UDACITY

Logout

PROJECT

Unscented Kalman Filters

A part of the Self Driving Car Engineer Nanodegree Program

PROJECT REVIEW | CODE REVIEW 9 | NOTES

src/ukf.cpp 9

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

using namespace std;
using Eigen::MatrixXd;
using Eigen::VectorXd;
using std::vector;

/**
 * Initializes Unscented Kalman filter
 */
UKF::UKF() {
```

REQUIRED

You need to initialize the member variable `is_initialized_ = false`. In C++ you are responsible for initia variables to sensible values. Otherwise, a declared variable will contain whatever was stored at its assigned m before. In case of a `bool`, any value other than zero will be interpreted as `true`. Thus, your filter is never in may be obfuscated for certain combinations of compiler and build configuration, making it especially easy to development.

```
  // if this is false, laser measurements will be ignored (except during init)
  use_laser_ = true;

  // if this is false, radar measurements will be ignored (except during init)
  use_radar_ = true;

  // initial state vector
  x_ = VectorXd(5);

  // initial covariance matrix
  P_ = MatrixXd(5, 5);

  // Process noise standard deviation longitudinal acceleration in m/s^2
  std_a_ = 4;//30; //experiment with process noise.. to get close accuracy level

  // Process noise standard deviation yaw acceleration in rad/s^2
  std_yawdd_ = 1;//30; // experiment with noise .. keeping it minimum

  // Laser measurement noise standard deviation position1 in m
  std_laspx_ = 0.15;

  // Laser measurement noise standard deviation position2 in m
  std_laspy_ = 0.15;

  // Radar measurement noise standard deviation radius in m
  std_radr_ = 0.3;

  // Radar measurement noise standard deviation angle in rad
  std_radphi_ = 0.03;

  // Radar measurement noise standard deviation radius change in m/s
  std_radrd_ = 0.3;

  /**
  TODO:

  Complete the initialization. See ukf.h for other member properties.
```

```
  Complete the initialization. See ukf.h for other member properties.

  Hint: one or more values initialized above might be wildly off...
  */
    //just init with 1
    P_ << 1,0,0,0,0,
            0,1,0,0,0,
            0,0,1,0,0,
            0,0,0,1,0,
            0,0,0,0,1;

    n_x_ = 5;
    n_aug_ = n_x_ + 2;

    lambda_ = 3-n_aug_;

    //holding predicted sigma points, 5 x 15 dimention matrix, including augmente
    Xsig_pred_ = MatrixXd(n_x_, 2* n_aug_ +1);


}

UKF::~UKF() {}


// predicting state with dt
VectorXd predictX(VectorXd x, double dt,int n_x_){



    double px =  x(0);
    double py =  x(1);
    double  v =  x(2);
    double psi = x(3);
    double dpsi =x(4);
    double va =  x(5);
    double vp =  x(6);
```


SUGGESTION

It is a good idea to define constants that will not or should not change in the current scope as `const`. This helps to prevent bugs and will help the compiler to optimize the code during compilation.

```

    double dt2 = dt*dt;

    VectorXd x_pl = VectorXd(n_x_);
    VectorXd fx = VectorXd(n_x_);
    fx.setZero();

    //to avoid divide by zero
    if (fabs(dpsi)<0.001){

        fx << v*cos(psi)*dt + 1.0/2.0*dt2*cos(psi)*va,
                v*sin(psi)*dt + 1.0/2.0*dt2*sin(psi)*va,
                dt*va,
                1/2*dt2*vp,
                dt*vp;

    }else{
        fx << v/dpsi*(sin(psi+dpsi*dt)-sin(psi)) + 1.0/2.0*dt2*cos(psi)*va,
                v/dpsi*(-cos(psi+dpsi*dt)+cos(psi)) + 1.0/2.0*dt2*sin(psi)*va,
                dt*va,
                dpsi*dt+1/2*dt2*vp,
                dt*vp;

    }
    x_pl = x.head(n_x_) + fx;
    return x_pl;
}


// Radar measurement
VectorXd getRadarMeasurement(const VectorXd x){
    double  px = x(0);
    double  py = x(1);
    double   v = x(2);
    double  psi = x(3);
    double dpsi = x(4);
    VectorXd Z = VectorXd(3);

    if (sqrt(pow(px*px + py*py,2))>0.001){

        Z(0) = sqrt(px*px+py*py);
        Z(1) = atan2(py,px);

        Z(2) = (px*v*cos(psi) + py*v*sin(psi))/Z(0);

        cout<<"the z for radar  is "<<Z<<endl;
```

```
        return Z;
    } else {
        Z(0) = sqrt(0.001);
        Z(1) = atan2(0.001,0.001);
        Z(2) = (px*v*cos(psi) + py*v*sin(psi))/Z(0);
        return Z;
```

AWESOME

You avoid both division by zero and undefined `atan(0.0,0.0)`!

```
    }
}

// LIDAR measurement
VectorXd getLidarMeasurement(const VectorXd x){
    double  px = x(0);
    double  py = x(1);
    VectorXd Z = VectorXd(2);
    Z(0) = px;
    Z(1) = py;

    cout<<"the z for lidar  is "<<Z<<endl;

    return Z;
}




/**
 * @param {MeasurementPackage} meas_package The latest measurement data of
 * either radar or laser.
 */
void UKF::ProcessMeasurement(MeasurementPackage meas_package) {
  /**
  TODO:

  Complete this function! Make sure you switch between lidar and radar
  measurements.
  */

    if (!is_initialized_) {

        // first measurement
        //cout << "UKF: " << endl;
        if (meas_package.sensor_type_ == MeasurementPackage::RADAR) {
            /**
            Convert radar from polar to cartesian coordinates and initialize stat
            */
            float rho = meas_package.raw_measurements_(0);
            float psi =  meas_package.raw_measurements_(1);
            float drho =  meas_package.raw_measurements_(2);

            x_(0) = rho*cos(psi);
            x_(1) = rho*sin(psi);
            x_(2) = drho*cos(psi);
            x_(3) = drho*sin(psi);
```

REQUIRED

The components of the state vector x_ are: `[px, py, v, yaw, yaw_d]`. So you set `v = drho*cos(psi)` ar `yaw=drho*sin(psi)`, which is not correct. You could use `abs(drho)` as an initial estimate for the velocity, c would set `yaw = 0`.

```
            x_(4) = 0; //init to zero
            if (fabs(rho)>0.001){
                is_initialized_ = true;
            }
        }
        else if (meas_package.sensor_type_ == MeasurementPackage::LASER) {
            /**
            Initialize state.
            */
            x_(0) =meas_package.raw_measurements_(0);
            x_(1) =meas_package.raw_measurements_(1);
            x_(2) = 0; // Approximate value of 0
            x_(3) = 0; // Approximate value of 0
            x_(4) = 0; // Approximate value of 0

            if (sqrt(pow(x_(0),2)+pow(x_(1),2))>0.001){
                is_initialized_ = true;
            }

        }

```

```
        // done initializing, no need to predict or update

        previous_timestamp_ = meas_package.timestamp_;
        return;

    }

    //compute the time elapsed between the current and previous measurements
    float dt = (meas_package.timestamp_ - previous_timestamp_) / 1000000.0; //dt
    previous_timestamp_ = meas_package.timestamp_;




    //Use small dt to allow for turn effect
    const double diff_t = 0.1;

    while (dt > diff_t){
        Prediction(diff_t);
        dt -= diff_t;
    }
```

SUGGESTION

This empirical trick seems to improve numerical stability for the second older dataset which has a different u
delta_t = 1.0s) than the first older one (ca. delta_t = 0.05s). In a real implementation, we would only have to de
against a fixed and defined update rate (e.g. delta_t = 0.05s), so this subdividing of the prediction steps would

```

    if ( dt > 0.001 ) {
        Prediction(dt); // update states only if dt is above 0.001
    }



    cout<< "good till here"<<x_<<endl;

    if (meas_package.sensor_type_ == MeasurementPackage::RADAR) {

        if (fabs(meas_package.raw_measurements_(0))>0.001){
            UpdateRadar(meas_package);
        }

    } else if (meas_package.sensor_type_ == MeasurementPackage::LASER){
        double l_px = meas_package.raw_measurements_(0);
        double l_py = meas_package.raw_measurements_(1);
        if (sqrt(pow(l_px,2)+pow(l_py,2))>0.001){
            UpdateLidar(meas_package);
        }
    }

}

/**
 * Predicts sigma points, the state, and the state covariance matrix.
 * @param {double} delta_t the change in time (in seconds) between the last
 * measurement and this one.
 */
void UKF::Prediction(double delta_t) {
  /**
  TODO:

  Complete this function! Estimate the object's location. Modify the state
  vector, x_. Predict sigma points, the state, and the state covariance matrix.
  */

    // Generate sigma points
    //initialization of matrices for sigma point calculations
    MatrixXd Xsig_aug = MatrixXd(n_aug_, 2 * n_aug_ + 1);
    VectorXd x_aug = VectorXd(n_aug_);
    MatrixXd P_aug = MatrixXd(n_aug_,n_aug_);
    MatrixXd A_aug = MatrixXd(n_aug_,n_aug_);
    MatrixXd Ones_nAug = MatrixXd(1, n_aug_);
    Ones_nAug.setOnes();
    MatrixXd Ones_nA = MatrixXd(1, 2*n_aug_+1);
    Ones_nA.setOnes();
    //calculate AUGMENTED points ...
    P_aug.setZero();
    P_aug.topLeftCorner( n_x_, n_x_) = P_;
    P_aug(5,5) = std_a_*std_a_; // 0 based indexing
    P_aug(6,6) = std_yawdd_*std_yawdd_; // 0 based indexing

    A_aug = P_aug.llt().matrixL();
    x_aug.setZero();
    x_aug.head(n_x_) << x_;

    // augmented sigma points

    Xsig_aug << x_aug,
```

```
            x_aug*Ones_nAug+sqrt(lambda_+n_aug_)*A_aug,
            x_aug*Ones_nAug-sqrt(lambda_+n_aug_)*A_aug; // generate sigma pts
```

SUGGESTION

You could precompute `sqrt(lambda_ + n_aug_)` before the loop in order avoid some redundant computat

```




    //create matrix with predicted sigma points as columns
    MatrixXd Xsig_pred = MatrixXd(n_x_, 2 * n_aug_ + 1);
    Xsig_pred.col(0) = predictX(Xsig_aug.col(0),delta_t,n_x_);
    for (int i=1;i<=n_aug_;i++){
        Xsig_pred.col(i) = predictX(Xsig_aug.col(i),delta_t,n_x_);
        Xsig_pred.col(i+n_aug_) = predictX(Xsig_aug.col(i+n_aug_),delta_t,n_x_);
    }

    VectorXd weights = VectorXd(2*n_aug_+1);
    MatrixXd Wts_diag = MatrixXd(2*n_aug_+1,2*n_aug_+1);
    //set weights
    weights(0) = lambda_/(lambda_+n_aug_);
    weights.tail(2*n_aug_).fill(1/2./(lambda_+n_aug_));
```

SUGGESTION

The weights do not change between iterations, so you could precompute them in the constructor.

```
    Wts_diag  = MatrixXd(weights.asDiagonal());
    //predict state mean
    x_ = Xsig_pred*weights;
    // predict covariance
    P_ =(Xsig_pred-x_*Ones_nA)*Wts_diag*(Xsig_pred-x_*Ones_nA).transpose();
    Xsig_pred_ = Xsig_pred;


}

/**
 * Updates the state and the state covariance matrix using a laser measurement.
 * @param {MeasurementPackage} meas_package
 */
void UKF::UpdateLidar(MeasurementPackage meas_package) {
```

SUGGESTION

The measurement model in the LIDAR case is linear. That means that we do not need to use the unscented tr
all! Using the linear Kalman filter update like in the EKF project should yield the same results with less comput

```
  /**
  TODO:

  Complete this function! Use lidar data to update the belief about the object's
  position. Modify the state vector, x_, and covariance, P_.

  You'll also need to calculate the lidar NIS.
  */

    int n_z_ = 2;

    VectorXd weights = VectorXd(2*n_aug_+1);
    MatrixXd Wts_diag = MatrixXd(2*n_aug_+1,2*n_aug_+1);
    MatrixXd Zsig = MatrixXd(n_z_, 2 * n_aug_ + 1);
    MatrixXd S = MatrixXd(n_z_,n_z_);
    VectorXd z_pred = VectorXd(n_z_);
    VectorXd z_true = VectorXd(n_z_);
    MatrixXd Ones_A = MatrixXd(1,2*n_aug_+1);
    Ones_A.setOnes();

    for (int i=0;i<2*n_aug_+1;i++){
        Zsig.col(i) = getLidarMeasurement(Xsig_pred_.col(i));
    }
    weights(0) = lambda_/(lambda_+n_aug_);
    weights.tail(2*n_aug_).fill(1/2./(lambda_+n_aug_));
    //predict state mean
    z_pred = Zsig*weights;
    //calculate measurement covariance matrix S
    S =(Zsig-z_pred*Ones_A)*MatrixXd(weights.asDiagonal())*(Zsig-z_pred*Ones_A).t
    VectorXd R_d = VectorXd(n_z_);
    R_d << std_laspx_*std_laspx_,
            std_laspy_*std_laspy_;
    S = S + MatrixXd(R_d.asDiagonal());

```

```
    z_true = meas_package.raw_measurements_;

    MatrixXd Tc = MatrixXd(n_x_, n_z_);
    MatrixXd Z_diff = MatrixXd(n_z_,2*n_aug_+1);
    //calculate cross correlation matrix
    MatrixXd K = MatrixXd(n_x_,n_z_);
    Z_diff = (Zsig-z_pred*Ones_A);

    Tc = (Xsig_pred_-x_*Ones_A)*MatrixXd(weights.asDiagonal())*Z_diff.transpose()
    //calculate Kalman gain K;
    K = Tc*S.inverse();
    //update state mean and covariance matrix
    x_ = x_ + K *(z_true - z_pred);
    P_ = P_ - K*S*K.transpose();
    NIS_laser_ = (z_true - z_pred).transpose()*S*(z_true - z_pred);


}

/**
 * Updates the state and the state covariance matrix using a radar measurement.
 * @param {MeasurementPackage} meas_package
 */
void UKF::UpdateRadar(MeasurementPackage meas_package) {
  /**
  TODO:

  Complete this function! Use radar data to update the belief about the object's
  position. Modify the state vector, x_, and covariance, P_.

  You'll also need to calculate the radar NIS.
  */

    int n_z_ = 3;

    VectorXd weights = VectorXd(2*n_aug_+1);
    MatrixXd Wts_diag = MatrixXd(2*n_aug_+1,2*n_aug_+1);
    MatrixXd Zsig = MatrixXd(n_z_, 2 * n_aug_ + 1);
    MatrixXd S = MatrixXd(n_z_,n_z_);
    VectorXd z_pred = VectorXd(n_z_);
    VectorXd z_true = VectorXd(n_z_);
    MatrixXd Ones_A = MatrixXd(1,2*n_aug_+1);
    Ones_A.setOnes();

    for (int i=0;i<2*n_aug_+1;i++){
        Zsig.col(i) = getRadarMeasurement(Xsig_pred_.col(i));
    }
    weights(0) = lambda_/(lambda_+n_aug_);
    weights.tail(2*n_aug_).fill(1/2./(lambda_+n_aug_));
    //predict state mean
    z_pred = Zsig*weights;
    //calculate measurement covariance matrix S
    S =(Zsig-z_pred*Ones_A)*MatrixXd(weights.asDiagonal())*(Zsig-z_pred*Ones_A).t
    VectorXd R_d = VectorXd(n_z_);
    R_d << std_radr_*std_radr_,
            std_radphi_*std_radphi_,
            std_radrd_*std_radrd_;
    S = S + MatrixXd(R_d.asDiagonal());
    //cout<<"S"<<S<<endl;
    z_true = meas_package.raw_measurements_;

    MatrixXd Tc = MatrixXd(n_x_, n_z_);
    MatrixXd Z_diff = MatrixXd(n_z_,2*n_aug_+1);
    //calculate cross correlation matrix
    MatrixXd K = MatrixXd(n_x_,n_z_);
    Z_diff = (Zsig-z_pred*Ones_A);

    for (int i=0;i<n_z_;i++){
        Z_diff(i,2) = atan2(sin(Z_diff(i,2)),cos(Z_diff(i,2)));
```

SUGGESTION

Good work on normalizing the angle! This will help keep the filter stable and accurate.

- In order to make this code more readable and maintainable, you could implement a small function/me normalization.
- This function could be verified *once* using unit testing.

```

    }

    Tc = (Xsig_pred_-x_*Ones_A)*MatrixXd(weights.asDiagonal())*Z_diff.transpose()
    //calculate Kalman gain K;
    K = Tc*S.inverse();
    //update state mean and covariance matrix
    x_ = x_ + K *(z_true - z_pred);

    P_ = P_ - K*S*K.transpose();
```

```
  P_ = P_ - K*S*K.transpose();
  NIS_radar_ = (z_true - z_pred).transpose()*S*(z_true - z_pred);

}

```

- src/Eigen/src/LU/Inverse.h
- src/Eigen/src/LU/FullPivLU.h
- src/Eigen/src/LU/Determinant.h
- src/Eigen/src/LU/CMakeLists.txt
- src/Eigen/src/Jacobi/Jacobi.h
- src/Eigen/src/Jacobi/CMakeLists.txt
- src/Eigen/src/IterativeLinearSolvers/IterativeSolverBase.h
- src/Eigen/src/IterativeLinearSolvers/IncompleteLUT.h
- src/Eigen/src/IterativeLinearSolvers/ConjugateGradient.h
- src/Eigen/src/IterativeLinearSolvers/CMakeLists.txt
- src/Eigen/src/IterativeLinearSolvers/BiCGSTAB.h
- src/Eigen/src/IterativeLinearSolvers/BasicPreconditioners.h
- src/Eigen/src/Householder/HouseholderSequence.h
- src/Eigen/src/Householder/Householder.h
- src/Eigen/src/Householder/CMakeLists.txt
- src/Eigen/src/Householder/BlockHouseholder.h
- src/Eigen/src/Geometry/arch/Geometry_SSE.h
- src/Eigen/src/Geometry/arch/CMakeLists.txt
- src/Eigen/src/Geometry/Umeyama.h
- src/Eigen/src/Geometry/Translation.h
- src/Eigen/src/Geometry/Transform.h
- src/Eigen/src/Geometry/Scaling.h
- src/Eigen/src/Geometry/RotationBase.h
- src/Eigen/src/Geometry/Rotation2D.h
- src/Eigen/src/Geometry/Quaternion.h
- src/Eigen/src/Geometry/ParametrizedLine.h
- src/Eigen/src/Geometry/OrthoMethods.h

- src/Eigen/src/Geometry/Hyperplane.h
- src/Eigen/src/Geometry/Homogeneous.h
- src/Eigen/src/Geometry/EulerAngles.h
- src/Eigen/src/Geometry/CMakeLists.txt
- src/Eigen/src/Geometry/AngleAxis.h
- src/Eigen/src/Geometry/AlignedBox.h
- src/Eigen/src/Eigenvalues/Tridiagonalization.h
- src/Eigen/src/Eigenvalues/RealSchur_MKL.h
- src/Eigen/src/Eigenvalues/RealSchur.h
- src/Eigen/src/Eigenvalues/RealQZ.h
- src/Eigen/src/Eigenvalues/HessenbergDecomposition.h
- src/Eigen/src/Eigenvalues/ComplexSchur_MKL.h
- src/Eigen/src/Eigenvalues/ComplexSchur.h
- src/Eigen/src/Eigenvalues/CMakeLists.txt
- src/Eigen/src/Eigen2Support/VectorBlock.h
- src/Eigen/src/Eigen2Support/TriangularSolver.h
- src/Eigen/src/Eigen2Support/SVD.h
- src/Eigen/src/Eigen2Support/QR.h
- src/Eigen/src/Eigen2Support/Minor.h
- src/Eigen/src/Eigen2Support/Meta.h
- src/Eigen/src/Eigen2Support/Memory.h
- src/Eigen/src/Eigen2Support/MathFunctions.h
- src/Eigen/src/Eigen2Support/Macros.h
- src/Eigen/src/Eigen2Support/LeastSquares.h
- src/Eigen/src/Eigen2Support/Lazy.h
- src/Eigen/src/Eigen2Support/LU.h
- src/Eigen/src/Eigen2Support/Geometry/Translation.h
- src/Eigen/src/Eigen2Support/Geometry/Transform.h

- src/Eigen/src/Eigen2Support/Geometry/Scaling.h
- src/Eigen/src/Eigen2Support/Geometry/RotationBase.h
- src/Eigen/src/Eigen2Support/Geometry/Rotation2D.h
- src/Eigen/src/Eigen2Support/Geometry/Quaternion.h
- src/Eigen/src/Eigen2Support/Geometry/ParametrizedLine.h
- src/Eigen/src/Eigen2Support/Geometry/Hyperplane.h
- src/Eigen/src/Eigen2Support/Geometry/CMakeLists.txt
- src/Eigen/src/Eigen2Support/Geometry/AngleAxis.h
- src/Eigen/src/Eigen2Support/Geometry/All.h
- src/Eigen/src/Eigen2Support/Geometry/AlignedBox.h
- src/Eigen/src/Eigen2Support/CwiseOperators.h
- src/Eigen/src/Eigen2Support/Cwise.h
- src/Eigen/src/Eigen2Support/CMakeLists.txt
- src/Eigen/src/Eigen2Support/Block.h
- src/Eigen/src/Core/util/XprHelper.h
- src/Eigen/src/Core/util/StaticAssert.h
- src/Eigen/src/Core/util/ReenableStupidWarnings.h
- src/Eigen/src/Core/util/NonMPL2.h
- src/Eigen/src/Core/util/Meta.h
- src/Eigen/src/Core/util/Memory.h
- src/Eigen/src/Core/util/Macros.h
- src/Eigen/src/Core/util/MKL_support.h
- src/Eigen/src/Core/util/ForwardDeclarations.h
- src/Eigen/src/Core/util/DisableStupidWarnings.h
- src/Eigen/src/Core/util/Constants.h
- src/Eigen/src/Core/util/CMakeLists.txt
- src/Eigen/src/Core/util/BlasUtil.h
- src/Eigen/src/Core/products/TriangularSolverVector.h
- src/Eigen/src/Core/products/TriangularSolverMatrix_MKL.h

- src/Eigen/src/Core/products/TriangularSolverMatrix.h
- src/Eigen/src/Core/products/TriangularMatrixVector_MKL.h
- src/Eigen/src/Core/products/TriangularMatrixVector.h
- src/Eigen/src/Core/products/TriangularMatrixMatrix_MKL.h
- src/Eigen/src/Core/products/TriangularMatrixMatrix.h
- src/Eigen/src/Core/products/SelfadjointRank2Update.h
- src/Eigen/src/Core/products/SelfadjointProduct.h
- src/Eigen/src/Core/products/SelfadjointMatrixVector_MKL.h
- src/Eigen/src/Core/products/SelfadjointMatrixVector.h
- src/Eigen/src/Core/products/SelfadjointMatrixMatrix_MKL.h
- src/Eigen/src/Core/products/SelfadjointMatrixMatrix.h
- src/Eigen/src/Core/products/Parallelizer.h
- src/Eigen/src/Core/products/GeneralMatrixVector_MKL.h
- src/Eigen/src/Core/products/GeneralMatrixVector.h
- src/Eigen/src/Core/products/GeneralMatrixMatrix_MKL.h
- src/Eigen/src/Core/products/GeneralMatrixMatrixTriangular_MKL.h
- src/Eigen/src/Core/products/GeneralMatrixMatrixTriangular.h
- src/Eigen/src/Core/products/GeneralMatrixMatrix.h
- src/Eigen/src/Core/products/GeneralBlockPanelKernel.h
- src/Eigen/src/Core/products/CoeffBasedProduct.h
- src/Eigen/src/Core/products/CMakeLists.txt
- src/Eigen/src/Core/arch/SSE/PacketMath.h
- src/Eigen/src/Core/arch/SSE/MathFunctions.h
- src/Eigen/src/Core/arch/SSE/Complex.h
- src/Eigen/src/Core/arch/SSE/CMakeLists.txt
- src/Eigen/src/Core/arch/NEON/PacketMath.h
- src/Eigen/src/Core/arch/NEON/Complex.h
- src/Eigen/src/Core/arch/NEON/CMakeLists.txt

- src/Eigen/src/Core/arch/Default/Settings.h
- src/Eigen/src/Core/arch/Default/CMakeLists.txt
- src/Eigen/src/Core/arch/CMakeLists.txt
- src/Eigen/src/Core/arch/AltiVec/PacketMath.h
- src/Eigen/src/Core/arch/AltiVec/Complex.h
- src/Eigen/src/Core/arch/AltiVec/CMakeLists.txt
- src/Eigen/src/Core/Visitor.h
- src/Eigen/src/Core/VectorwiseOp.h
- src/Eigen/src/Core/VectorBlock.h
- src/Eigen/src/Core/TriangularMatrix.h
- src/Eigen/src/Core/Transpositions.h
- src/Eigen/src/Core/Transpose.h
- src/Eigen/src/Core/Swap.h
- src/Eigen/src/Core/Stride.h
- src/Eigen/src/Core/StableNorm.h
- src/Eigen/src/Core/SolveTriangular.h
- src/Eigen/src/Core/SelfCwiseBinaryOp.h
- src/Eigen/src/Core/SelfAdjointView.h
- src/Eigen/src/Core/Select.h
- src/Eigen/src/Core/Reverse.h
- src/Eigen/src/Core/ReturnByValue.h
- src/Eigen/src/Core/Replicate.h
- src/Eigen/src/Core/Ref.h
- src/Eigen/src/Core/Redux.h
- src/Eigen/src/Core/Random.h
- src/Eigen/src/Core/ProductBase.h
- src/Eigen/src/Core/PlainObjectBase.h
- src/Eigen/src/Core/PermutationMatrix.h
- src/Eigen/src/Core/NumTraits.h

- src/Eigen/src/Core/NoAlias.h
- src/Eigen/src/Core/NestByValue.h
- src/Eigen/src/Core/MatrixBase.h
- src/Eigen/src/Core/Matrix.h
- src/Eigen/src/Core/MathFunctions.h
- src/Eigen/src/Core/MapBase.h
- src/Eigen/src/Core/Map.h
- src/Eigen/src/Core/IO.h
- src/Eigen/src/Core/GlobalFunctions.h
- src/Eigen/src/Core/GenericPacketMath.h
- src/Eigen/src/Core/GeneralProduct.h
- src/Eigen/src/Core/Fuzzy.h
- src/Eigen/src/Core/Functors.h
- src/Eigen/src/Core/ForceAlignedAccess.h
- src/Eigen/src/Core/Flagged.h
- src/Eigen/src/Core/Dot.h
- src/Eigen/src/Core/DiagonalProduct.h
- src/Eigen/src/Core/DiagonalMatrix.h
- src/Eigen/src/Core/Diagonal.h
- src/Eigen/src/Core/DenseStorage.h
- src/Eigen/src/Core/DenseCoeffsBase.h
- src/Eigen/src/Core/DenseBase.h
- src/Eigen/src/Core/CwiseUnaryView.h
- src/Eigen/src/Core/CwiseUnaryOp.h
- src/Eigen/src/Core/CwiseNullaryOp.h
- src/Eigen/src/Core/CwiseBinaryOp.h
- src/Eigen/src/Core/CoreIterators.h
- src/Eigen/src/Core/CommaInitializer.h

- src/Eigen/src/Core/CMakeLists.txt
- src/Eigen/src/Core/BooleanRedux.h
- src/Eigen/src/Core/Block.h
- src/Eigen/src/Core/BandMatrix.h
- src/Eigen/src/Core/Assign_MKL.h
- src/Eigen/src/Core/Assign.h
- src/Eigen/src/Core/ArrayWrapper.h
- src/Eigen/src/Core/ArrayBase.h
- src/Eigen/src/Core/Array.h
- src/Eigen/src/CholmodSupport/CholmodSupport.h
- src/Eigen/src/CholmodSupport/CMakeLists.txt
- src/Eigen/src/Cholesky/LLT_MKL.h
- src/Eigen/src/Cholesky/LLT.h
- src/Eigen/src/Cholesky/LDLT.h
- src/Eigen/src/Cholesky/CMakeLists.txt
- src/Eigen/src/CMakeLists.txt
- src/Eigen/UmfPackSupport
- src/Eigen/SuperLUSupport
- src/Eigen/StdVector
- src/Eigen/StdList
- src/Eigen/StdDeque
- src/Eigen/SparseQR
- src/Eigen/SparseLU
- src/Eigen/SparseCore
- src/Eigen/SparseCholesky
- src/Eigen/Sparse
- src/Eigen/SVD
- src/Eigen/SPQRSupport
- src/Eigen/QtAlignedMalloc

- src/Eigen/QR
- src/Eigen/PardisoSupport
- src/Eigen/PaStiXSupport
- src/Eigen/OrderingMethods
- src/Eigen/MetisSupport
- src/Eigen/LeastSquares
- src/Eigen/LU
- src/Eigen/Jacobi
- src/Eigen/IterativeLinearSolvers
- src/Eigen/Householder
- src/Eigen/Geometry
- src/Eigen/Dense
- src/Eigen/Core
- src/Eigen/CholmodSupport
- src/Eigen/Cholesky
- src/Eigen/CMakeLists.txt
- src/Eigen/Array
- readme.txt
- output/output2.txt
- output/output1.txt
- data/sample-laser-radar-measurement-data-1.txt
- data/obj_pose-laser-radar-synthetic-input.txt
- cmake-build-debug/cmake_install.cmake
- cmake-build-debug/UnscentedKF.cbp
- cmake-build-debug/Makefile
- cmake-build-debug/CMakeFiles/progress.marks
- cmake-build-debug/CMakeFiles/feature_tests.cxx
- cmake-build-debug/CMakeFiles/feature_tests.c

- cmake-build-debug/CMakeFiles/feature_tests.bin
- cmake-build-debug/CMakeFiles/cmake.check_cache
- src/ukf.h
- src/tools.h
- src/tools.cpp
- src/measurement_package.h
- src/main.cpp
- src/ground_truth_package.h
- src/Eigen/src/plugins/MatrixCwiseUnaryOps.h
- src/Eigen/src/plugins/MatrixCwiseBinaryOps.h
- src/Eigen/src/plugins/CommonCwiseUnaryOps.h
- src/Eigen/src/plugins/CommonCwiseBinaryOps.h
- src/Eigen/src/plugins/CMakeLists.txt
- src/Eigen/src/plugins/BlockMethods.h
- src/Eigen/src/plugins/ArrayCwiseUnaryOps.h
- src/Eigen/src/plugins/ArrayCwiseBinaryOps.h
- src/Eigen/src/misc/blas.h
- src/Eigen/src/misc/SparseSolve.h
- src/Eigen/src/misc/Solve.h
- src/Eigen/src/misc/Kernel.h
- src/Eigen/src/misc/Image.h
- src/Eigen/src/misc/CMakeLists.txt
- src/Eigen/src/UmfPackSupport/UmfPackSupport.h
- src/Eigen/src/UmfPackSupport/CMakeLists.txt
- src/Eigen/src/SuperLUSupport/SuperLUSupport.h
- src/Eigen/src/SuperLUSupport/CMakeLists.txt
- src/Eigen/src/StlSupport/details.h
- src/Eigen/src/StlSupport/StdVector.h
- src/Eigen/src/StlSupport/StdList.h

- src/Eigen/src/StlSupport/StdDeque.h
- src/Eigen/src/StlSupport/CMakeLists.txt
- src/Eigen/src/SparseQR/SparseQR.h
- src/Eigen/src/SparseQR/CMakeLists.txt
- src/Eigen/src/SparseLU/SparseLU_relax_snode.h
- src/Eigen/src/SparseLU/SparseLU_pruneL.h
- src/Eigen/src/SparseLU/SparseLU_pivotL.h
- src/Eigen/src/SparseLU/SparseLU_panel_dfs.h
- src/Eigen/src/SparseLU/SparseLU_panel_bmod.h
- src/Eigen/src/SparseLU/SparseLU_kernel_bmod.h
- src/Eigen/src/SparseLU/SparseLU_heap_relax_snode.h
- src/Eigen/src/SparseLU/SparseLU_gemm_kernel.h
- src/Eigen/src/SparseLU/SparseLU_copy_to_ucol.h
- src/Eigen/src/SparseLU/SparseLU_column_dfs.h
- src/Eigen/src/SparseLU/SparseLU_column_bmod.h
- src/Eigen/src/SparseLU/SparseLU_Utils.h
- src/Eigen/src/SparseLU/SparseLU_SupernodalMatrix.h
- src/Eigen/src/SparseLU/SparseLU_Structs.h
- src/Eigen/src/SparseLU/SparseLU_Memory.h
- src/Eigen/src/SparseLU/SparseLUImpl.h
- src/Eigen/src/SparseLU/SparseLU.h
- src/Eigen/src/SparseLU/CMakeLists.txt
- src/Eigen/src/SparseCore/TriangularSolver.h
- src/Eigen/src/SparseCore/SparseView.h
- src/Eigen/src/SparseCore/SparseVector.h
- src/Eigen/src/SparseCore/SparseUtil.h
- src/Eigen/src/SparseCore/SparseTriangularView.h
- src/Eigen/src/SparseCore/SparseTranspose.h

- src/Eigen/src/SparseCore/SparseSparseProductWithPruning.h
- src/Eigen/src/SparseCore/SparseSelfAdjointView.h
- src/Eigen/src/SparseCore/SparseRedux.h
- src/Eigen/src/SparseCore/SparseProduct.h
- src/Eigen/src/SparseCore/SparsePermutation.h
- src/Eigen/src/SparseCore/SparseMatrixBase.h
- src/Eigen/src/SparseCore/SparseMatrix.h
- src/Eigen/src/SparseCore/SparseFuzzy.h
- src/Eigen/src/SparseCore/SparseDot.h
- src/Eigen/src/SparseCore/SparseDiagonalProduct.h
- src/Eigen/src/SparseCore/SparseDenseProduct.h
- src/Eigen/src/SparseCore/SparseCwiseUnaryOp.h
- src/Eigen/src/SparseCore/SparseCwiseBinaryOp.h
- src/Eigen/src/SparseCore/SparseColEtree.h
- src/Eigen/src/SparseCore/SparseBlock.h
- src/Eigen/src/SparseCore/MappedSparseMatrix.h
- src/Eigen/src/SparseCore/ConservativeSparseSparseProduct.h
- src/Eigen/src/SparseCore/CompressedStorage.h
- src/Eigen/src/SparseCore/CMakeLists.txt
- src/Eigen/src/SparseCore/AmbiVector.h
- src/Eigen/src/SparseCholesky/SimplicialCholesky_impl.h
- src/Eigen/src/SparseCholesky/SimplicialCholesky.h
- src/Eigen/src/SparseCholesky/CMakeLists.txt
- src/Eigen/src/SVD/UpperBidiagonalization.h
- src/Eigen/src/SVD/JacobiSVD_MKL.h
- src/Eigen/src/SVD/JacobiSVD.h
- src/Eigen/src/SVD/CMakeLists.txt
- src/Eigen/src/SPQRSupport/SuiteSparseQRSupport.h
- src/Eigen/src/SPQRSupport/CMakeLists.txt

- src/Eigen/src/QR/HouseholderQR_MKL.h
- src/Eigen/src/QR/HouseholderQR.h
- src/Eigen/src/QR/FullPivHouseholderQR.h
- src/Eigen/src/QR/ColPivHouseholderQR_MKL.h
- src/Eigen/src/QR/ColPivHouseholderQR.h
- src/Eigen/src/QR/CMakeLists.txt
- src/Eigen/src/PardisoSupport/PardisoSupport.h
- src/Eigen/src/PardisoSupport/CMakeLists.txt
- src/Eigen/src/PaStiXSupport/PaStiXSupport.h
- src/Eigen/src/PaStiXSupport/CMakeLists.txt
- src/Eigen/src/OrderingMethods/Ordering.h
- src/Eigen/src/OrderingMethods/CMakeLists.txt
- src/Eigen/src/OrderingMethods/Amd.h
- src/Eigen/src/MetisSupport/MetisSupport.h
- src/Eigen/src/MetisSupport/CMakeLists.txt
- src/Eigen/src/LU/arch/Inverse_SSE.h
- src/Eigen/src/LU/arch/CMakeLists.txt
- src/Eigen/src/LU/PartialPivLU_MKL.h
- src/Eigen/src/LU/PartialPivLU.h
- cmake-build-debug/CMakeFiles/clion-log.txt
- cmake-build-debug/CMakeFiles/clion-environment.txt
- cmake-build-debug/CMakeFiles/UnscentedKF.dir/progress.make
- cmake-build-debug/CMakeFiles/UnscentedKF.dir/link.txt
- cmake-build-debug/CMakeFiles/UnscentedKF.dir/flags.make
- cmake-build-debug/CMakeFiles/UnscentedKF.dir/depend.make
- cmake-build-debug/CMakeFiles/UnscentedKF.dir/cmake_clean.cmake
- cmake-build-debug/CMakeFiles/UnscentedKF.dir/build.make
- cmake-build-debug/CMakeFiles/UnscentedKF.dir/DependInfo.cmake

- cmake-build-debug/CMakeFiles/TargetDirectories.txt
- cmake-build-debug/CMakeFiles/Makefile2
- cmake-build-debug/CMakeFiles/Makefile.cmake
- cmake-build-debug/CMakeFiles/CMakeOutput.log
- cmake-build-debug/CMakeFiles/CMakeDirectoryInformation.cmake
- cmake-build-debug/CMakeFiles/3.6.3/CompilerIdCXX/a.out
- cmake-build-debug/CMakeFiles/3.6.3/CompilerIdCXX/CMakeCXXCompilerId.cpp
- cmake-build-debug/CMakeFiles/3.6.3/CompilerIdC/a.out
- cmake-build-debug/CMakeFiles/3.6.3/CompilerIdC/CMakeCCompilerId.c
- cmake-build-debug/CMakeFiles/3.6.3/CMakeSystem.cmake
- cmake-build-debug/CMakeFiles/3.6.3/CMakeDetermineCompilerABI_CXX.bin
- cmake-build-debug/CMakeFiles/3.6.3/CMakeDetermineCompilerABI_C.bin
- cmake-build-debug/CMakeFiles/3.6.3/CMakeCXXCompiler.cmake
- cmake-build-debug/CMakeFiles/3.6.3/CMakeCCompiler.cmake
- cmake-build-debug/CMakeCache.txt
- build/cmake_install.cmake
- build/UnscentedKF
- build/Makefile
- build/CMakeFiles/progress.marks
- build/CMakeFiles/feature_tests.cxx
- build/CMakeFiles/feature_tests.c
- build/CMakeFiles/feature_tests.bin
- build/CMakeFiles/cmake.check_cache
- build/CMakeFiles/UnscentedKF.dir/src/ukf.cpp.o
- build/CMakeFiles/UnscentedKF.dir/src/tools.cpp.o
- build/CMakeFiles/UnscentedKF.dir/src/main.cpp.o
- build/CMakeFiles/UnscentedKF.dir/progress.make
- build/CMakeFiles/UnscentedKF.dir/link.txt
- build/CMakeFiles/UnscentedKF.dir/flags.make

- build/CMakeFiles/UnscentedKF.dir/depend.make
- build/CMakeFiles/UnscentedKF.dir/depend.internal
- build/CMakeFiles/UnscentedKF.dir/cmake_clean.cmake
- build/CMakeFiles/UnscentedKF.dir/build.make
- build/CMakeFiles/UnscentedKF.dir/DependInfo.cmake
- build/CMakeFiles/UnscentedKF.dir/CXX.includecache
- build/CMakeFiles/TargetDirectories.txt
- build/CMakeFiles/Makefile2
- build/CMakeFiles/Makefile.cmake
- build/CMakeFiles/CMakeOutput.log
- build/CMakeFiles/CMakeDirectoryInformation.cmake
- build/CMakeFiles/3.8.1/CompilerIdCXX/a.out
- build/CMakeFiles/3.8.1/CompilerIdCXX/CMakeCXXCompilerId.cpp
- build/CMakeFiles/3.8.1/CompilerIdC/a.out
- build/CMakeFiles/3.8.1/CompilerIdC/CMakeCCompilerId.c
- build/CMakeFiles/3.8.1/CMakeSystem.cmake
- build/CMakeFiles/3.8.1/CMakeDetermineCompilerABI_CXX.bin
- build/CMakeFiles/3.8.1/CMakeDetermineCompilerABI_C.bin
- build/CMakeFiles/3.8.1/CMakeCXXCompiler.cmake
- build/CMakeFiles/3.8.1/CMakeCCompiler.cmake
- build/CMakeCache.txt
- README.md
- CMakeLists.txt