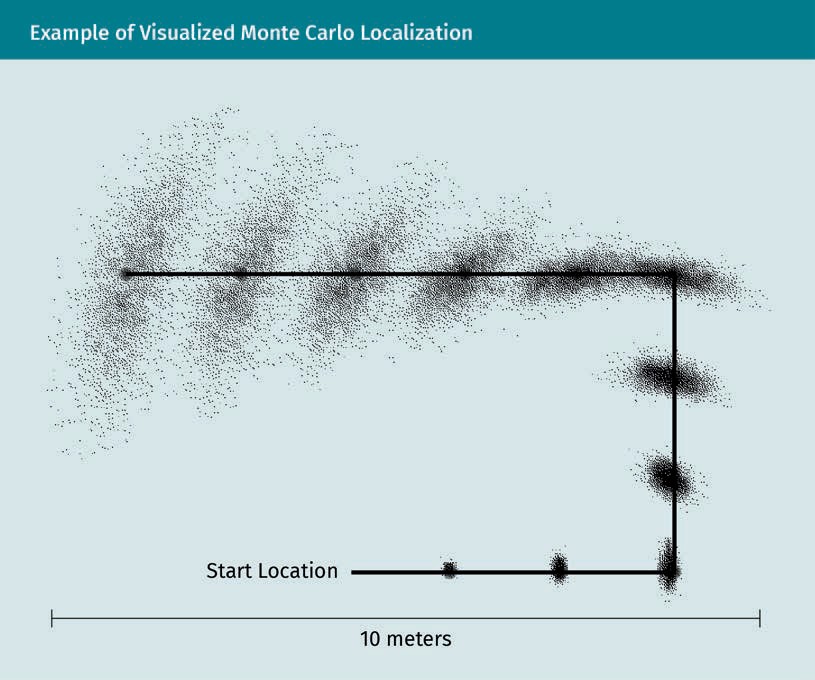
**Calculation of J**

**Concept of Markov Localization**

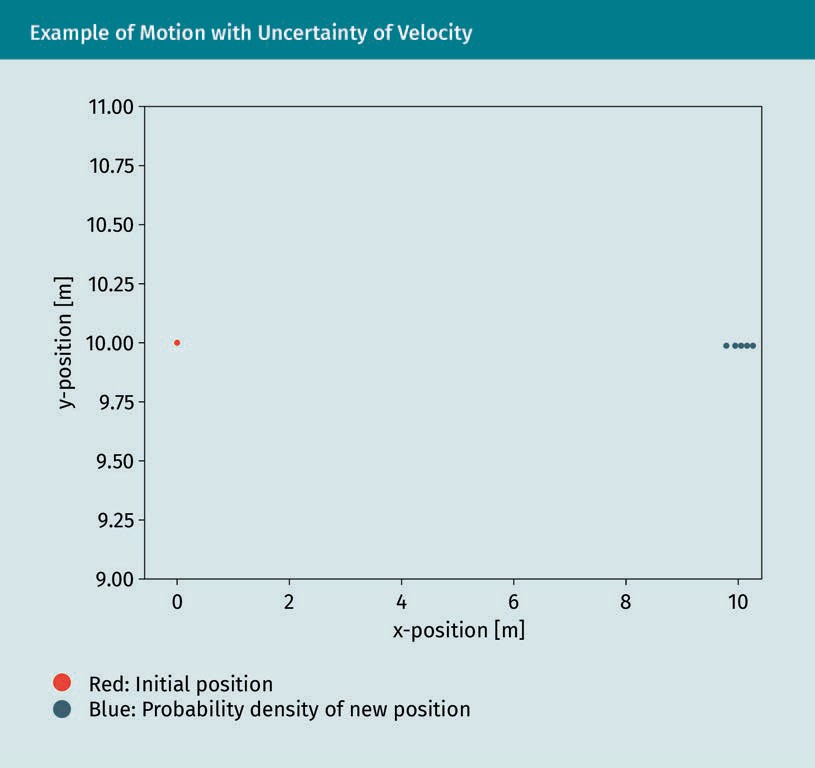
**Pseudocode for the Markovian Localization Algorithm**

|  |  |
| --- | --- |
| for each state x do: |  |
| Bel (XΘ=x) |  |
| end for |  |
| while True do: |  |
| if new zeT: |  |
| αT=Θ |  |
| for each state x do: |  |
|  |  |
|  |  |
| end for |  |
|  |  |
| else if new zpT: |  |
|  |  |
| end for |  |
| end if |  |
| end while |  |

**Example of Visualized Monte Carlo Localization**

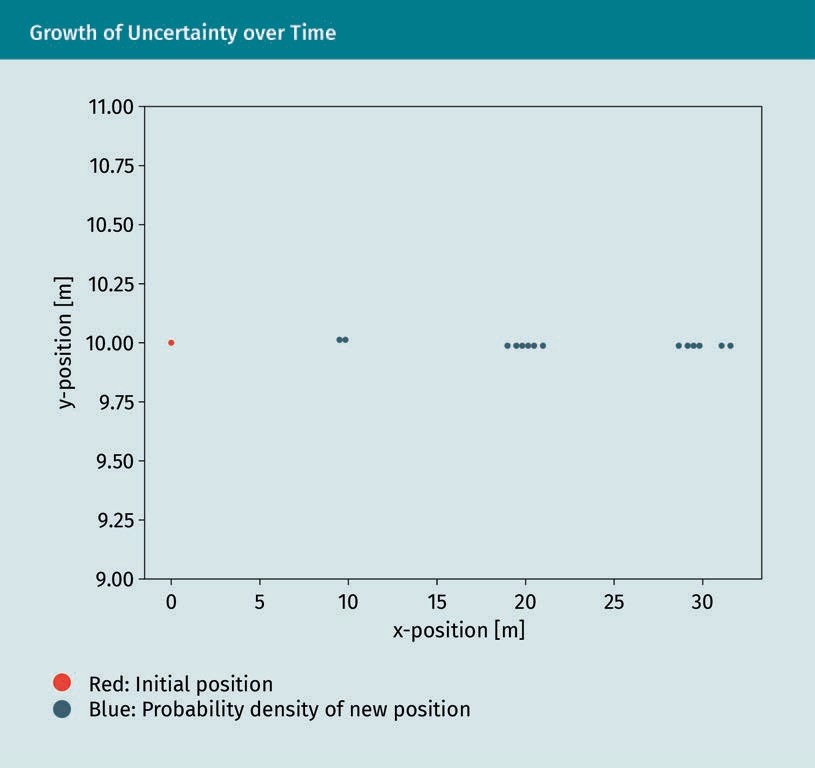


|  |  |
| --- | --- |
| Start Location |  |
| 10 meters |  |

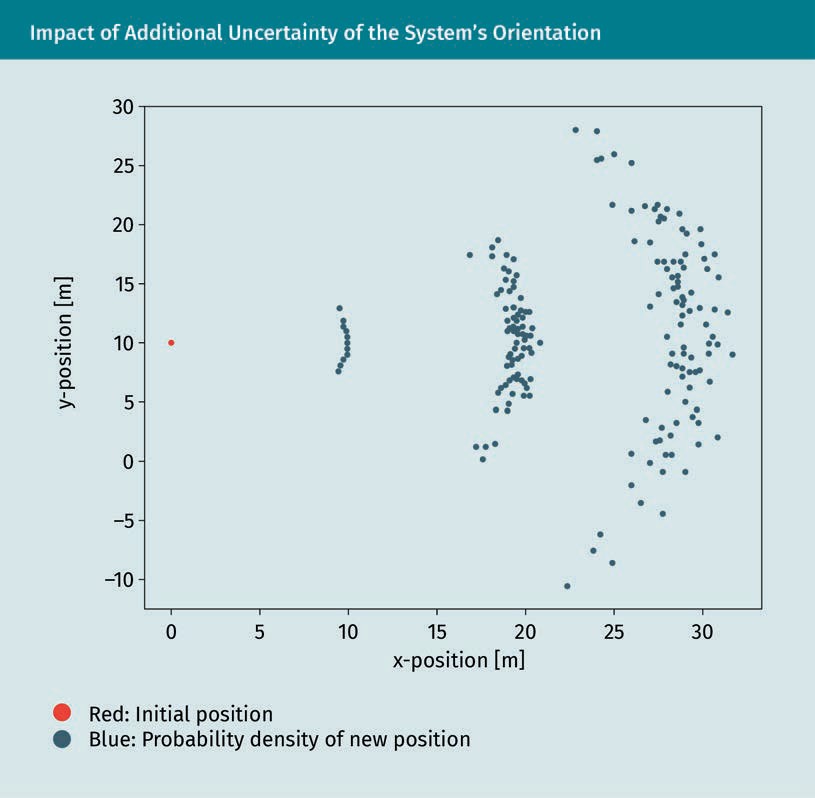
**Example of Motion with Uncertainty of Velocity**

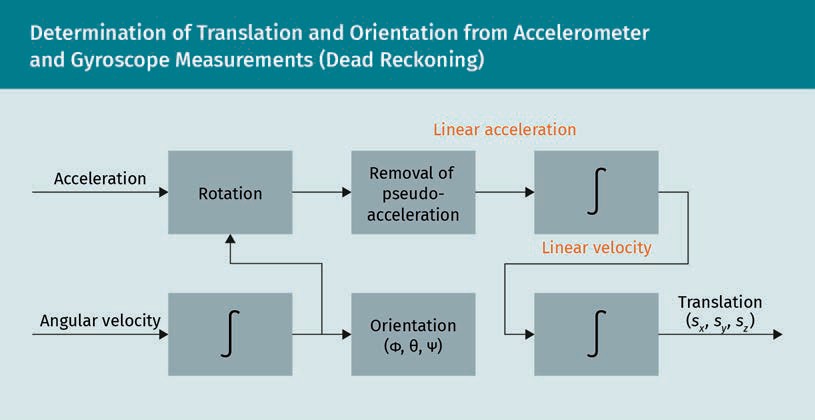
|  |  |
| --- | --- |
| y-position [m] |  |
| x-position [m] |  |
| Red: Initial position |  |
| Blue: Probability density of new position |  |

**Growth of uncertainty over time**

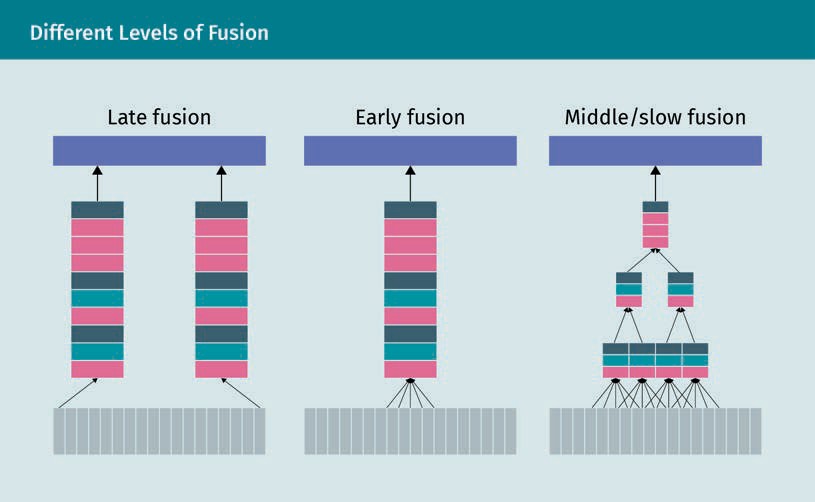


**Impact of Additional Uncertainty of the System’s Orientation**

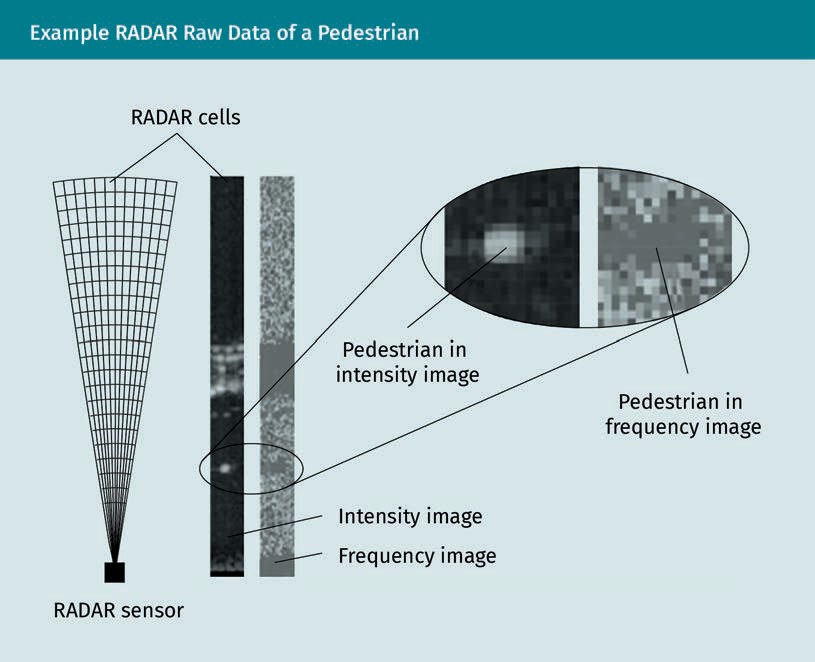


**Determination of Translation and Orientation from Acelerometer and Gyroscope Measurements (Dead Reckoning)**

|  |  |
| --- | --- |
| Acceleration |  |
| Rotation |  |
| Removal of pseudo-acceleration |  |
| Linear acceleration |  |
| Linear velocity |  |
| Angular velocity |  |
| Orientation |  |
| Translation |  |

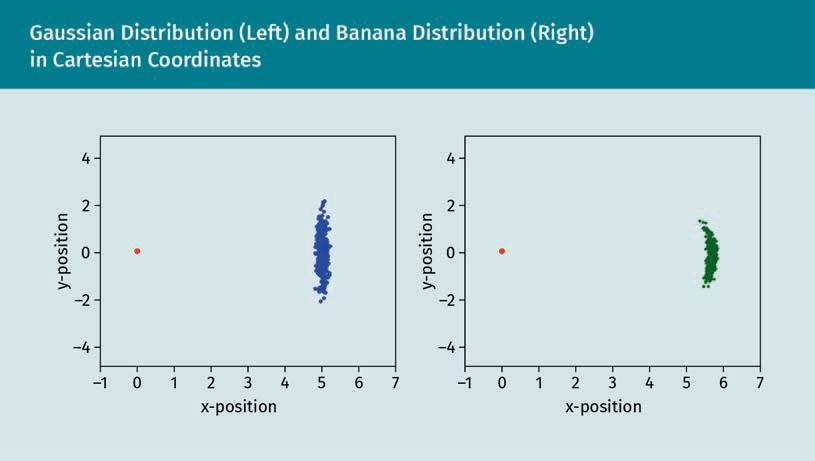
**Different Levels of Fusion**

|  |  |
| --- | --- |
| Late fusion |  |
| Early fusion |  |
| Middle/slow fusion |  |

**Example RADAR Raw Data of a Pedestrian**

|  |  |
| --- | --- |
| RADAR cells |  |
| RADAR sensor |  |
| Pedestrian in intensity image |  |
| Pedestrian in frequency image |  |
| Intensity image |  |
| Frequency image |  |

**Gaussian Distribution (Left) and Banana Distribution (Right) in Cartesian Coordinates**



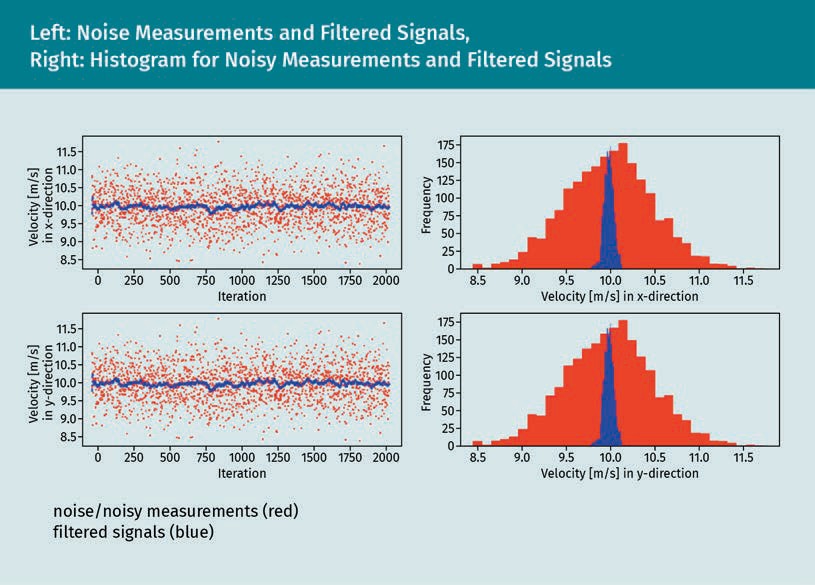
|  |  |
| --- | --- |
| y-position |  |
| x-position |  |

**Example of Python Implementation**

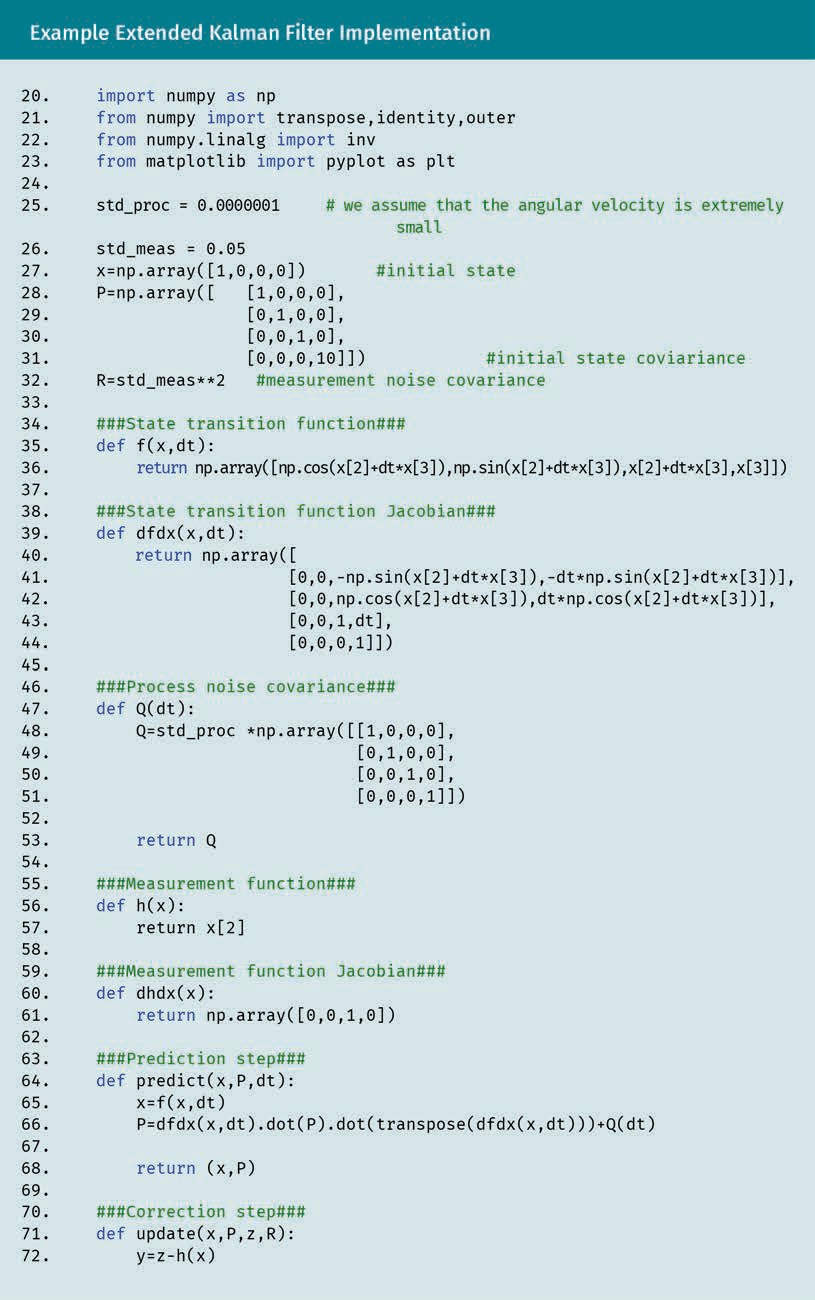


|  |  |
| --- | --- |
| import numpy as np |  |
| from numpy import transpose, indentity, outer |  |
| from numpy.linalg import inv |  |
| from matplotlib import pyplot as plt |  |
| std\_acc = 0.001 |  |
| #we assume a standard deviation of the acceleration of 0.001 (m/s^2) |  |
| std\_meas = 0.3 |  |
| #standard deviation of the measurement |  |
| x=np.array([0,0]).transpose |  |
| #initial state |  |
| P=10\*identity(“) |  |
| #initial state covariance |  |
| R=std\_meas\*\*2 |  |
| #measurement noise |  |
| ###State transition matrix### |  |
| def F(dt): |  |
| return np.array([[1,dt],[0,1]]) |  |
| ###Process noise covariance### |  |
| def Q(dt): |  |
| Q=std\_acc\*np.array([[dt\*\*3/3,dt\*\*2/2],[dt\*\*2/2,dt]]) |  |
| return Q |  |
| ###Measurement model### |  |
| def H(): |  |
| return np.array([0,1]) |  |
| ###Prediction step### |  |
| def predict(x,P,dt): |  |
| x=F(dt).dot(x) |  |
| P=F(dt).dot(P).dot(transpose(F(dt)))+Q(dt) |  |
| return (x,P) |  |
| ###Correction step### |  |
| def update (x,P,z,R): |  |
| y=z-H().dot(x) |  |
| S=H().dot(P).dot(transpose(H()))+R |  |
| K=P.dot(transpose(H))).dot(S++(-1)) |  |
| x=x+K\*y |  |
| P=(identity(2)-outer(K,H())).dot(P) |  |
| return (x,P) |  |

**Left: Noise Measurements and Filtered Signals, Right: Histogram for**

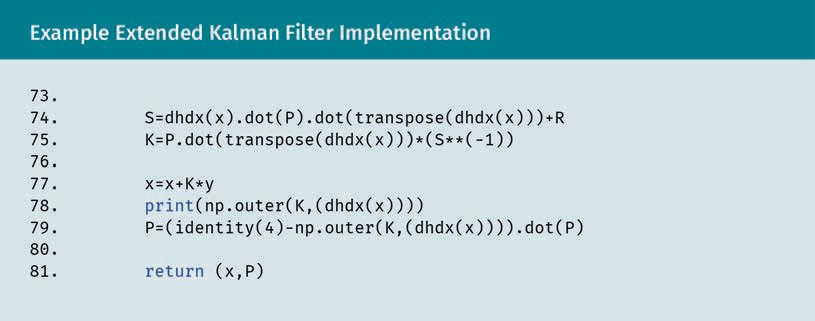
**Noisy Measurements and Filtered Signals**

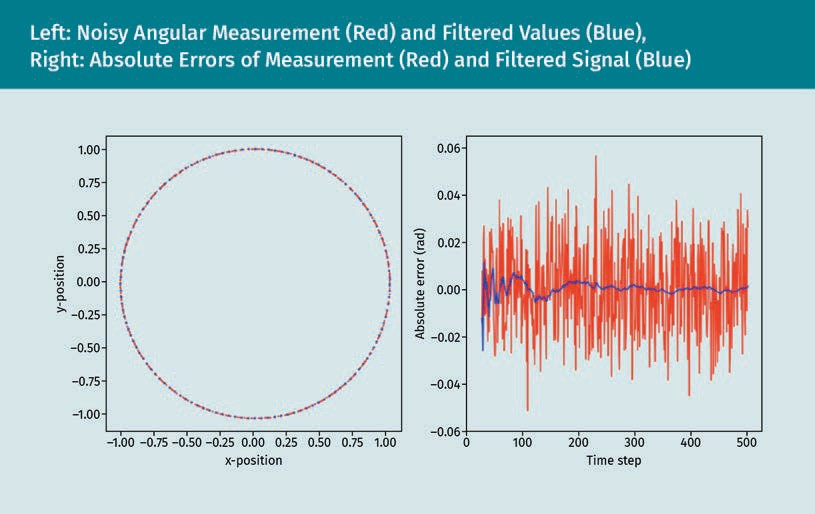
|  |  |
| --- | --- |
| Velocity [m/s] in x-direction |  |
| Velocity [m/s] in y-direction |  |
| Iteration |  |
| Frequency |  |

**Example Extended Kalman Filter Implementation**

|  |  |
| --- | --- |
| **import** numpy as np |  |
| **from** numpy **import** transpose,identity,outer |  |
| **from** numpy.linalg **import** inv |  |
| **from** matplotlib **import** pyplot as plt |  |
| std\_proc = 0.0000001 |  |
| # we assume that the angular  velocity is extremely small |  |
| std\_meas = 0.05 |  |
| x=np.array([1,0,0,0])  #initial state |  |
| P=np.array([ [1,0,0,0], |  |
| [0,1,0,0], |  |
| [0,0,1,0], |  |
| [0,0,0,10]])  #initial state coviariance |  |
| R=std\_meas\*\*2  #measurement noise covariance |  |
| ###State transition function### |  |
| **def** f(x,dt): |  |
| **return** np.array  ([np.cos(x[2]+dt\*x[3]),  np.sin(x[2]+dt\*x[3]),x[2]+dt\*x[3],x[3]]) |  |
| ###State transition function Jacobian### |  |
| **def** dfdx(x,dt): |  |
| **return** np.array([ |  |
| [0,0,-np.sin(x[2]+dt\*x[3]),  -dt\*np.sin(x[2]+dt\*x[3])], |  |
| [0,0,np.cos(x[2]+dt\*x[3]),  dt\*np.cos(x[2]+dt\*x[3])], |  |
| [0,0,1,dt], |  |
| [0,0,0,1]]) |  |
| ###Process noise covariance### |  |
| **def** Q(dt): |  |
| Q=std\_proc \*np.array([[1,0,0,0], |  |
| [0,1,0,0], |  |
| [0,0,1,0], |  |
| [0,0,0,1]]) |  |
| **return** Q |  |
| ###Measurement function### |  |
| **def** h(x): |  |
| **return** x[2] |  |
| ###Measurement function Jacobian### |  |
| **def** dhdx(x): |  |
| **return** np.array([0,0,1,0]) |  |
| ###Prediction step### |  |
| **def** predict(x,P,dt): |  |
| x=f(x,dt) |  |
| P=dfdx(x,dt).dot(P).dot  (transpose(dfdx(x,dt)))+Q(dt) |  |
| **return** (x,P) |  |
| ###Correction step### |  |
| **def** update(x,P,z,R): |  |
| y=z-h(x) |  |
| S=dhdx(x).dot(P).dot(transpose(dhdx(x)))+R |  |
| K=P.dot(transpose(dhdx(x)))\*(S\*\*(-1)) |  |
| x=x+K\*y |  |
| **print**(np.outer(K,(dhdx(x)))) |  |
| P=(identity(4)-np.outer(K,(dhdx(x)))).dot(P) |  |
| **return** (x,P) |  |

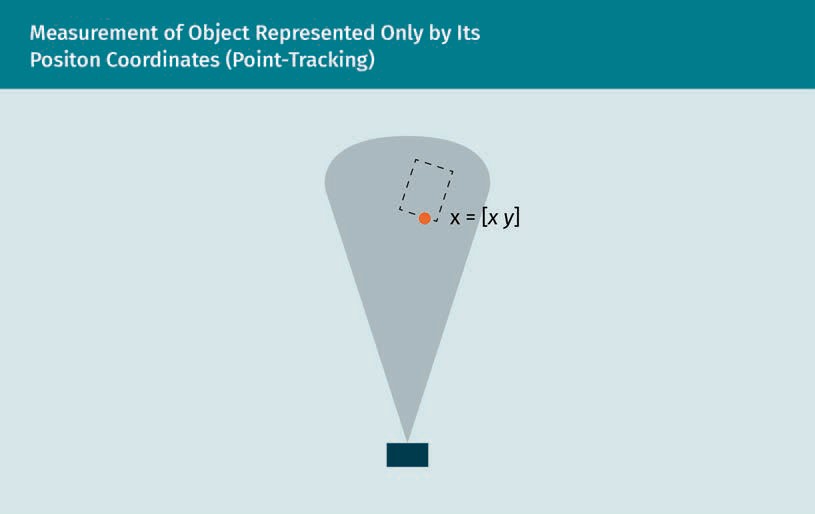
**Example Extended Kalman Filter Implementation**

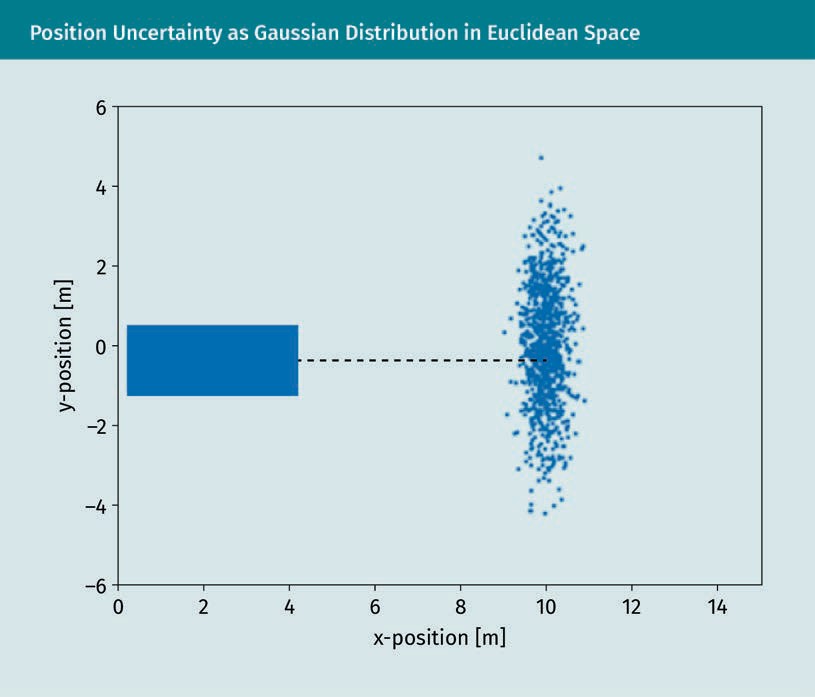


**Left: Noisy Angular Measurement (Red) and Filtered Values (Blue), Right: Absolute Errors of Measurement (Red) and Filtered Signal (Blue)**

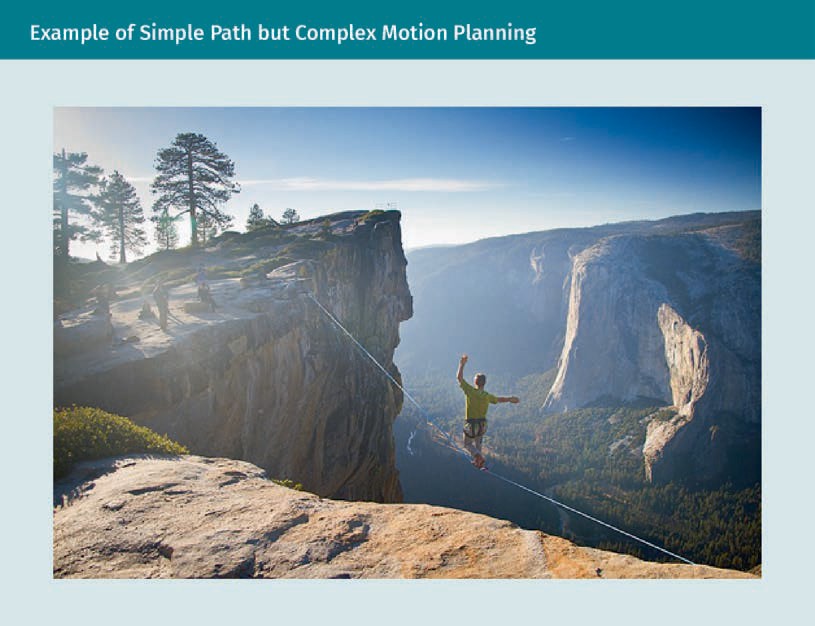
|  |  |
| --- | --- |
| y-position |  |
| x-position |  |
| Absolute error (rad) |  |
| Time step |  |

**Measurement of Object Represented Only by Its Position Coordinates (Point-Tracking)**

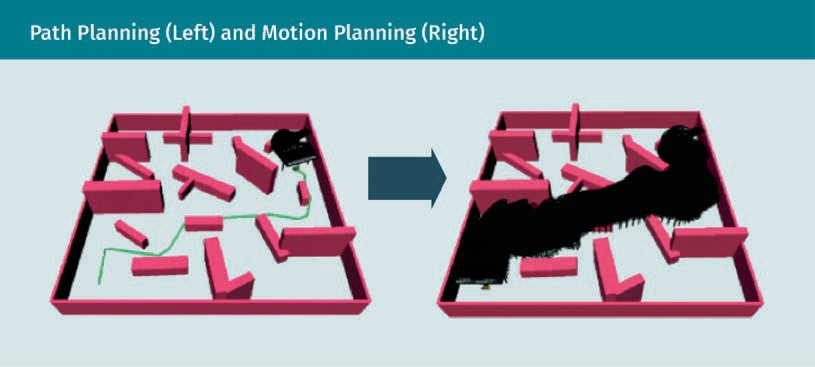


**Position Uncertainty as Gaussian Distribution in Euclidean Space**

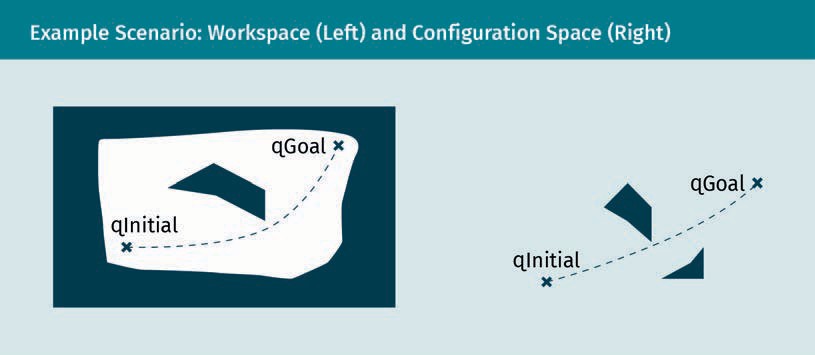
|  |  |
| --- | --- |
| y-position [m] |  |
| x-position [m] |  |

**Example of Simple Path but Complex Motion Planning**

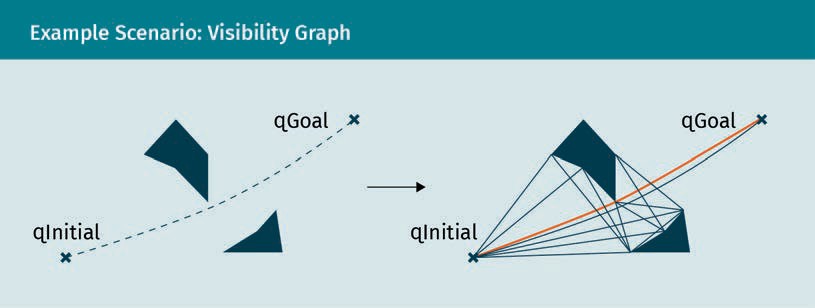
**Path Planning (Left) and Motion Planning (Right)**



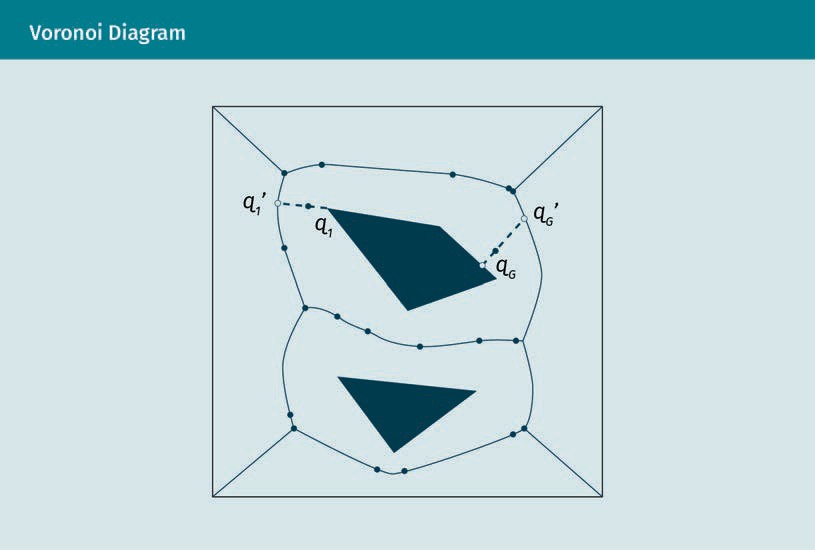
**Example Scenario: Workspace (Left) and Configuration Space (Right)**

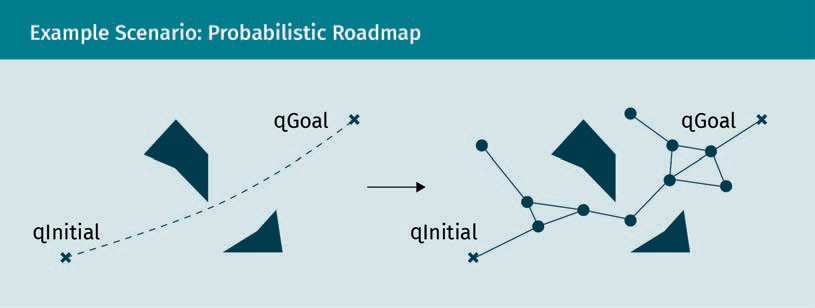


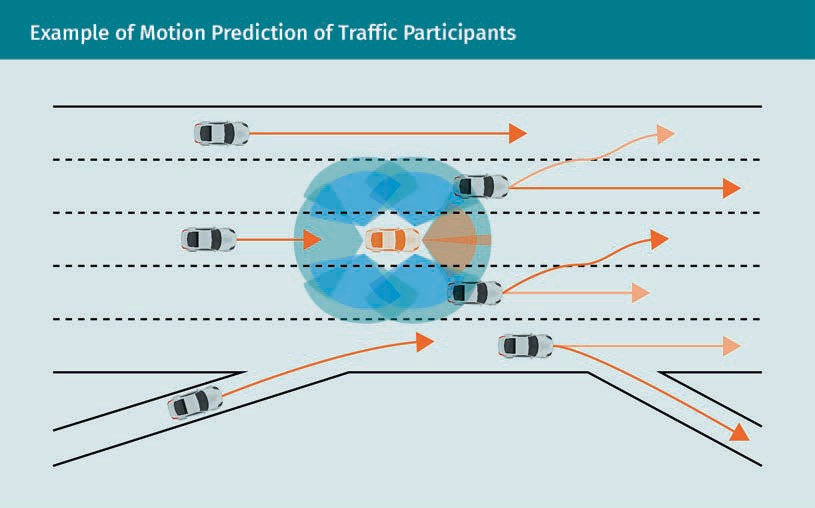
|  |  |
| --- | --- |
| qInitial |  |
| qGoal |  |

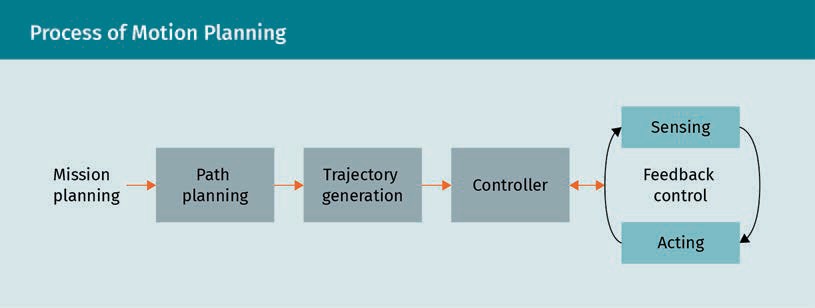
**Example Scenario: Visibility Graph**

**Voronio Diagram**

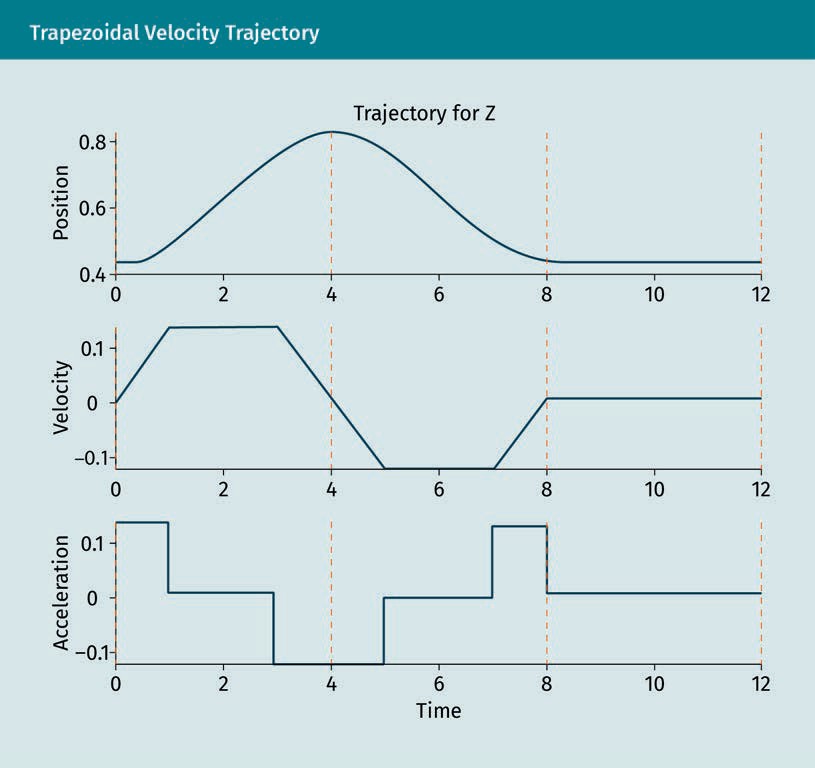


**Example Scenario: Probabilistic Roadmap**

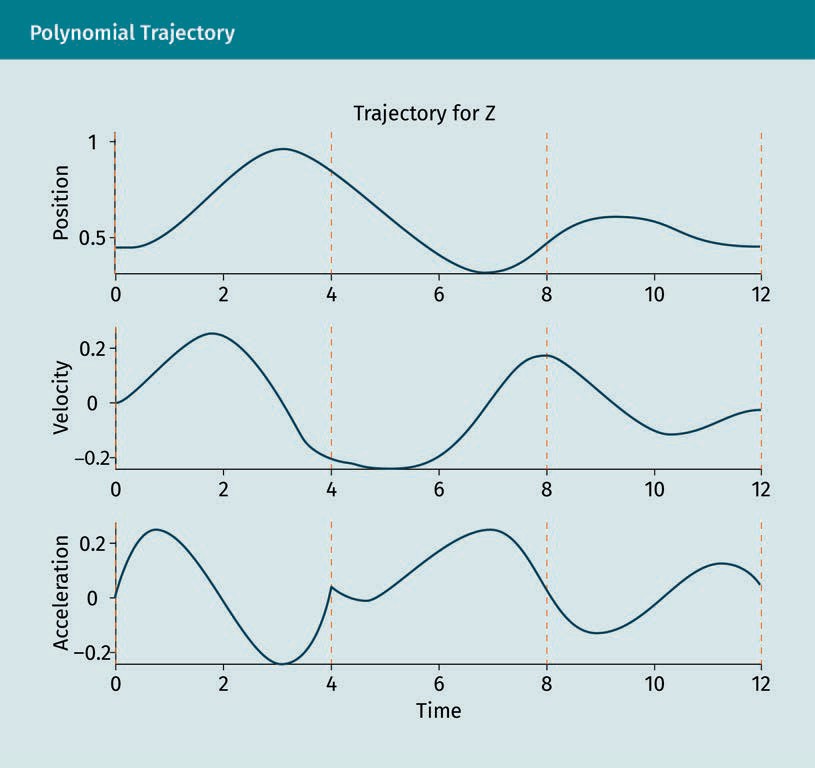
**Example of Motion Prediction of Traffic Participants**

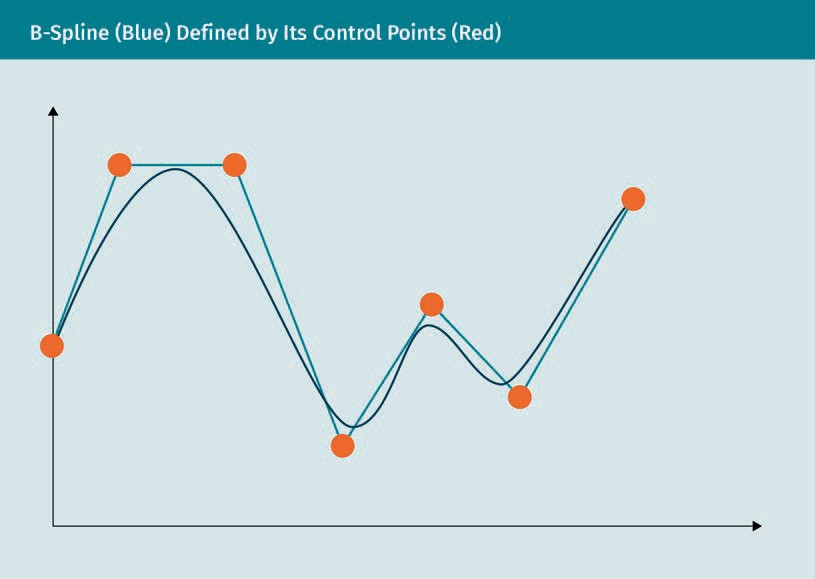
**Process of Motion Planning**

|  |  |
| --- | --- |
| Mission planning |  |
| Path planning |  |
| Trajectory generation |  |
| Controller |  |
| Sensing |  |
| Feedback Control |  |
| Acting |  |

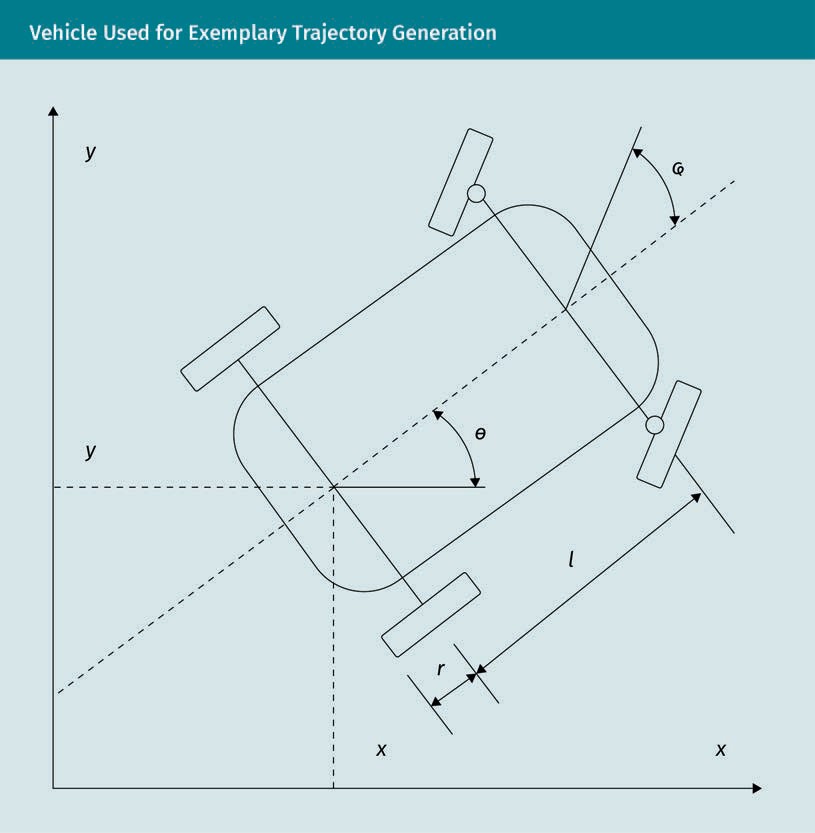
**Trapezoidal Velocity Trajectory**

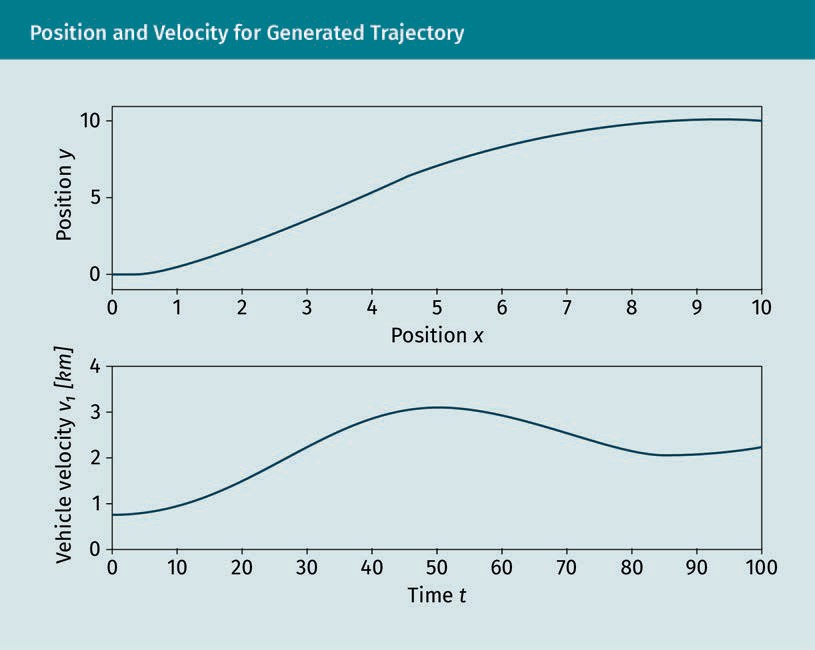
|  |  |
| --- | --- |
| Trajectory for Z |  |
| Acceleration |  |
| Velocity |  |
| Position |  |
| Time |  |

**Polynomial Trajectory**

**B-Spline (Blue) Defined by Its Control Points (Red)**

**Vehicle Used for Exemplary Trajectory Generation**



**Position and Velocity for Generated Trajectory**

|  |  |
| --- | --- |
| Position *y* |  |
| Vehicle velocity v1 *[km]* |  |
| Position *x* |  |
| Time *t* |  |