UDACITY

Return to Classroom

Extended Kalman Filters

REVIEW

CODE REVIEW 2

HISTORY

▸ CarND-Extended-Kalman-Filter-Project/src/kalman_filter.cpp 1

▾ CarND-Extended-Kalman-Filter-Project/src/FusionEKF.cpp 1

```
#include "FusionEKF.h"
#include <iostream>
#include "Eigen/Dense"
#include "tools.h"

using Eigen::MatrixXd;
using Eigen::VectorXd;
using std::cout;
using std::endl;
using std::vector;

/**
 * Constructor.
 */
FusionEKF::FusionEKF() {
  is_initialized_ = false;

  previous_timestamp_ = 0;

  // initializing matrices
  R_laser_ = MatrixXd(2, 2);
  R_radar_ = MatrixXd(3, 3);
  H_laser_ = MatrixXd(2, 4);
  Hj_ = MatrixXd(3, 4);

  //measurement covariance matrix - laser
  R_laser_ << 0.0225, 0,
              0, 0.0225;

  //measurement covariance matrix - radar
  R_radar_ << 0.09, 0, 0,
              0, 0.0009, 0,
              0, 0, 0.09;

  /**
   * TODO: Finish initializing the FusionEKF.
   * TODO: Set the process and measurement noises
   */
  // measurement function
  H_laser_ << 1, 0, 0, 0,
  0, 1, 0, 0;

  //state covariance
  MatrixXd P_ = MatrixXd(4, 4);
```

SUGGESTION

The P matrix is the corresponding covariance matrix. Its diagonal elements tell us how certain we are in the state variables (means).
When we initialize P, we would like to describe our certainty on the state variables. We are relatively certain in the x and y values, because both measurement types give us some relatively accurate value (so we can set 1 for the first two diagonal elements). Because the first measurement can be RADAR measurement and in case of radar measurement we do not really know the exact velocity vector (vx,vy), we can set a higher value (like 100 or 1000) for the 3rd and 4th diagonal elements.
However, it is often used that we set P as the identity matrix, and we hope that the Kalman Filter will converge relatively fast to the correct values.

```
  P_ << 1, 0, 0, 0,
  0, 1, 0, 0,
  0, 0, 10000, 0,
  0, 0, 0, 10000;

  //next state function
  MatrixXd F_ = MatrixXd(4, 4);
  F_ << 1, 0, 1, 0,
  0, 1, 0, 1,
  0, 0, 1, 0,
  0, 0, 0, 1;

  //process covariance
  MatrixXd Q_ = MatrixXd(4, 4);
  Q_ << 1, 0, 1, 0,
  0, 1, 0, 1,
  1, 0, 1, 0,
  0, 1, 0, 1;

  //initial state
  VectorXd x_ = VectorXd(4);
  x_ << 1, 1, 1, 1;
  ekf_.Init(x_, P_, F_, H_laser_, R_laser_, Q_);

}

/**
 * Destructor.
 */
FusionEKF::~FusionEKF() {}

void FusionEKF::ProcessMeasurement(const MeasurementPackage &measurement_pack) {
  /**
   * Initialization
   */
  if (!is_initialized_) {
    /**
     * TODO: Initialize the state ekf_.x_ with the first measurement.
     * TODO: Create the covariance matrix.
     * You'll need to convert radar from polar to cartesian coordinates.
     */

    //first measurement
    cout << "EKF: " << endl;
    ekf_.x_ << 1, 1, 1, 1;

    if (measurement_pack.sensor_type_ == MeasurementPackage::RADAR) {
      // TODO: Convert radar from polar to cartesian coordinates
      //         and initialize state.
      cout << "First measurement - RADAR" << endl;
      double rho = measurement_pack.raw_measurements_[0]; //range
      double phi = measurement_pack.raw_measurements_[1]; //bearing
      double rho_dot = measurement_pack.raw_measurements_[2]; //range rate
      //polar to cartesian
      double x = rho * cos(phi);
      if ( x < 0.0001 ) {
        x = 0.0001;
      }
      double y = rho * sin(phi);
      if ( y < 0.0001 ) {
        y = 0.0001;
      }
      double vx = rho_dot * cos(phi);
      double vy = rho_dot * sin(phi);
      ekf_.x_ << x, y, vx , vy;

    }
    else if (measurement_pack.sensor_type_ == MeasurementPackage::LASER) {
      // TODO: Initialize state.
```

```
      cout << "First measurement - LIDAR" << endl;
      ekf_.x_ << measurement_pack.raw_measurements_[0], measurement_pack.raw_measurements_[1], 0, 0;
    }
    previous_timestamp_ = measurement_pack.timestamp_ ;
    // done initializing, no need to predict or update
    is_initialized_ = true;
    return;
  }

  /**
   * Prediction
   */

  /**
   * TODO: Update the state transition matrix F according to the new elapsed time.
   * Time is measured in seconds.
   * TODO: Update the process noise covariance matrix.
   * Use noise_ax = 9 and noise_ay = 9 for your Q matrix.
   */
  double noise_ax = 9.0;
  double noise_ay = 9.0;

  double deltat = (measurement_pack.timestamp_ - previous_timestamp_) / 1000000.0;
  previous_timestamp_ = measurement_pack.timestamp_;

  //next state update
  ekf_.F_ << 1, 0, deltat, 0,
             0, 1, 0, deltat,
             0, 0, 1, 0,
             0, 0, 0, 1;

  double delta_t_2 = deltat * deltat;
  double quater_delta_t_4 = delta_t_2 * delta_t_2 / 4;
  double half_delta_t_3 = delta_t_2 * deltat / 2;
  ekf_.Q_ = MatrixXd(4, 4);
  ekf_.Q_ << quater_delta_t_4 * noise_ax, 0, half_delta_t_3 * noise_ax, 0,
             0, quater_delta_t_4 * noise_ay, 0, half_delta_t_3 * noise_ay,
             half_delta_t_3 * noise_ax, 0, delta_t_2 * noise_ax, 0,
             0, half_delta_t_3 * noise_ay, 0, delta_t_2 * noise_ay;


  ekf_.Predict();

  /**
   * Update
   */

  /**
   * TODO:
   * - Use the sensor type to perform the update step.
   * - Update the state and covariance matrices.
   */

  if (measurement_pack.sensor_type_ == MeasurementPackage::RADAR) {
    // TODO: Radar updates
    ekf_.H_ = tools.CalculateJacobian(ekf_.x_);
    ekf_.R_ = R_radar_;
    ekf_.UpdateEKF(measurement_pack.raw_measurements_);
  } else {
    // TODO: Laser updates
    ekf_.H_ = H_laser_;
    ekf_.R_ = R_laser_;
    ekf_.Update(measurement_pack.raw_measurements_);
  }

  // print the output
  cout << "x_ = " << ekf_.x_ << endl;
  cout << "P_ = " << ekf_.P_ << endl;
}
```

▸ **CarND-Extended-Kalman-Filter-Project/src/Eigen/src/Core/arch/SSE/CMakeLists.txt**

▸ **CarND-Extended-Kalman-Filter-Project/src/Eigen/src/Core/arch/NEON/CMakeLists.txt**

- CarND-Extended-Kalman-Filter-Project/src/Eigen/src/Core/arch/Default/CMakeLists.txt
- CarND-Extended-Kalman-Filter-Project/src/Eigen/src/Core/arch/AltiVec/CMakeLists.txt
- CarND-Extended-Kalman-Filter-Project/src/Eigen/src/LU/arch/CMakeLists.txt
- CarND-Extended-Kalman-Filter-Project/src/Eigen/src/Geometry/arch/CMakeLists.txt
- CarND-Extended-Kalman-Filter-Project/src/Eigen/src/Eigen2Support/Geometry/CMakeLists.txt
- CarND-Extended-Kalman-Filter-Project/src/Eigen/src/Core/util/CMakeLists.txt
- CarND-Extended-Kalman-Filter-Project/src/Eigen/src/Core/products/CMakeLists.txt
- CarND-Extended-Kalman-Filter-Project/src/Eigen/src/Core/arch/CMakeLists.txt
- CarND-Extended-Kalman-Filter-Project/src/Eigen/src/plugins/CMakeLists.txt
- CarND-Extended-Kalman-Filter-Project/src/Eigen/src/misc/CMakeLists.txt
- CarND-Extended-Kalman-Filter-Project/src/Eigen/src/UmfPackSupport/CMakeLists.txt
- CarND-Extended-Kalman-Filter-Project/src/Eigen/src/SuperLUSupport/CMakeLists.txt
- CarND-Extended-Kalman-Filter-Project/src/Eigen/src/StlSupport/CMakeLists.txt
- CarND-Extended-Kalman-Filter-Project/src/Eigen/src/SparseQR/CMakeLists.txt
- CarND-Extended-Kalman-Filter-Project/src/Eigen/src/SparseLU/CMakeLists.txt
- CarND-Extended-Kalman-Filter-Project/src/Eigen/src/SparseCore/CMakeLists.txt
- CarND-Extended-Kalman-Filter-Project/src/Eigen/src/SparseCholesky/CMakeLists.txt
- CarND-Extended-Kalman-Filter-Project/src/Eigen/src/SVD/CMakeLists.txt
- CarND-Extended-Kalman-Filter-Project/src/Eigen/src/SPQRSupport/CMakeLists.txt
- CarND-Extended-Kalman-Filter-Project/src/Eigen/src/QR/CMakeLists.txt
- CarND-Extended-Kalman-Filter-Project/src/Eigen/src/PardisoSupport/CMakeLists.txt
- CarND-Extended-Kalman-Filter-Project/src/Eigen/src/PaStiXSupport/CMakeLists.txt
- CarND-Extended-Kalman-Filter-Project/src/Eigen/src/OrderingMethods/CMakeLists.txt
- CarND-Extended-Kalman-Filter-Project/src/Eigen/src/MetisSupport/CMakeLists.txt
- CarND-Extended-Kalman-Filter-Project/src/Eigen/src/LU/CMakeLists.txt
- CarND-Extended-Kalman-Filter-Project/src/Eigen/src/Jacobi/CMakeLists.txt
- CarND-Extended-Kalman-Filter-Project/src/Eigen/src/IterativeLinearSolvers/CMakeLists.txt

- CarND-Extended-Kalman-Filter-Project/src/Eigen/src/Householder/CMakeLists.txt
- CarND-Extended-Kalman-Filter-Project/src/Eigen/src/Geometry/CMakeLists.txt
- CarND-Extended-Kalman-Filter-Project/src/Eigen/src/Eigenvalues/CMakeLists.txt
- CarND-Extended-Kalman-Filter-Project/src/Eigen/src/Eigen2Support/CMakeLists.txt
- CarND-Extended-Kalman-Filter-Project/src/Eigen/src/Core/CMakeLists.txt
- CarND-Extended-Kalman-Filter-Project/src/Eigen/src/CholmodSupport/CMakeLists.txt
- CarND-Extended-Kalman-Filter-Project/src/Eigen/src/Cholesky/CMakeLists.txt
- CarND-Extended-Kalman-Filter-Project/src/CMakeFiles/3.13.0-rc2/CompilerIdCXX/CMakeCXXCompilerId.cpp
- CarND-Extended-Kalman-Filter-Project/src/Eigen/src/CMakeLists.txt
- CarND-Extended-Kalman-Filter-Project/src/CMakeFiles/ExtendedKF.dir/link.txt
- CarND-Extended-Kalman-Filter-Project/src/Eigen/CMakeLists.txt
- CarND-Extended-Kalman-Filter-Project/src/CMakeFiles/TargetDirectories.txt
- CarND-Extended-Kalman-Filter-Project/ide_profiles/xcode/README.md
- CarND-Extended-Kalman-Filter-Project/ide_profiles/Eclipse/README.md
- CarND-Extended-Kalman-Filter-Project/src/tools.cpp
- CarND-Extended-Kalman-Filter-Project/src/main.cpp
- CarND-Extended-Kalman-Filter-Project/src/Makefile
- CarND-Extended-Kalman-Filter-Project/src/CMakeCache.txt
- CarND-Extended-Kalman-Filter-Project/ide_profiles/README.md
- CarND-Extended-Kalman-Filter-Project/data/obj_pose-laser-radar-synthetic-input.txt
- CarND-Extended-Kalman-Filter-Project/Docs/Input_Output File Format.txt
- CarND-Extended-Kalman-Filter-Project/Docs/Data_Flow_Doc.txt
- CarND-Extended-Kalman-Filter-Project/cmakepatch.txt
- CarND-Extended-Kalman-Filter-Project/README.md
- CarND-Extended-Kalman-Filter-Project/CMakeLists.txt

Rate this review

START