Marouen Azzouz

SFND5- Unscented Kalman Filter

The submission

The code compiles without exceeding threshold on my ubuntu18 machine. Below some notes which explain the steps taken to implement UKF

- UKF is called from
  - **Highway.h** :: `tools.radarSense(traffic[i], egoCar, viewer, timestamp, visualize_radar);`
  - **Highway.h** :: `tools.lidarSense(traffic[i], viewer, timestamp, visualize_lidar);`

"`tools.radarSense`" and "`tools.lidarSense`" functions call ukf.cpp implementation.

- UKF blocks are as follows

1. Initialization: ***ukf.cpp line 81***

```
if(!is_initialized_){
  //use first reading as initialization
  if (meas_package.sensor_type_ == MeasurementPackage::RADAR) {
    double rho = meas_package.raw_measurements_[0];
    double phi = meas_package.raw_measurements_[1];
    double rho_dot = meas_package.raw_measurements_[2];
    double x = rho * cos(phi);
    double y = rho * sin(phi);
    double vx = rho_dot * cos(phi);
    double vy = rho_dot * sin(phi);
    double v = sqrt(vx * vx + vy * vy);
    if(cos(phi)<0) v*=-1;
    x_ << x, y, 0, 0, 0;
  }
  if (meas_package.sensor_type_ == MeasurementPackage::LASER)  {
    double x = meas_package.raw_measurements_[0];
    double y = meas_package.raw_measurements_[1];
    x_ << x, y, 0, 0, 0;
  }

  P_ <<
      1.0, 0.0, 0.0, 0.0, 0.0,
      0.0, 1.0, 0.0, 0.0, 0.0,
      0.0, 0.0, 1.0, 0.0, 0.0,
      0.0, 0.0, 0.0, 1.0, 0.0,
      0.0, 0.0, 0.0, 0.0, 1.0;
  // initalize time_us
```

```
   time_us_ = meas_package.timestamp_ ;

   is_initialized_ = true;
   return;
 }
```

2. Motion model ***ukf.cpp line 129***
    - Generate sigma points ***ukf.cpp line 147***
    - Sigma points prediction ***ukf.cpp line 177***
    - Predict mean and covariance ***ukf.cpp line 223***

```
void UKF::PredictMotion(MeasurementPackage meas_package)
{
 /**
  * TODO: Complete this function! Estimate the object's location.
  * Modify the state vector, x_. Predict sigma points, the state,
  * and the state covariance matrix.
  */


 Xsig_aug_ = MatrixXd(n_aug_, 2 * n_aug_ + 1);
 Xsig_pred_ = MatrixXd(n_x_, 2 * n_aug_ + 1);
 weights_ = VectorXd(2*n_aug_+1);

 GenerateSigmaPoints();
 SigmaPointPrediction(dt_);
 PredictMeanAndCovariance();
}
```

3. UKF update if radar sensor
    - Predict radar measurement **ukf.cpp line348**
    - Compute a posteriori (update) **ukf.cpp line 408**

```
void UKF::ProcessRadarMeasurement(MeasurementPackage meas_package)
{
 n_z_= 3;
 S_ = MatrixXd(n_z_,n_z_);
 z_pred_ = VectorXd(n_z_);
 Zsig_ = MatrixXd(n_z_, 2 * n_aug_ + 1);
 R_ = MatrixXd(n_z_,n_z_);
```

```
 R_ <<
   std_radr_*std_radr_, 0, 0,
   0, std_radphi_*std_radphi_, 0,
   0, 0,std_radrd_*std_radrd_;

 PredictRadarMeasurement();
 UpdateRadar(meas_package);

}
```

3. UKF Update if Laser
    - Predict Lidar measurement ***ukf.cpp line 316***
    - Update lidar measurement ***ukf.cpp line 285***

```
void UKF::ProcessLidarMeasurement(MeasurementPackage meas_package)
{
 n_z_= 2;
 S_ = MatrixXd(n_z_,n_z_);
 z_pred_ = VectorXd(n_z_);
 Zsig_ = MatrixXd(n_z_, 2 * n_aug_ + 1);
 H_ = MatrixXd(n_z_,n_x_);
 R_ = MatrixXd(n_z_,n_z_);

 H_ <<
   1.0, 0.0, 0.0, 0.0, 0.0,
   0.0, 1.0, 0.0, 0.0, 0.0;

 R_ <<
   std_laspx_*std_laspx_, 0.0,
   0.0, std_laspy_*std_laspy_;

 PredictLidarMeasurement();
 UpdateLidar(meas_package);
 //UpdateRadar(meas_package);

}
```