**Process and Implementation**

As an accompaniment to the videos we will follow the particle filter algorithm process and implementation details.

**Particle Filter Algorithm Steps and Inputs**

The flowchart below represents the steps of the particle filter algorithm as well as its inputs.

Diagram

Description automatically generated

Particle Filter Algorithm Flowchart

**Psuedo Code**

This is an outline of steps you will need to take with your code in order to implement a particle filter for localizing an autonomous vehicle. The pseudo code steps correspond to the steps in the algorithm flow chart, initialization, prediction, particle weight updates, and resampling. Python implementation of these steps was covered in the previous lesson.

Text

Description automatically generated

Initialization

At the initialization step we estimate our position from GPS input. The subsequent steps in the process will refine this estimate to localize our vehicle.

Text

Description automatically generated

Prediction

During the prediction step we add the control input (yaw rate & velocity) for all particles

Text

Description automatically generated

Update

During the update step, we update our particle weights using map landmark positions and feature measurements.

Text

Description automatically generated

Resampling

During resampling we will resample M times (M is range of 0 to length\_of\_particleArray) drawing a particle i (i is the particle index) proportional to its weight . Sebastian covered one implementation of this in his [discussion and implementation of a resampling wheel](https://classroom.udacity.com/nanodegrees/nd013/parts/40f38239-66b6-46ec-ae68-03afd8a601c8/modules/2c318113-724b-4f9f-860c-cb334e6e4ad7/lessons/4d7950f7-f519-4dc6-9142-052a1261f5bf/concepts/487480820923).

Text

Description automatically generated

Return New Particle Set

The new set of particles represents the Bayes filter posterior probability. We now have a refined estimate of the vehicles position based on input evidence.

Diagram

Description automatically generated

The most practical way to initialize our particles and generate real time output, is to make an initial estimate using GPS input. As with all sensor based operations, this step is impacted by noise.

## Project Implementation

* Particles shall be implemented by sampling a Gaussian distribution, taking into account Gaussian sensor noise around the initial GPS position and heading estimates.
* Use the [C++ standard library normal distribution](http://en.cppreference.com/w/cpp/numeric/random/normal_distribution) and [C++ standard library random engine](http://www.cplusplus.com/reference/random/default_random_engine) functions to sample positions around GPS measurements.

Diagram

Description automatically generated

Now that we have initialized our particles it's time to predict the vehicle's position. Here we will use what we learned in the motion models lesson to predict where the vehicle will be at the next time step, by updating based on yaw rate and velocity, while accounting for Gaussian sensor noise.

Diagram

Description automatically generated

# Instructor Notes

Note that for a vector \mathbf{v} = \text{(x,y)}**v**=(x,y), \vert \mathbf{v} \vert∣**v**∣ is used here to denote the vector length or magnitude = \sqrt{x^2 + y^2}*x*2+*y*2​.

For a difference between two vectors, \mathbf{v} = {(x\_v,y\_v)}**v**=(*xv*​,*yv*​) and \mathbf{w} = {(x\_w,y\_w)}**w**=(*xw*​,*yw*​), the magnitude \vert \mathbf{v - w} \vert∣**v**−**w**∣ is also the Euclidean distance between these two vectors = \sqrt{(x\_v - x\_w)^2 + (y\_v - y\_w)^2}(*xv*​−*xw*​)2+(*yv*​−*yw*​)2​

In this case \sqrt{\vert p\_i - g \vert}∣*pi*​−*g*∣​ is meant to note the root squared error of a particular particle, resulting in the general form described in the quiz:

\sqrt{(x\_p - x\_g)^2 + (y\_p - y\_g)^2}(*xp*​−*xg*​)2+(*yp*​−*yg*​)2​

where:

* Position RMSE = \sqrt{(x\_p - x\_g)^2 + (y\_p - y\_g)^2}(*xp*​−*xg*​)2+(*yp*​−*yg*​)2​
* Theta RMSE = \sqrt{(\theta\_p - \theta\_g)^2}(*θp*​−*θg*​)2​

### Root Squared Error Quiz

Given that the car’s ground truth position was (x, y, theta) = (5.2 m, 19.3 m, pi/16) and the best particle’s position was (x, y, theta) = (5 m, 18.7 m, pi/8), what is the error?

Position RMSE = \sqrt{(x-x\_{meas})^2+(y-y\_{meas})^2}(*x*−*xmeas*​)2+(*y*−*ymeas*​)2​

Theta RMSE = \sqrt{(\theta-\theta\_{meas})^2}(*θ*−*θmeas*​)2​

# Transformations and Associations

In the project you will need to correctly perform observation measurement transformations, along with identifying measurement landmark associations in order to correctly calculate each particle's weight. Remember, our ultimate goal is to find a weight parameter for each particle that represents how well that particle fits to being in the same location as the actual car.

In the quizzes that follow we will be given a single particle with its position and heading along with the car's observation measurements. We will first need to transform the car's measurements from its local car coordinate system to the map's coordinate system. Next, each measurement will need to be [associated with a landmark identifier](https://classroom.udacity.com/nanodegrees/nd013/parts/40f38239-66b6-46ec-ae68-03afd8a601c8/modules/2c318113-724b-4f9f-860c-cb334e6e4ad7/lessons/5c50790c-5370-4c80-aff6-334659d5c0d9/concepts/44dc964a-7cff-4b31-b0b2-94b90d68b96b), for this part we will take the closest landmark to each transformed observation. Finally, we will use this information to calculate the weight value of the particle.

Chart

Description automatically generated

In the graph above we have a car (**ground truth position**) that observes three nearby landmarks, each one labeled OBS1, OBS2, OBS3. Each observation measurement has x, and y values in the car's coordinate system. We have a particle "P" (**estimated position of the car**) above with position (4,5) on the map with heading -90 degrees. The first task is to transform each observation marker from the vehicle's coordinates to the map's coordinates, with respect to our particle.

Here is another example that might help your intuition.

Referring to the figures below:

Suppose the map coordinate system (grey lines) and the vehicle coordinate system (orange lines) are offset, as depicted below. If we know the location of the observation in vehicle coordinates (grey lines), we would need to rotate the entire system, observation included, -45 degrees to find it in map coordinates (grey lines), Once this rotation is done, we can easily see the location of the observation in map coordinates.

## Particle (blue dot) in Map Frame (grey)

Shape

Description automatically generated

## Particle (blue dot) in Vehicle Frame (orange)

A picture containing dark, light

Description automatically generated

Chart

Description automatically generated

Map with Car Observations and Particle

In the graph above we have a car (**ground truth position**) that observes three nearby landmarks, each one labeled OBS1, OBS2, OBS3. Each observation measurement has x, and y values in the car's coordinate system. We have a particle "P" (**estimated position of the car**) above with position (4,5) on the map with heading -90 degrees. The first task is to transform each observation marker from the vehicle's coordinates to the map's coordinates, with respect to our particle.

## Transformations

There is a generalized trigonometric function that given any particle position and heading along with any observation measurement (x\_obs,y\_obs) will output the transformed observation (x\_map,y\_map) for that particle. In other words, we need to map particle coordinates to map coordinates, by passing particle and observation coordinates through a function.

**Transformed Observation (x\_map,y\_map)** = func(x\_particle, y\_particle, heading\_particle, x\_obs, y\_obs)

You will need to derive and use this function to efficiently calculate particle observation transformations in the project.

* Some excellent resources discussing coordinate transformation and rotation matrices:
  + [Coordinate Transformations](http://farside.ph.utexas.edu/teaching/336k/Newtonhtml/node153.html)
  + [Coordinate Transformation Under Rotation](https://www.miniphysics.com/coordinate-transformation-under-rotation.html)

Chart

Description automatically generated

Map with Car Observations and Particle

Observations in the car coordinate system can be transformed into map coordinates (\text{x}\_mx*m*​ and \text{y}\_my*m*​) by passing car observation coordinates (\text{x}\_cx*c*​ and \text{y}\_cy*c*​), map particle coordinates (\text{x}\_px*p*​ and \text{y}\_py*p*​), and our rotation angle (-90 degrees) through a homogenous transformation matrix. This homogenous transformation matrix, shown below, performs rotation and translation.

## Homogenous Transformation

\left[ \begin{array}{c} \text{x}\_m \\ \text{y}\_m \\ 1 \end{array} \right] = \begin{bmatrix} \cos\theta & -\sin\theta & \text{x}\_p \\ \sin\theta & \cos\theta & \text{y}\_p \\ 0 & 0 & 1 \end{bmatrix} \times \left[ \begin{array}{c} \text{x}\_c \\ \text{y}\_c \\ 1 \end{array} \right]⎣⎡​x*m*​y*m*​1​⎦⎤​=⎣⎡​cos*θ*sin*θ*0​−sin*θ*cos*θ*0​x*p*​y*p*​1​⎦⎤​×⎣⎡​x*c*​y*c*​1​⎦⎤​

[Matrix multiplication](https://www.mathsisfun.com/algebra/matrix-multiplying.html) results in:

\text{x}\_m= \text{x}\_p + (\cos\theta \times \text{x}\_c) - (\sin\theta \times \text{y}\_c)x*m*​=x*p*​+(cos*θ*×x*c*​)−(sin*θ*×y*c*​)

\text{y}\_m= \text{y}\_p + (\sin\theta \times \text{x}\_c) + (\cos\theta \times \text{y}\_c)y*m*​=y*p*​+(sin*θ*×x*c*​)+(cos*θ*×y*c*​)

# Quiz Solutions

## Observation 1 Solution

**#include <cmath>**

**#include <iostream>**

**int** **main**() {

*// define coordinates and theta*

**double** x\_part, y\_part, x\_obs, y\_obs, theta;

x\_part = 4;

y\_part = 5;

x\_obs = 2;

y\_obs = 2;

theta = -M\_PI/2; *// -90 degrees*

*// transform to map x coordinate*

**double** x\_map;

x\_map = x\_part + (cos(theta) \* x\_obs) - (sin(theta) \* y\_obs);

*// transform to map y coordinate*

**double** y\_map;

y\_map = y\_part + (sin(theta) \* x\_obs) + (cos(theta) \* y\_obs);

*// (6,3)*

std::cout << **int**(round(x\_map)) << ", " << **int**(round((y\_map)) << std::endl;

**return** 0;

}

## Observation 2 Solution

**#include <cmath>**

**#include <iostream>**

**int** **main**() {

*// define coordinates and theta*

**double** x\_part, y\_part, x\_obs, y\_obs, theta;

x\_part = 4;

y\_part = 5;

x\_obs = 3;

y\_obs = -2;

theta = -M\_PI/2; *// -90 degrees*

*// transform to map x coordinate*

**double** x\_map;

x\_map = x\_part + (cos(theta) \* x\_obs) - (sin(theta) \* y\_obs);

*// transform to map y coordinate*

**double** y\_map;

y\_map = y\_part + (sin(theta) \* x\_obs) + (cos(theta) \* y\_obs);

*// (2,2)*

std::cout << **int**(round(x\_map)) << ", " << **int**(round(y\_map)) << std::endl;

**return** 0;

}

## Observation 3 Solution

**#include <cmath>**

**#include <iostream>**

**int** **main**() {

*// define coordinates and theta*

**double** x\_part, y\_part, x\_obs, y\_obs, theta;

x\_part = 4;

y\_part = 5;

x\_obs = 0;

y\_obs = -4;

theta = -M\_PI/2; *// -90 degrees*

*// transform to map x coordinate*

**double** x\_map;

x\_map = x\_part + (cos(theta) \* x\_obs) - (sin(theta) \* y\_obs);

*// transform to map y coordinate*

**double** y\_map;

y\_map = y\_part + (sin(theta) \* x\_obs) + (cos(theta) \* y\_obs);

*// (0,5)*

std::cout << **int**(round(x\_map)) << ", " << **int**(round(y\_map)) << std::endl;

**return** 0;

}

Chart

Description automatically generated

Map with Car Observations and Particle

## Associations

Now that observations have been transformed into the map's coordinate space, the next step is to associate each transformed observation with a land mark identifier. In the map exercise above we have 5 total landmarks each identified as L1, L2, L3, L4, L5, and each with a known map location. We need to associate each transformed observation TOBS1, TOBS2, TOBS3 with one of these 5 identifiers. To do this we must associate the closest landmark to each transformed observation.

As a reminder:

TOBS1 = (6,3), TOBS2 = (2,2) and TOBS3 = (0,5).

# Calculating the Particle's Final Weight

Now we that we have done the measurement transformations and associations, we have all the pieces we need to calculate the particle's final weight. The particles final weight will be calculated as the product of each measurement's Multivariate-Gaussian probability density.

The Multivariate-Gaussian probability density has two dimensions, x and y. The mean of the Multivariate-Gaussian is the measurement's associated landmark position and the Multivariate-Gaussian's standard deviation is described by our initial uncertainty in the x and y ranges. The Multivariate-Gaussian is evaluated at the point of the transformed measurement's position. The formula for the Multivariate-Gaussian can be seen below.

To complete the next set of quizzes, calculate each measurement's Multivariate-Gaussian probability density using the formula above and the previously calculated values. In this example the standard deviation for both x and y is 0.3.

Note that x and y are the observations in map coordinates from the landmarks quiz and *μx*​, *μy*​ are the coordinates of the nearest landmarks. These should correspond to the correct responses from previous quizzes.

**Resources, Tips, and Hints**

* The elements of the formula above can be entered into a calculator but consider writing a function. This will be a valuable addition to your toolkit.
* Recall that x and y are the observations in map coordinates from the landmarks quiz and \mu\_x*μx*​, \mu\_y*μy*​ are the coordinates of the nearest landmarks. These should correspond to the correct responses from previous quizzes.

**Particle's Final Weight**

To get the final weight just multiply all the calculated measurement probabilities together.

We can calculate particle weights using the following equation:

* Recall that for this example the standard deviation for both x and y is 0.3.
* x and y are the observations in map coordinates from landmarks quiz and \mu\_x*μx*​, \mu\_y*μy*​ are the coordinates of the nearest landmarks. These should correspond to the correct responses from previous quizzes.

**Quiz Solutions**

OBS1 weight should be around 0.00683644777551 rounding to 6.84E-3.

OBS2 weight should be around 0.00683644777551, rounding to 6.84E-3.

OBS3 weight should be around 9.83184874151e-49, rounding to 9.83E-49.

The final weight should be around 4.59112934464959e-53, rounding to 4.60E-53 (the above three multiplied together).

We have included a C++ solution below; the multiv\_gauss.cpp file contains the primary calculations.

### Explanation of Project Code

We are constantly updating and improving our projects so the code you see may be slightly different, such as M in the video now being named num\_particles. Refer to the repository for the most up to date instructions.

Go to the next lesson to read more about the project and get the link to the github repo.

## Additional Resources on Localization

Nice work reaching the end of the localization content! While you still have the project left to do here, we're also providing some additional resources and recent research on the topic that you can come back to if you have time later on.

Reading research papers is a great way to get exposure to the latest and greatest in the field, as well as expand your learning. However, just like the project ahead, it's often best to learn by doing - if you find a paper that really excites you, try to implement it (or even something better) yourself!

##### Optional Reading

All of these are completely optional reading - you could spend hours reading through the entirety of these! We suggest moving onto the project first so you have what you’ve learned fresh on your mind, before coming back to check these out.

We've categorized these papers to hopefully help you narrow down which ones might be of interest, as well as highlighted a couple key reads by category by including their Abstract section, which summarizes the paper.

### Simultaneous Localization and Mapping (SLAM)

The below papers cover Simultaneous Localization and Mapping (SLAM) - which as the name suggests, combines localization and mapping into a single algorithm without a map created beforehand.

[Past, Present, and Future of Simultaneous Localization And Mapping: Towards the Robust-Perception Age](https://arxiv.org/abs/1606.05830) by C. Cadena, et. al.

***Abstract:****Simultaneous Localization and Mapping (SLAM) consists in the concurrent construction of a model of the environment (the map), and the estimation of the state of the robot moving within it. The SLAM community has made astonishing progress over the last 30 years, enabling large-scale real-world applications, and witnessing a steady transition of this technology to industry. We survey the current state of SLAM. We start by presenting what is now the de-facto standard formulation for SLAM. We then review related work, covering a broad set of topics including robustness and scalability in long-term mapping, metric and semantic representations for mapping, theoretical performance guarantees, active SLAM and exploration, and other new frontiers. [...]*

[Navigating the Landscape for Real-time Localisation and Mapping for Robotics and Virtual and Augmented Reality](https://arxiv.org/abs/1808.06352) by S. Saeedi, et. al.

***Abstract:****Visual understanding of 3D environments in real-time, at low power, is a huge computational challenge. Often referred to as SLAM (Simultaneous Localisation and Mapping), it is central to applications spanning domestic and industrial robotics, autonomous vehicles, virtual and augmented reality. This paper describes the results of a major research effort to assemble the algorithms, architectures, tools, and systems software needed to enable delivery of SLAM, by supporting applications specialists in selecting and configuring the appropriate algorithm and the appropriate hardware, and compilation pathway, to meet their performance, accuracy, and energy consumption goals. [...]*

### Other Methods

The below paper from Udacity's founder Sebastian Thrun, while from 2002, is still relevant for many different methods of mapping used today in robotics.

[Robotic Mapping: A Survey](http://robots.stanford.edu/papers/thrun.mapping-tr.pdf) by S. Thrun

***Abstract:****This article provides a comprehensive introduction into the field of robotic mapping, with a focus on indoor mapping. It describes and compares various probabilistic techniques, as they are presently being applied to a vast array of mobile robot mapping problems. The history of robotic mapping is also described, along with an extensive list of open research problems.*