In the following there are some notes to my work on this little project. Just to remember what I did, when I come back to it.

Preparations:

Downloaded Repository to local machine.

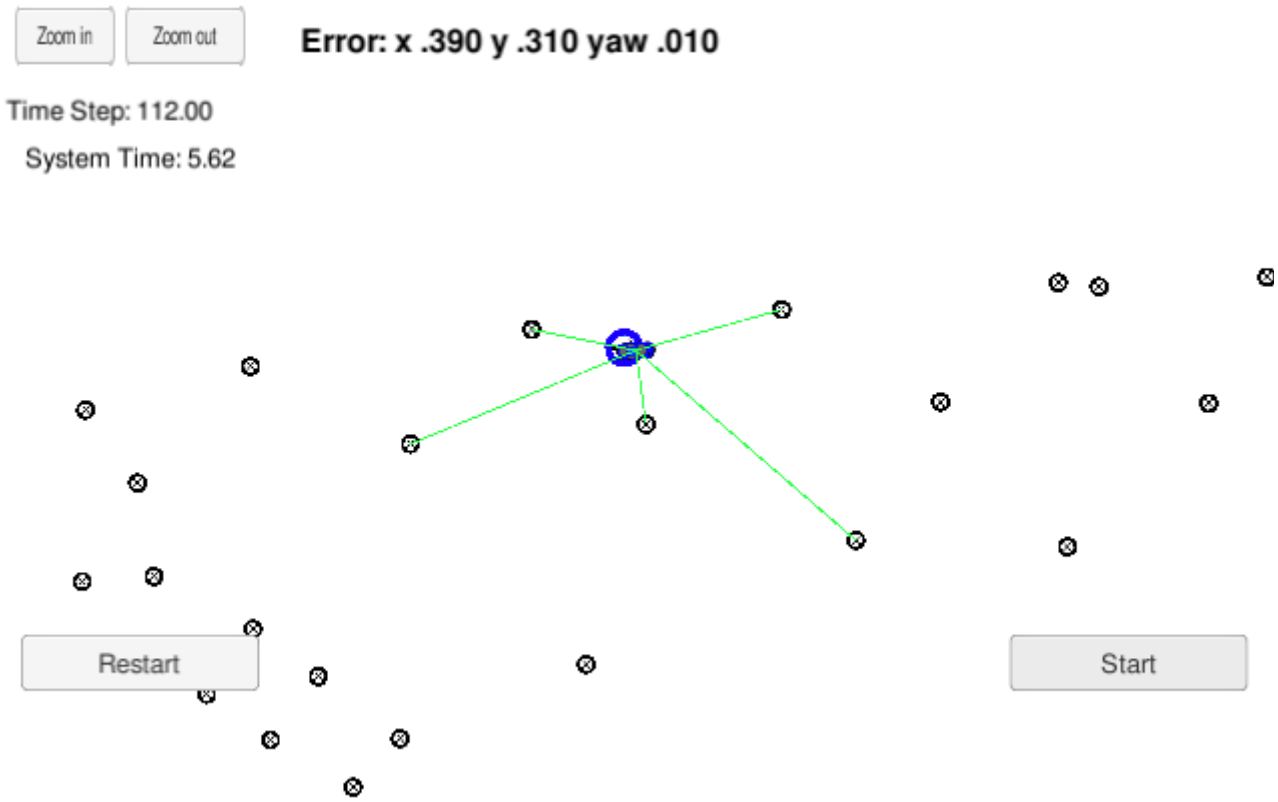
Generated Eclipse Project with cmake
make -G "Eclipse CDT4 - Unix Makefiles" -DCMAKE_BUILD_TYPE=Debug

Added c++ to compiler flags and precompiler symbols, so that eclipse recognizes C++ code. Only managed to do that in Eclipse Luna. I had this problem once already.

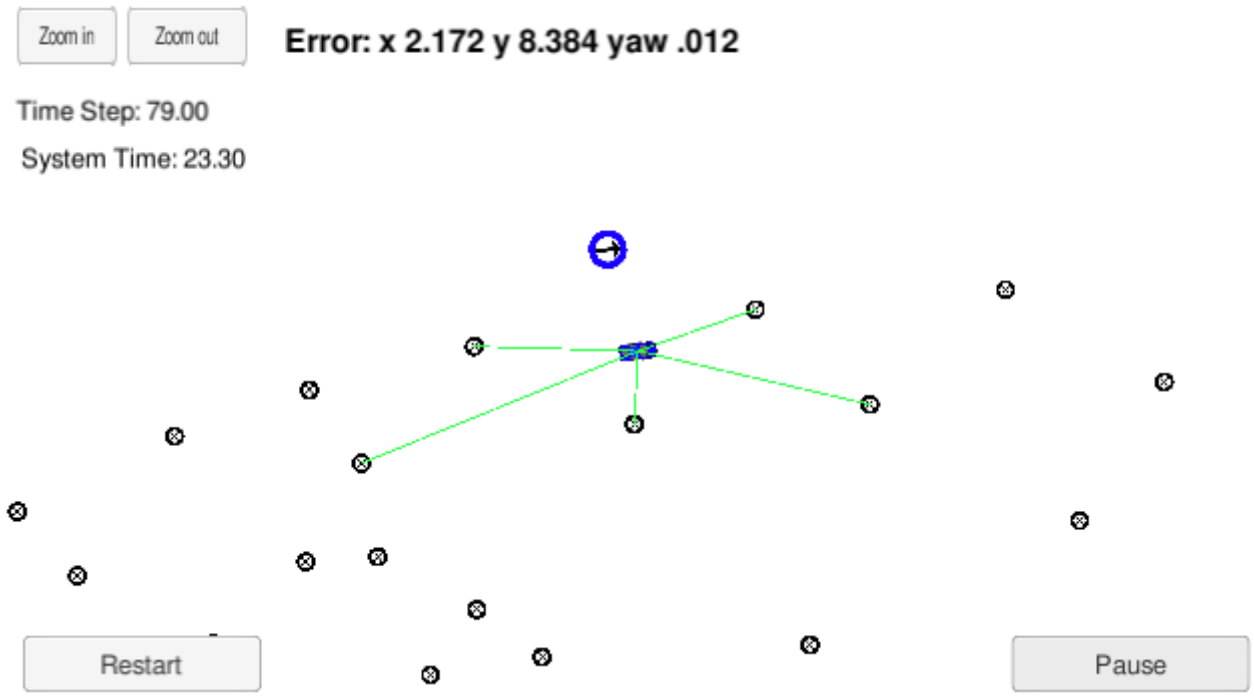
Started with one particle, to get several things right:
- Prediction of the particle – without noise and with noise
- Transformation of observations to particle
- Visualization of particle pose
- Visualization of associations

1) Prediction Process Model (using Bicycle Motion Model)

Without noise:
The cars position can be calculated pretty accurately. The error seems to be constant and due to the GPS localization initialization (also noiseless). GPS init loc != real car pos at the beginning.



With noise:
Predicted position shows a log term drift from the real car position



2) Update weights:

Next step is to transform all observations into each particle relative coordinate system.

We need to take the observations vector and apply it to each particle.
We need to find the expected observations for each particle, by finding the nearest map landmark to each pseudo observation of the particle. The error between the associated landmark positions and the pseudo observations, will define the particle importance weight using the multivariate Gaussian distribution.

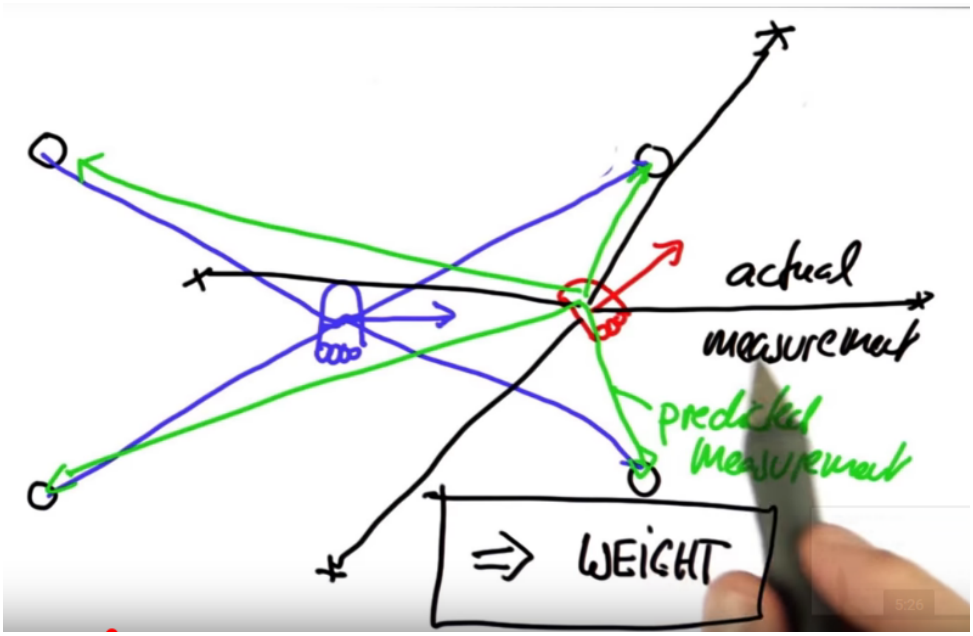
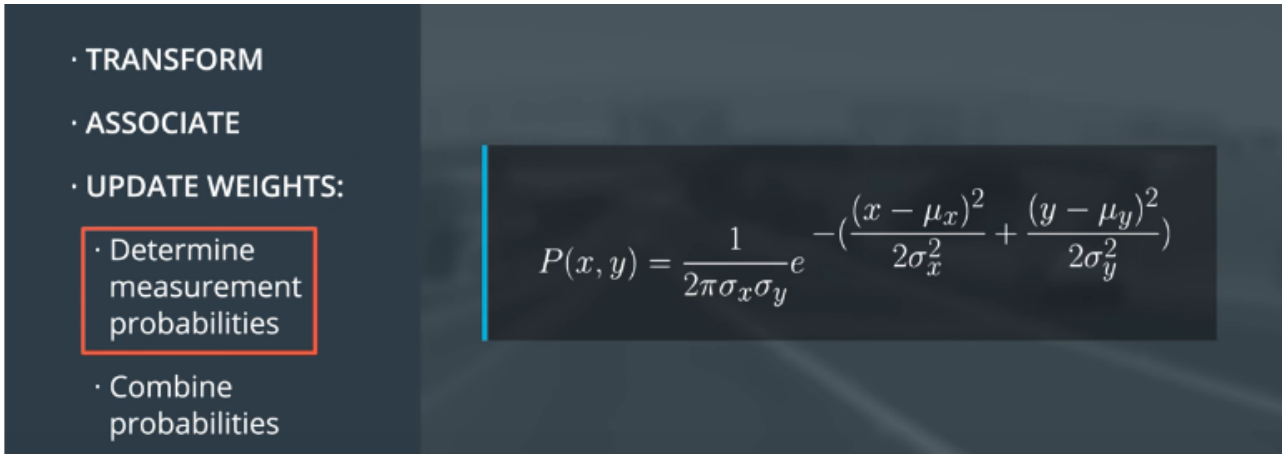


Illustration 1: SDCNP Term2 Lesson 5 Part 14 – Copyright Udacity

From Python particle filter quiz:

```
def measurement_prob(self, measurement):

        # calculates how likely a measurement should be

        prob = 1.0;
        for i in range(len(landmarks)):
            dist = sqrt((self.x - landmarks[i][0]) ** 2 + (self.y - landmarks[i][1]) ** 2)
            prob *= self.Gaussian(dist, self.sense_noise, measurement[i])
        return prob

def Gaussian(self, mu, sigma, x):

        # calculates the probability of x for 1-dim Gaussian with mean mu and var. sigma
        return exp(- ((mu - x) ** 2) / (sigma ** 2) / 2.0) / sqrt(2.0 * pi * (sigma ** 2))
```

This seems to be the same as:



Transformation of observations to particle

```
xc,yc == Observations coordinates in car coordinate system
xp,yp == particle coordinates
theta orientation of particle coordinate system
xm,ym == Observations of car coordinate system from point of view of partice coordinate system in
map coordinates

This homogeneous observation is a transformation of car coordinates into map coordinates in the maps
frame.
```

$$\begin{bmatrix} x_m \\ y_m \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & x_p \\ \sin\theta & \cos\theta & y_p \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} x_c \\ y_c \\ 1 \end{bmatrix}$$

Matrix multiplication results in:

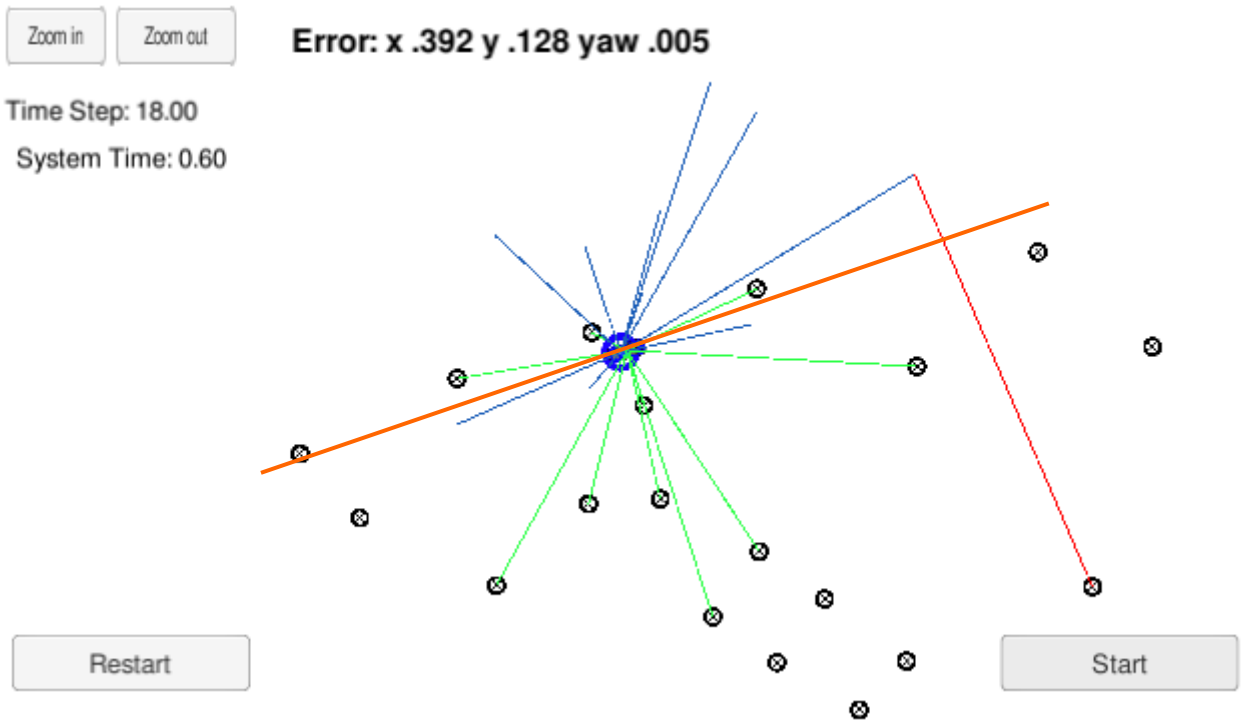
$$x_m = x_p + (\cos\theta \times x_c) - (\sin\theta \times y_c)$$

$$y_m = y_p + (\sin\theta \times x_c) + (\cos\theta \times y_c)$$

Observations are given relative to the car coordinate system (measurements in the real world from the real current car position)
Each generated particle is a pseudo car in the map world and depicts a car coordinate system
Observations of each particles car coordinate system are transformed into the map coordinate system in order to beeing able to

- Associate map landmarks with observations

- calculate distance of each landmark to the associated observation in order to determine weight of the particle



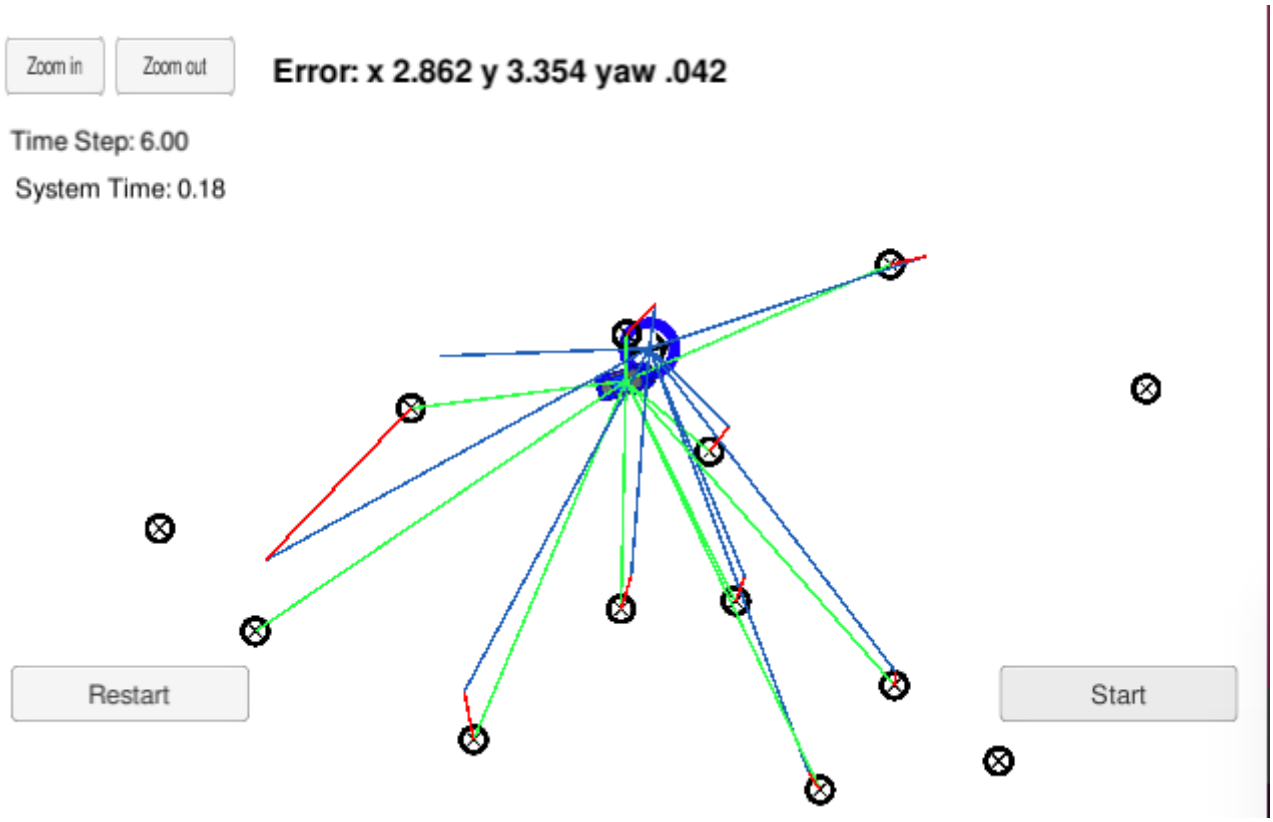
The pseudo observations seem to be mirrored along the X axis of the robot….. hmmm..
Bug found: I messed up a + as a minus in the equations:

```
xy_map.push_back(x_p + (cos(theta)*x_c) - sin(theta)*y_c);
xy_map.push_back(y_p + (sin(theta)*x_c) + cos(theta)*y_c);
```

Nearest neighbor association

As stated in the course the nearest neighbor association is not the best way to find the observation/map landmarks association, but it works. Because I just iterate over all observations and find the nearest landmark, it happens to be that not the right landmark gets associated. Following illustration shows my vehicle with a initialization offset of 5.0m to check if associations are working properly.

Better and faster way to find the correct assignments is the Hungarian Algorithm. After the generation of a cost table, it find the optimal associations according to Pair/Cost combination. The costs can be either Euclidean Distance or more sophisticated factors like Malhalanobis Distance.



4) Patricle re-sampling

This is one of the most important steps. First I had an implementation bug and my weights just collapsed to 0 after few seconds. After fixing this bug, the right particles survived. I used the resampling wheel example of the Python particle filter quiz. Just needed to find the right equivalents of the random functions in C++.

From Python particle filter quiz:

```
    for i in range(N):
        w.append(p[i].measurement_prob(Z))

    p3 = []
    index = int(random.random() * N)
    beta = 0.0
    mw = max(w)
    for i in range(N):
        beta += random.random() * 2.0 * mw
        while beta > w[index]:
            beta -= w[index]
            index = (index + 1) % N
        p3.append(p[index])
    p = p3
```

Translated to C++:

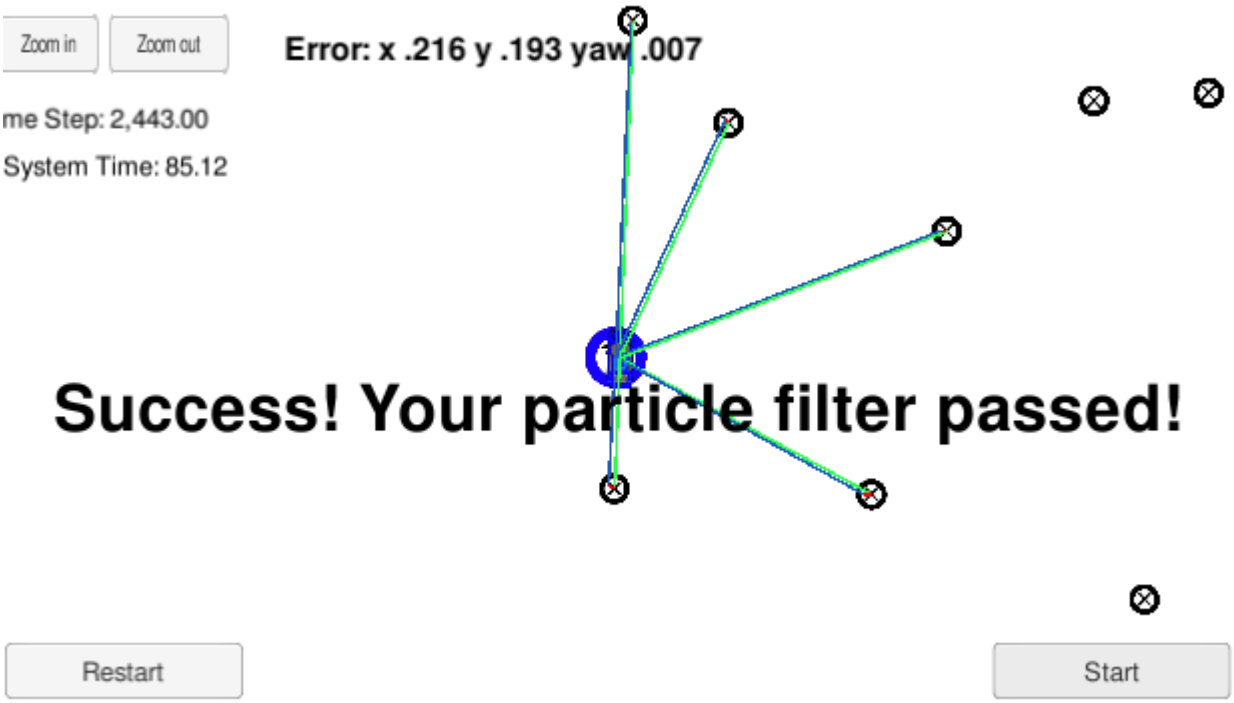
```
      for(unsigned int i=0; i<weights.size();i++)
      {

            random = static_cast <float> (rand()) / static_cast <float>
(RAND_MAX);
            //std::cout << random << std::endl;
            beta +=random * 2.0 * max_weight;

            while(beta > weights[index])
            {
               beta -= weights[index];
               index = (index + 1) % weights.size();
            }

            new_particles.push_back(particles[index]);

      }

      particles = new_particles;
```

And my bug was to iteratoe over particles[index] int the while loop, instead of the weights array. Which caused my particle weights to collapse to 0.

5) Success!

Yay! My particle filter converges and keeps track of the vehicle. Even initialized with a big offset if converges very fast to the right position.

This simple algorithm is very impressive!



Some bugs I had to track down
1) Observation vector not always as long as predicted landmarks vector. I could happen, that predicted landmarks vector is empty because all landmarks are very far away from initialized particle. If the predicted landmarks observation landmark was zero, I gave the particle a very low weight, so that it drops out in the resampling phase.

2) Yaw_rate caused zero division because I mistyped my if statement and the 0 yaw rate was not handled properly.

3) I used the wrong weights in the re sampling process, which caused my weights to collapse to 0.