Kidnapped Vehicle

REVIEW

CODE REVIEW 3

HISTORY

▾ src/particle_filter.cpp 3

```
/**
 * particle_filter.cpp
 *
 * Created on: Dec 12, 2016
 * Author: Tiffany Huang
 */

#include "particle_filter.h"

#include <math.h>
#include <algorithm>
#include <iostream>
#include <iterator>
#include <numeric>
#include <random>
#include <string>
#include <vector>

#include "helper_functions.h"

using std::string;
using std::vector;

void ParticleFilter::init(double x, double y, double theta, double std[]) {
  /**
   * TODO: Set the number of particles. Initialize all particles to
```

```
   *   first position (based on estimates of x, y, theta and their uncertainties
   *   from GPS) and all weights to 1.
   * TODO: Add random Gaussian noise to each particle.
   * NOTE: Consult particle_filter.h for more information about this method
   *   (and others in this file).
   */
  std::default_random_engine gen;
```

SUGGESTION

This is actually pseudo random, we are always generating the same particles during every run of ParticleFi

we can create a truly random generator with something like below

```
random_device rd;
default_random_engine gen(rd());
```

```

  num_particles = 50;  // TODO: Set the number of particles

  std::normal_distribution<double> dist_x(x, std[0]);
  std::normal_distribution<double> dist_y(y, std[1]);
  std::normal_distribution<double> dist_theta(theta, std[2]);

  for(int i=0; i<num_particles; i++){
    Particle p;
    p.id = i;
    // To add random Gaussian noise to the particles I use the default_random_engine
    p.x = dist_x(gen);
    p.y = dist_y(gen);
    p.theta = dist_theta(gen);
    p.weight = 1.0;
    particles.push_back(p);
    weights.push_back(p.weight);
  }

  is_initialized = true;
}

void ParticleFilter::prediction(double delta_t, double std_pos[],
                                double velocity, double yaw_rate) {
  /**
   * TODO: Add measurements to each particle and add random Gaussian noise.
   * NOTE: When adding noise you may find std::normal_distribution
   *   and std::default_random_engine useful.
   *  http://en.cppreference.com/w/cpp/numeric/random/normal_distribution
   *  http://www.cplusplus.com/reference/random/default_random_engine/
   */
  std::default_random_engine gen;

  // Generate normal distribution with mean 0 to add noise later. I think is faster t
  std::normal_distribution<double> dist_x(0, std_pos[0]);
  std::normal_distribution<double> dist_y(0, std_pos[1]);
  std::normal_distribution<double> dist_theta(0, std_pos[2]);

  for(int i=0; i<num_particles; i++){
    // If yaw_rate is 0, use different equations to predict the new state
    if(yaw_rate == 0){
```

```
      particles[i].x += velocity * delta_t * cos(particles[i].theta);
      particles[i].y += velocity * delta_t * sin(particles[i].theta);
    }else{
      particles[i].x += (velocity / yaw_rate) * (sin(particles[i].theta + yaw_rate*de
      particles[i].y += (velocity / yaw_rate) * (cos(particles[i].theta) - cos(partic
      particles[i].theta += yaw_rate * delta_t;
    }

    // Add noise
    particles[i].x += dist_x(gen);
    particles[i].y += dist_y(gen);
    particles[i].theta += dist_theta(gen);
  }

}

void ParticleFilter::dataAssociation(vector<LandmarkObs> predicted,
                                     vector<LandmarkObs>& observations) {
  /**
   * TODO: Find the predicted measurement that is closest to each
   *   observed measurement and assign the observed measurement to this
   *   particular landmark.
   * NOTE: this method will NOT be called by the grading code. But you will
   *   probably find it useful to implement this method and use it as a helper
   *   during the updateWeights phase.
   */

  // predicted and observations vectors have to be given in the map coordinates syster
  for(unsigned int o=0; o<observations.size(); o++){
    double dist_min = std::numeric_limits<double>::max();

    for(unsigned int p=0; p<predicted.size(); p++){
      // Calculate the distance between the two points
      double dist_curr = dist(predicted[p].x, predicted[p].y, observations[o].x, obser

      // How i am using maximum numeric limit and "<=" there will always be a lower d
      if(dist_curr <= dist_min){
        dist_min = dist_curr;
        observations[o].id = predicted[p].id;
      }
    }
  }

}

void ParticleFilter::updateWeights(double sensor_range, double std_landmark[],
                                   const vector<LandmarkObs> &observations,
                                   const Map &map_landmarks) {
  /**
   * TODO: Update the weights of each particle using a mult-variate Gaussian
   *   distribution. You can read more about this distribution here:
   *   https://en.wikipedia.org/wiki/Multivariate_normal_distribution
   * NOTE: The observations are given in the VEHICLE'S coordinate system.
   *   Your particles are located according to the MAP'S coordinate system.
   *   You will need to transform between the two systems. Keep in mind that
   *   this transformation requires both rotation AND translation (but no scaling).
   *   The following is a good resource for the theory:
   *   https://www.willamette.edu/~gorr/classes/GeneralGraphics/Transforms/transforms
   *   and the following is a good resource for the actual equation to implement
   *   (look at equation 3.33) http://planning.cs.uiuc.edu/node99.html
   */
```

```
  for(int i=0; i<num_particles; i++){
    /**
     *Filter all landmarks that are out of the maximum range of the sensor
     */
    // given in map coordiante system
    vector<LandmarkObs> predicted;
    for(unsigned int l=0; l<map_landmarks.landmark_list.size(); l++){
      // Only use the landmarks inside the circle, with center in the current particl
      double len = dist(particles[i].x, particles[i].y, map_landmarks.landmark_list[l
      if(len <= sensor_range){
```

SUGGESTION

What would happen if all landmarks are outside of sensor range? In such case, we should set the weight o

```
        predicted.push_back( LandmarkObs{map_landmarks.landmark_list[l].id_i, map_lan
      }
    }

    /**
     * Transform observation from car coordinate system to map coordinate system
     */
    vector <LandmarkObs> observations_map;
    for(unsigned int o=0; o<observations.size(); o++){
      double x_map = particles[i].x + cos(particles[i].theta) * observations[o].x - s
      double y_map = particles[i].y + sin(particles[i].theta) * observations[o].x + c
      observations_map.push_back( LandmarkObs{observations[o].id, x_map, y_map} );
    }

    /**
     *Associate observations with landmarks
     */
    dataAssociation(predicted, observations_map);

    /**
     *Calculate weights
     */
    particles[i].weight = 1.0;
    double x_obs, y_obs, mu_x, mu_y;
    double sig_x = std_landmark[0];
    double sig_y = std_landmark[1];
    double gauss_norm = 1 / (2 * M_PI * sig_x * sig_y);

    for(unsigned int o=0; o<observations_map.size(); o++){
      x_obs = observations_map[o].x;
      y_obs = observations_map[o].y;
      for(unsigned int p=0; p<predicted.size(); p++){
        if(observations_map[o].id == predicted[p].id){
          mu_x = predicted[p].x;
          mu_y = predicted[p].y;
          double exponent = (pow(x_obs - mu_x, 2) / (2 * pow(sig_x, 2)))
                          + (pow(y_obs - mu_y, 2) / (2 * pow(sig_y, 2)));
          particles[i].weight *= (gauss_norm * exp(-exponent));
        }
      }
    }
```

```
    weights[i] = particles[i].weight;
  }

}

void ParticleFilter::resample() {
  /**
   * TODO: Resample particles with replacement with probability proportional
   *   to their weight.
   * NOTE: You may find std::discrete_distribution helpful here.
   *   http://en.cppreference.com/w/cpp/numeric/random/discrete_distribution
   */
  std::default_random_engine gen;
  vector<Particle> new_particles;

  // generate random index to begin the resampling
  std::uniform_int_distribution<int> dist_index(0, num_particles-1);
  int index = dist_index(gen);

  double beta = 0.0;

  // calculate the maxium weight of all particles
  double mw = *max_element(weights.begin(), weights.end());

  // generate a real uniformon distribution
  std::uniform_real_distribution<double> dist_beta(0.0, 1.0);

```

SUGGESTION

An alternative approach is to use std::discrete_distribution, http://en.cppreference.com/w/cpp/numeric/ra

```
std::random_device seed;
std::mt19937 random_generator(seed());
// sample particles based on their weight
std::discrete_distribution<> sample(weights.begin(), weights.end());

std::vector<Particle> new_particles(num_particles);
for(auto & p : new_particles)
p = particles[sample(random_generator)];
particles = std::move(new_particles);
```

```
  for(int i=0; i<num_particles; i++){
    beta += dist_beta(gen) * 2.0 * mw;
    while(beta > weights[index]){
      beta -= weights[index];
      index = (index + 1) % num_particles;
    }
    new_particles.push_back(particles[index]);
  }

  particles = new_particles;
}

void ParticleFilter::SetAssociations(Particle& particle,
                                     const vector<int>& associations,
                                     const vector<double>& sense_x,
                                     const vector<double>& sense_y) {
```

```
  // particle: the particle to which assign each listed association,
  //   and association's (x,y) world coordinates mapping
  // associations: The landmark id that goes along with each listed association
  // sense_x: the associations x mapping already converted to world coordinates
  // sense_y: the associations y mapping already converted to world coordinates
  particle.associations= associations;
  particle.sense_x = sense_x;
  particle.sense_y = sense_y;
}

string ParticleFilter::getAssociations(Particle best) {
  vector<int> v = best.associations;
  std::stringstream ss;
  copy(v.begin(), v.end(), std::ostream_iterator<int>(ss, " "));
  string s = ss.str();
  s = s.substr(0, s.length()-1);  // get rid of the trailing space
  return s;
}

string ParticleFilter::getSenseCoord(Particle best, string coord) {
  vector<double> v;

  if (coord == "X") {
    v = best.sense_x;
  } else {
    v = best.sense_y;
  }

  std::stringstream ss;
  copy(v.begin(), v.end(), std::ostream_iterator<float>(ss, " "));
  string s = ss.str();
  s = s.substr(0, s.length()-1);  // get rid of the trailing space
  return s;
}
```

▸ src/particle_filter.h

▸ src/map.h

▸ src/main.cpp

▸ src/helper_functions.h

▸ output/Makefile

▸ output/CMakeFiles/particle_filter.dir/link.txt

▸ output/CMakeFiles/TargetDirectories.txt

▸ output/CMakeFiles/3.5.1/CompilerIdCXX/CMakeCXXCompilerId.cpp

- output/CMakeCache.txt
- data/map_data.txt
- cmakepatch.txt
- README.md
- CMakeLists.txt

RETURN TO PATH

Rate this review