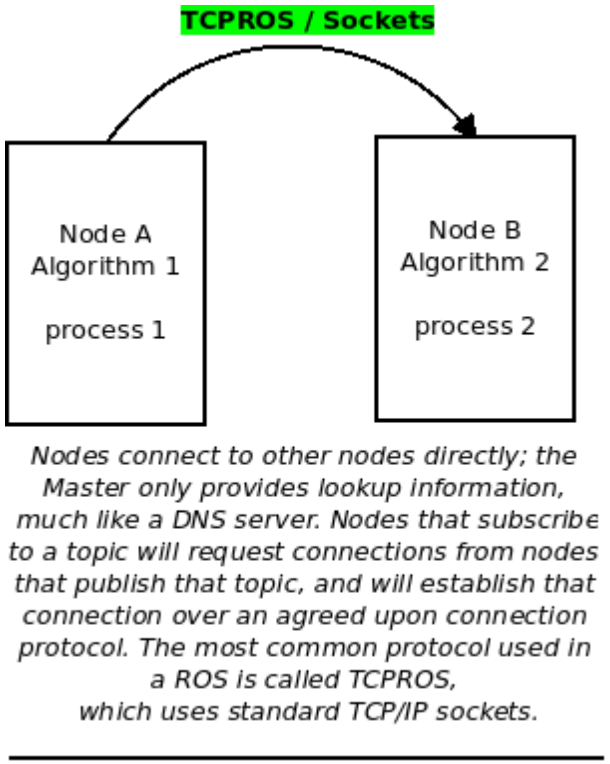
Robot localization code analysis

- **EKF**
- **NAVSAT_TRANSFORM**

Reference : Robot localization github

발표자 : 이담

- **Node**
  - 노드는 다른 노드에 직접 연결
- **Nodelet**
  - Nodelet은 단일 프로세스로 여러 개의 노드를 실행
    > Nodelet은 ROS의 성능을 더 많이 발휘하는데 있어서 중요
    > Nodelet은 많은 양의 데이터가 포함된 메시지를 사용해야하는 프로세스가 여러 개 있을 때, Node는 TCP를 통해 데이터 자체를 보내지만 Nodelet은 해당 데이터에 대한 boost shared 포인터를 보내기 때문에 빠름(**Zero copy**)



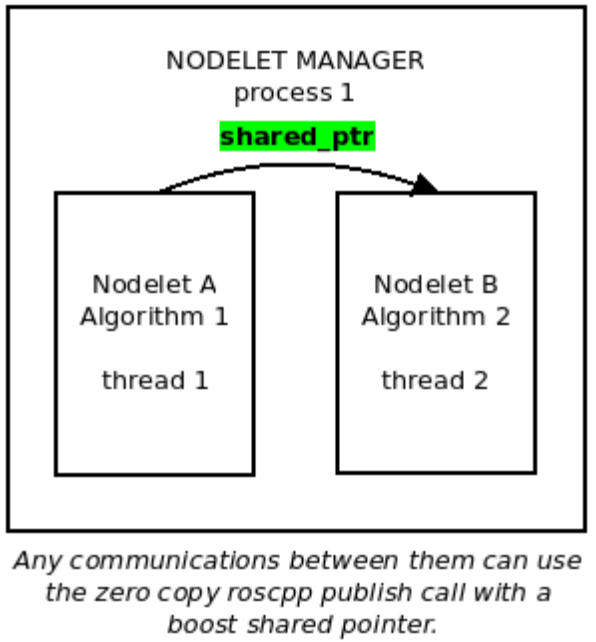


그림 1. Node와 Nodelet

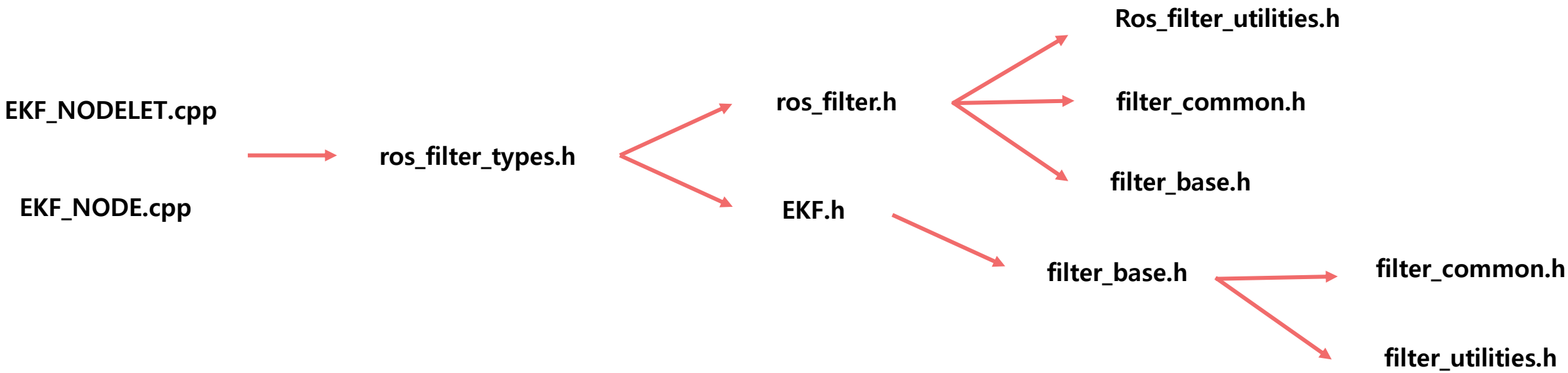

EKF_NODELET.cpp
EKF_NODE.cpp
ros_filter_types.h
ros_filter.h
EKF.h
Ros_filter_utilities.h
filter_common.h
filter_base.h
filter_base.h
filter_common.h
filter_utilities.h

그림 2. EKF 흐름도

filter_common.h

Enum StateMembers
- x,y,z,r,p,y,Vx,Vy,Vz,Vr,Vp
Vy,Ax,Ay,Az

Enum ControlMember
(제어 벡터)
- Vx,Vy,Vz,Vr,Vp,Vy

Const int
(Our State Global Constant)
: vector size & offset
- STATE_SIZE
- POSITION_OFFSET = StateMemberX;
- ORIENTATION_OFFSET = StateMemberRoll;
- POSITION_V_OFFSET = StateMemberVx;
- ORIENTATION_V_OFFSET = StateMemberVroll;
- POSITION_A_OFFSET = StateMemberAx;

(**Pose and Twist message)**
- POSE_SIZE
- TWIST_SIZE
- POSITION_SIZE
- ORIENTATION_SIZE
- LINEAR_VELOCITY_SIZE
- ACCELERATION_SIZE

(Common variables)
- PI
- TAU

filter_utilities.h

유틸리티 : 필수적 X 최적화 O

Eigen – MatrixXd, VectorXd 출력 관리
Std – vector(size_t), vector(int) 출력 관리

RPY Roatation 관리 – 범위 유지
Tf관리 – tf2 추가
: frame id+tfprefix > new frameid

filter_base.h

Struct Measurement

: 측정값(m or rad / s) 저장 비교 / Queue가 우선선위

- double latestControlTime_;
- double mahalanobisThresh_;
- double time_;
- std::string topicName_;
- std::vector<int> updateVector_;
- Eigen::VectorXd latestControl_;
- Eigen::VectorXd measurement_;
- Eigen::MatrixXd covariance_;

Struct FilterState

: 필터 상태 저장, 비교 / 측정값(m or rad / s)

- double lastMeasurementTime_;
- double latestControlTime_;
- Eigen::VectorXd latestControl_;
- Eigen::VectorXd state_;
- Eigen::MatrixXd estimateErrorCovariance_;
- bool operator()(const FilterState &a, const FilterState &b)

Class FilterBase

: public

- 동적프로세스노이즈공분산계산
- 예측과 관련된 멤버 함수와 멤버 변수
- 보정(업데이트)과 관련된 멤버 함수와 멤버 변수

:protected

- 벡터 업데이트, 가속, 감속, 델타, 타임아웃
- inline double computeControlAcceleration
- 공분산 관련된 멤버 변수
- 마할라노비스거리 thred 멤버 함수
- **Eigen::VectorXd state_;**

: 로봇의 상태 벡터로 이 벡터의 값은 노드에서 보고하는 값

- Eigen::MatrixXd transferFunction_; > KF의 A행렬
- Eigen::MatrixXd transferFunctionJacobian_; > 자코비안행렬
- Eigen::MatrixXd processNoiseCovariance_; > 공분산 행렬에 추가

ekf.h

class Ekf: public FilterBase

: FilterBase에서 파생되며 이산 시간 EKF 알고리즘에 따라 predict() 및 correct() 메서드를 재정의

- explicit Ekf(std::vector<double> args = std::vector<double>());
- void correct(const Measurement &measurement);
- void predict(const double referenceTime, const double delta);

Ros_filter_utilities.h

- TF 관련 헤더 :
#include <tf2/LinearMath/Quaternion.h> //tf
#include <tf2/LinearMath/Transform.h>
#include <tf2_ros/buffer.h>

- lookupTransformSafe
:변환 안전하게 받기
1. 특정 시간에 sourceFrame에서 targetFrame으로 TF 시도
2. 사용할 수 있는 TF 이 없으면 단순히 최신 TF 시도
3. 실패시 메서드는 TF가 지정된 frame_id에서 그자체로 이동하는지 확인
4. 하나라도 성공하면 메서드는 @p targetFrameTrans의 값을 설정하고 true를 반환, 그렇지 않으면 false를 반환

- void quatToRPY(const tf2::Quaternion &quat, double &roll, double &pitch, double &yaw);
- void stateToTF(const Eigen::VectorXd &state, tf2::Transform &stateTF);
- void TFtoState(const tf2::Transform &stateTF, Eigen::VectorXd &state);

Callback
Pub,sub,update
broadcast,listner,
Frequency
Time
TF 관련 함수
Frame id, child id
필터 통합 imu, acc, pose등

Ros_filter.h

Struct CallbackData
- std::string topicName_;
- std::vector<int> updateVector_;
- int updateSum_;
- bool differential_;
- bool relative_;
- bool pose_use_child_frame_;
- double rejectionThreshold_;

RosFilter template class
Public :
bool toggleFilterProcessingCallback
- void accelerationCallback
- void controlCallback
- void enqueueMeasurement
- void forceTwoD
bool getFilteredOdometryMessage
- bool getFilteredAccelMessage
- void imuCallback
- void integrateMeasurements
- void differentiateMeasurements
- void loadParams
- void odometryCallback
- void poseCallback
- void setPoseCallback
- bool setPoseSrvCallback
- bool enableFilterSrvCallback
- void twistCallback
- bool validateFilterOutput

Protected :
- Diagnostics : add, aggregate(집계)
- Copycovariance
: ROS 공분산 배열 → 고유 행렬
고유 행렬 → ROS 공분산 배열
- Prepare : 악셀, pose, twist,
- Publish, Enable, Print 관련
- FrameID 관련

Ros_filter_type.h

typedef RosFilter<Ekf> RosEkf;
typedef RosFilter<Ukf> RosUkf;

Correct

- Robotlocalization namespace에 EKF class의 생성자, 소멸자 선언
- EKF class의 void correct 함수 선언
- FB_DEBUG : getdebug면 FB_DEBUG의 msg 출력

```
namespace RobotLocalization
{
  Ekf::Ekf(std::vector<double>) :
    FilterBase()
  {
  }

  Ekf::~Ekf()
  {
  }

  void Ekf::correct(const Measurement &measurement)
  {
    FB_DEBUG("---------------------- Ekf::correct ----------------------\n" <<
             "State is:\n" << state_ << "\n"
             "Topic is:\n" << measurement.topicName_ << "\n"
             "Measurement is:\n" << measurement.measurement_ << "\n"
             "Measurement topic name is:\n" << measurement.topicName_ << "\n\n"
             "Measurement covariance is:\n" << measurement.covariance_ << "\n");
```

- 모든 것을 업데이트하고 싶지 않으므로, 상태 벡터의 측정된 부분만 업데이트하는 행렬을 만들어야 함
- **먼저 업데이트할 상태 벡터 값의 수를 결정**

1. For문에서 measurement의 updaterVector의 size를 하나씩 키우면서 볼 때, 측정이 Nan이면 Debug msg출력, Inf일 때도 Debug msg출력

- 여기서, NaN(Not a Number), InF(Infinite)

2. Nan, InF도 아니면 updateIndices에 measurement.updaterVector을 차례차례 push해라

```
std::vector<size_t> updateIndices;
for (size_t i = 0; i < measurement.updateVector_.size(); ++i)
{
  if (measurement.updateVector_[i])
  {
    if (std::isnan(measurement.measurement_(i)))
    {
      FB_DEBUG("Value at index " << i << " was nan. Excluding from update.\n");
    }
    else if (std::isinf(measurement.measurement_(i)))
    {
      FB_DEBUG("Value at index " << i << " was inf. Excluding from update.\n");
    }
    else
    {
      updateIndices.push_back(i);
    }
  }
}
```

- FB_DEBUG msg 출력 : Update Indices(업데이트 인자)
- Size_t updateSize = updateIndices.size(); > size_t = long unsigned int : 자료형 지정
- 관련 행렬들 셋업 : x, z, R, H, K, z-Hx
- 관련 행렬들 setZero() : value = 0으로 초기화

```
FB_DEBUG("Update indices are:\n" << updateIndices << "\n");

size_t updateSize = updateIndices.size();

Eigen::VectorXd stateSubset(updateSize);                              // x (in most literature)
Eigen::VectorXd measurementSubset(updateSize);                        // z
Eigen::MatrixXd measurementCovarianceSubset(updateSize, updateSize);  // R
Eigen::MatrixXd stateToMeasurementSubset(updateSize, state_.rows());  // H
Eigen::MatrixXd kalmanGainSubset(state_.rows(), updateSize);          // K
Eigen::VectorXd innovationSubset(updateSize);                         // z - Hx

stateSubset.setZero();
measurementSubset.setZero();
measurementCovarianceSubset.setZero();
stateToMeasurementSubset.setZero();
kalmanGainSubset.setZero();
innovationSubset.setZero();
```

- 전체 크기 행렬에서 부분 행렬을 만듬
- x : 현재 상태값, z : 현재 측정값, R : 측정공분산 set
- x, z는 벡터형식이므로 nx1이라 updateIndices[i]를 받아오고, R은 행렬이므로 nxn이라 (i , j) 를 받아옴
- 여기서 R에 대한 오류 (음수, 0에 가까움)를 처리할 때, (i , i)를 하는 이유 > 공분산은 대칭이니까 ?

R(측정공분산)이 음수일 때
::fabs > 절대값

R(측정공분산)이 1e-9보다 작을 때, 즉 0에 가까울 때
R = 1e-9 로 정의

```
for (size_t i = 0; i < updateSize; ++i)
{
X measurementSubset(i) = measurement.measurement_(updateIndices[i]);
Z stateSubset(i) = state_(updateIndices[i]);

  for (size_t j = 0; j < updateSize; ++j)
  {
R   measurementCovarianceSubset(i, j) = measurement.covariance_(updateIndices[i], updateIndices[j]);
  }

  if (measurementCovarianceSubset(i, i) < 0)
  {
    FB_DEBUG("WARNING: Negative covariance for index " << i <<
             " of measurement (value is" << measurementCovarianceSubset(i, i) <<
             "). Using absolute value...\n");

    measurementCovarianceSubset(i, i) = ::fabs(measurementCovarianceSubset(i, i));
  }

  if (measurementCovarianceSubset(i, i) < 1e-9)
  {
    FB_DEBUG("WARNING: measurement had very small error covariance for index " << updateIndices[i] <<
             ". Adding some noise to maintain filter stability.\n");

    measurementCovarianceSubset(i, i) = 1e-9;
  }
}
```

- H : stateToMeasurementSubset(updateSize, state_.rows())
- 행은 updateSize, 열은 nan,inf가 아닌 값이 들어가 있는 measurement.updatevector
- 업데이트할 값의 (i, i) 위치에 **1**이 있음
- FB_DEBUG msg 출력 : 현재 x, z, R, H

1. Kalman Gain K : K = (PH') / (HPH' + R) 정의 및 innovationSubset = z-Hx 정의

Statemember가 enum(열거형)에 정의
각각 값을 3, 4, 5가짐

- updateIndices[i]==StateMemberRoll or Pitch or Yaw(= 3,4,5)일 때, z-Hx 가 < -PI 이면 + TAU, > PI 이면 - TAU
  - ✓ 항상 –PI < (z-Hx)(i) < PI 범위를 가짐 : wrap angle (∵ roll, pitch, yaw 기본 단위 각도)

```
for (size_t i = 0; i < updateSize; ++i)
{
H stateToMeasurementSubset(i, updateIndices[i]) = 1;
}

FB_DEBUG("Current state subset is:\n" << stateSubset <<
         "\nMeasurement subset is:\n" << measurementSubset <<
         "\nMeasurement covariance subset is:\n" << measurementCovarianceSubset <<
         "\nState-to-measurement subset is:\n" << stateToMeasurementSubset << "\n");

// (1) Compute the Kalman gain: K = (PH') / (HPH' + R)
Eigen::MatrixXd pht = estimateErrorCovariance_ * stateToMeasurementSubset.transpose();
Eigen::MatrixXd hphrInv  = (stateToMeasurementSubset * pht + measurementCovarianceSubset).inverse();
kalmanGainSubset.noalias() = pht * hphrInv;

innovationSubset = (measurementSubset - stateSubset);
```

```
for (size_t i = 0; i < updateSize; ++i)
{
  if (updateIndices[i] == StateMemberRoll  ||
      updateIndices[i] == StateMemberPitch ||
      updateIndices[i] == StateMemberYaw)
  {
    while (innovationSubset(i) < -PI)
    {
      innovationSubset(i) += TAU;
    }

    while (innovationSubset(i) > PI)
    {
      innovationSubset(i) -= TAU;
    }
  }
}
```

병렬화 기능을 제대로 활용하기 위해서는 대상체가 서로 무관해야 함
서로 다른 신호를 구별함으로써 **최적화**하는 함수

2. 매핑된 z(측정값), x(상태값) 사이의 Mahalanobis 거리 확인 → 거리 측정 방법, 공분산의 크기에 따라 가중치를 고려해서 거리 계산

virtual bool checkMahalanobisThreshold**(const Eigen::VectorXd &innovation, const Eigen::MatrixXd &invCovariance, const double nsigmas);**

- 만약, innovation이 공분산의 Nsigma 내에 있는지 여부 > 즉, z-Hx가 (HPH' + R)의 마할라노비스 임계값 내에 있는지 확인

3. z(측정값), x(상태값) 사이의 게인 적용 : x = x + K(z - Hx)

4. 추정 오차 공분산 업데이트 : (I - KH)P(I - KH)' + KRK'

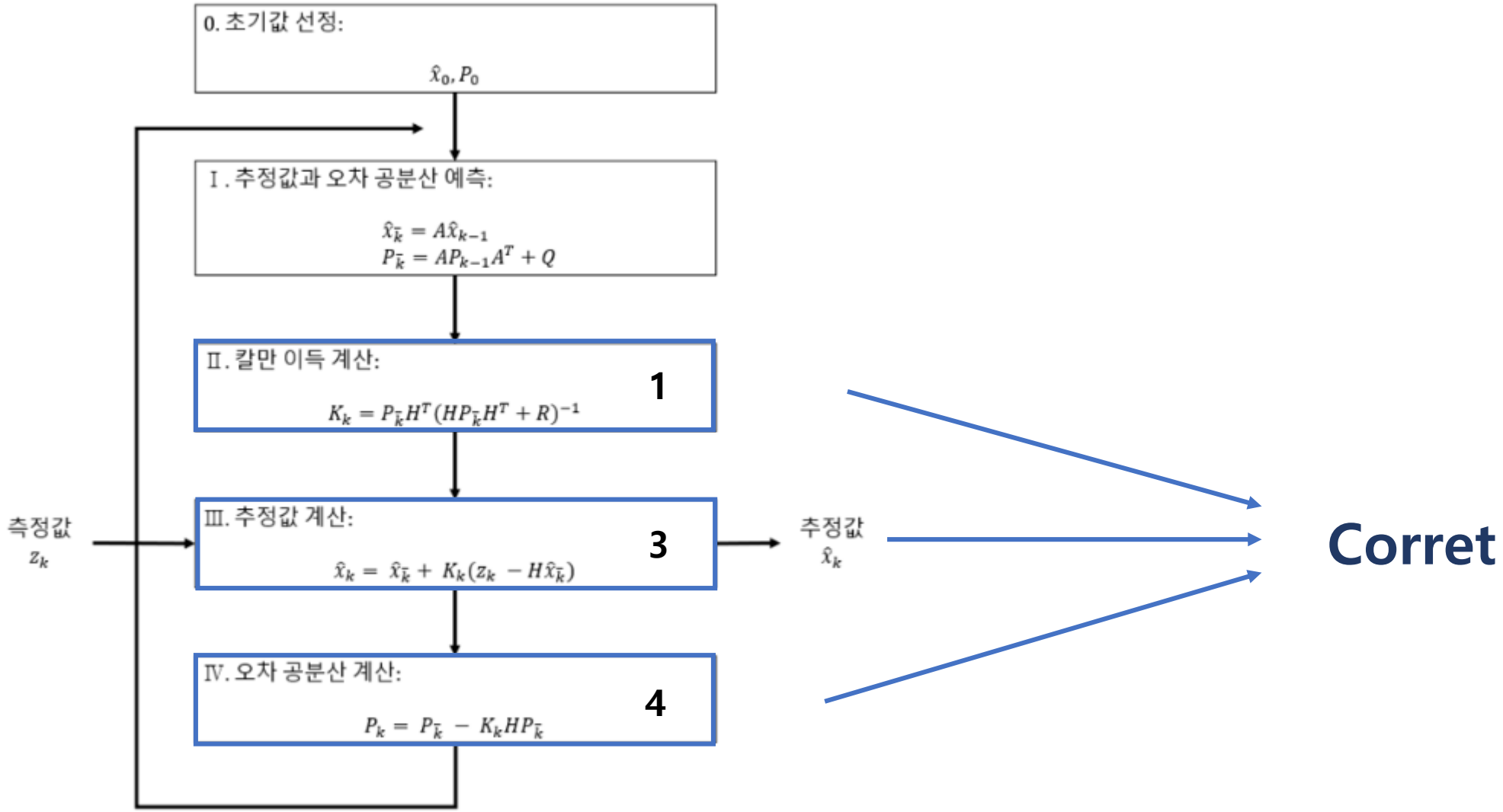
- FB_DEBUG msg 출력 : k, z-Hx, 업데이트된 full state, 업데이트된 full state의 공분산

```
// (2) Check Mahalanobis distance between mapped measurement and state.
if (checkMahalanobisThreshold(innovationSubset, hphrInv, measurement.mahalanobisThresh_))
{
  // (3) Apply the gain to the difference between the state and measurement: x = x + K(z - Hx)
  state_.noalias() += kalmanGainSubset * innovationSubset;

  // (4) Update the estimate error covariance using the Joseph form: (I - KH)P(I - KH)' + KRK'
  Eigen::MatrixXd gainResidual = identity_;
  gainResidual.noalias() -= kalmanGainSubset * stateToMeasurementSubset;
  estimateErrorCovariance_ = gainResidual * estimateErrorCovariance_ * gainResidual.transpose();
  estimateErrorCovariance_.noalias() += kalmanGainSubset *
                                        measurementCovarianceSubset *
                                        kalmanGainSubset.transpose();

  wrapStateAngles();

  FB_DEBUG("Kalman gain subset is:\n" << kalmanGainSubset <<
           "\nInnovation is:\n" << innovationSubset <<
           "\nCorrected full state is:\n" << state_ <<
           "\nCorrected full estimate error covariance is:\n" << estimateErrorCovariance_ <<
           "\n\n---------------------- /Ekf::correct ----------------------\n");
}
```

- 오차 공분산 계산

(I - KH)P(I - KH)' + KRK' vs P-KHP

- 오차 공분산을 계산한다는 것에서는 같은 역할을 하며, robot localization에서 쓴 오차 공분산 계산이 수치적으로 더 안정적임



Predict

```
void Ekf::predict(const double referenceTime, const double delta)
{
  FB_DEBUG("---------------------- Ekf::predict ----------------------\n" <<
           "delta is " << delta << "\n" <<
           "state is " << state_ << "\n");

  double roll = state_(StateMemberRoll);
  double pitch = state_(StateMemberPitch);
  double yaw = state_(StateMemberYaw);
  double xVel = state_(StateMemberVx);
  double yVel = state_(StateMemberVy);
  double zVel = state_(StateMemberVz);
  double pitchVel = state_(StateMemberVpitch);
  double yawVel = state_(StateMemberVyaw);
  double xAcc = state_(StateMemberAx);
  double yAcc = state_(StateMemberAy);
  double zAcc = state_(StateMemberAz);

  // We'll need these trig calculations a lot.
  double sp = ::sin(pitch);
  double cp = ::cos(pitch);
  double cpi = 1.0 / cp;
  double tp = sp * cpi;

  double sr = ::sin(roll);
  double cr = ::cos(roll);

  double sy = ::sin(yaw);
  double cy = ::cos(yaw);

  prepareControl(referenceTime, delta);
```

- EKF class의 void predict 함수 선언
- referenceTime : 업데이트 시간(예측 단계에서 사용된 측정)
- delta : 예측을 수행하는 데 걸리는 시간
- FB_DEBUG : getdebug면 FB_DEBUG의 msg 출력
- R, P, Y, xV, yV, zV, pV, yV, xA, yA,, zA 정의
- 삼각함수 R, P, Y 정의
- Preparecontrol : 제어 텀을 예측 단계에 적용할 가속도로 변환

```
// Prepare the transfer function
transferFunction_(StateMemberX, StateMemberVx) = cy * cp * delta;
transferFunction_(StateMemberX, StateMemberVy) = (cy * sp * sr - sy * cr) * delta;
transferFunction_(StateMemberX, StateMemberVz) = (cy * sp * cr + sy * sr) * delta;
transferFunction_(StateMemberX, StateMemberAx) = 0.5 * transferFunction_(StateMemberX, StateMemberVx) * delta;
transferFunction_(StateMemberX, StateMemberAy) = 0.5 * transferFunction_(StateMemberX, StateMemberVy) * delta;
transferFunction_(StateMemberX, StateMemberAz) = 0.5 * transferFunction_(StateMemberX, StateMemberVz) * delta;
transferFunction_(StateMemberY, StateMemberVx) = sy * cp * delta;
transferFunction_(StateMemberY, StateMemberVy) = (sy * sp * sr + cy * cr) * delta;
transferFunction_(StateMemberY, StateMemberVz) = (sy * sp * cr - cy * sr) * delta;
transferFunction_(StateMemberY, StateMemberAx) = 0.5 * transferFunction_(StateMemberY, StateMemberVx) * delta;
transferFunction_(StateMemberY, StateMemberAy) = 0.5 * transferFunction_(StateMemberY, StateMemberVy) * delta;
transferFunction_(StateMemberY, StateMemberAz) = 0.5 * transferFunction_(StateMemberY, StateMemberVz) * delta;
transferFunction_(StateMemberZ, StateMemberVx) = -sp * delta;
transferFunction_(StateMemberZ, StateMemberVy) = cp * sr * delta;
transferFunction_(StateMemberZ, StateMemberVz) = cp * cr * delta;
transferFunction_(StateMemberZ, StateMemberAx) = 0.5 * transferFunction_(StateMemberZ, StateMemberVx) * delta;
transferFunction_(StateMemberZ, StateMemberAy) = 0.5 * transferFunction_(StateMemberZ, StateMemberVy) * delta;
transferFunction_(StateMemberZ, StateMemberAz) = 0.5 * transferFunction_(StateMemberZ, StateMemberVz) * delta;
transferFunction_(StateMemberRoll, StateMemberVroll) = delta;
transferFunction_(StateMemberRoll, StateMemberVpitch) = sr * tp * delta;
transferFunction_(StateMemberRoll, StateMemberVyaw) = cr * tp * delta;
transferFunction_(StateMemberPitch, StateMemberVpitch) = cr * delta;
transferFunction_(StateMemberPitch, StateMemberVyaw) = -sr * delta;
transferFunction_(StateMemberYaw, StateMemberVpitch) = sr * cpi * delta;
transferFunction_(StateMemberYaw, StateMemberVyaw) = cr * cpi * delta;
transferFunction_(StateMemberVx, StateMemberAx) = delta;
transferFunction_(StateMemberVy, StateMemberAy) = delta;
transferFunction_(StateMemberVz, StateMemberAz) = delta;
```

$$\boldsymbol{R}_x = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\theta_x & -\sin\theta_x \\ 0 & \sin\theta_x & \cos\theta_x \end{pmatrix}$$

$$\boldsymbol{R}_y = \begin{pmatrix} \cos\theta_y & 0 & \sin\theta_y \\ 0 & 1 & \\ -\sin\theta_y & 0 & \cos\theta_y \end{pmatrix}$$

$$\boldsymbol{R}_z = \begin{pmatrix} \cos\theta_y & -\sin\theta_y & 0 \\ \sin\theta_y & \cos\theta_y & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

$$\mathrm{R} = \mathrm{R_z R_y R_x}$$

$$= \begin{pmatrix} \cos\theta_y\cos\theta_z & \sin\theta_x\sin\theta_y\cos\theta_z - \cos\theta_x\sin\theta_z & \cos\theta_x\sin\theta_y\cos\theta_z + \sin\theta_x\sin\theta_z \\ \cos\theta_y\sin\theta_z & \sin\theta_x\sin\theta_y\sin\theta_z + \cos\theta_x\cos\theta_z & \cos\theta_x\sin\theta_y\sin\theta_z - \sin\theta_x\cos\theta_z \\ -\sin\theta_y & \sin\theta_x\cos\theta_y & \cos\theta_x\cos\theta_y \end{pmatrix}$$

- 3차원 공간에서 자세를 표현하기 위해 나타내는 3개의 각도 > 오일러각
- Pose ? 위치를 설명하는 x,y,z 세 개의 직교 좌표 외에 3개의 각도 Rx, Ry 및 Rz는 3의 방향을 설명함
- Rx, Ry 및 Rz 는 (1, 0, 0), (0, 1, 0) 및 (0, 0, 1)에 대한 회전 > 회전 행렬
- 회전 변환 행렬 : 좌표계에서 회전 변환을 할 때 사용하는 행렬
- R = Roll, Pitch, Yaw 순서로 회전

- 식에 sin, cos이 들어가므로 비선형임
- 따라서 테일러 1차를 통해 편미분을 하고, 거기서 나온 행렬을 자코비안으로 정의

```
// Prepare the transfer function Jacobian. This function is analytically derived from the
// transfer function.
double xCoeff = 0.0;
double yCoeff = 0.0;
double zCoeff = 0.0;
double oneHalfATSquared = 0.5 * delta * delta;

yCoeff = cy * sp * cr + sy * sr;
zCoeff = -cy * sp * sr + sy * cr;
double dFx_dR = (yCoeff * yVel + zCoeff * zVel) * delta +
                (yCoeff * yAcc + zCoeff * zAcc) * oneHalfATSquared;
double dFR_dR = 1.0 + (cr * tp * pitchVel - sr * tp * yawVel) * delta;

xCoeff = -cy * sp;
yCoeff = cy * cp * sr;
zCoeff = cy * cp * cr;
double dFx_dP = (xCoeff * xVel + yCoeff * yVel + zCoeff * zVel) * delta +
                (xCoeff * xAcc + yCoeff * yAcc + zCoeff * zAcc) * oneHalfATSquared;
double dFR_dP = (cpi * cpi * sr * pitchVel + cpi * cpi * cr * yawVel) * delta;

xCoeff = -sy * cp;
yCoeff = -sy * sp * sr - cy * cr;
zCoeff = -sy * sp * cr + cy * sr;
double dFx_dY = (xCoeff * xVel + yCoeff * yVel + zCoeff * zVel) * delta +
                (xCoeff * xAcc + yCoeff * yAcc + zCoeff * zAcc) * oneHalfATSquared;

yCoeff = sy * sp * cr - cy * sr;
zCoeff = -sy * sp * sr - cy * cr;
double dFy_dR = (yCoeff * yVel + zCoeff * zVel) * delta +
                (yCoeff * yAcc + zCoeff * zAcc) * oneHalfATSquared;
double dFP_dR = (-sr * pitchVel - cr * yawVel) * delta;

xCoeff = -sy * sp;
yCoeff = sy * cp * sr;
zCoeff = sy * cp * cr;
double dFy_dP = (xCoeff * xVel + yCoeff * yVel + zCoeff * zVel) * delta +
                (xCoeff * xAcc + yCoeff * yAcc + zCoeff * zAcc) * oneHalfATSquared;
```

```
xCoeff = cy * cp;
yCoeff = cy * sp * sr - sy * cr;
zCoeff = cy * sp * cr + sy * sr;
double dFy_dY = (xCoeff * xVel + yCoeff * yVel + zCoeff * zVel) * delta +
                (xCoeff * xAcc + yCoeff * yAcc + zCoeff * zAcc) * oneHalfATSquared;

yCoeff = cp * cr;
zCoeff = -cp * sr;
double dFz_dR = (yCoeff * yVel + zCoeff * zVel) * delta +
                (yCoeff * yAcc + zCoeff * zAcc) * oneHalfATSquared;
double dFY_dR = (cr * cpi * pitchVel - sr * cpi * yawVel) * delta;

xCoeff = -cp;
yCoeff = -sp * sr;
zCoeff = -sp * cr;
double dFz_dP = (xCoeff * xVel + yCoeff * yVel + zCoeff * zVel) * delta +
                (xCoeff * xAcc + yCoeff * yAcc + zCoeff * zAcc) * oneHalfATSquared;
double dFY_dP = (sr * tp * cpi * pitchVel + cr * tp * cpi * yawVel) * delta;
```

$$
\begin{aligned}
\frac{\partial x_k}{\partial \theta_x} &= (\cos\theta_z \sin\theta_y \cos\theta_x + \sin\theta_z \sin\theta_x)\Delta t v_y + \\
&\quad (-\cos\theta_z \sin\theta_y \sin\theta_x + \sin\theta_z \sin\theta_y)\Delta t v_z + \\
&\quad \frac{1}{2}(\cos\theta_z \sin\theta_y \cos\theta_x + \sin\theta_z \sin\theta_x)\Delta t \Delta t a_y + \\
&\quad \frac{1}{2}(-\cos\theta_z \sin\theta_y \sin\theta_x + \sin\theta_z \sin\theta_y \Delta t \Delta t a_z \\
&= [(\cos\theta_z \sin\theta_y \cos\theta_x + \sin\theta_z \sin\theta_x)\Delta t v_y + (-\cos\theta_z \sin\theta_y \sin\theta_x + \sin\theta_z \sin\theta_y)\Delta t v_z]\Delta t + \\
&\quad [(\cos\theta_z \sin\theta_y \cos\theta_x + \sin\theta_z \sin\theta_x)a_y + (-\cos\theta_z \sin\theta_y \sin\theta_x + \sin\theta_z \sin\theta_y)a_z]\frac{1}{2}\Delta t \Delta t
\end{aligned}
$$

- Sin과 Cos은 비선형이므로 테일러 1차를 통해 편미분을 하고, 거기서 나온 행렬을 자코비안으로 정의
- 편미분하여 자코비안을 준비

- 자코비안 함수 정의
- FB_DEBUG msg 출력 : 전달함수, 전달함수 자코비안, 프로세스 노이즈 공분산, 현재 상태

```
// Much of the transfer function Jacobian is identical to the transfer function
transferFunctionJacobian_ = transferFunction_;
transferFunctionJacobian_(StateMemberX, StateMemberRoll) = dFx_dR;
transferFunctionJacobian_(StateMemberX, StateMemberPitch) = dFx_dP;
transferFunctionJacobian_(StateMemberX, StateMemberYaw) = dFx_dY;
transferFunctionJacobian_(StateMemberY, StateMemberRoll) = dFy_dR;
transferFunctionJacobian_(StateMemberY, StateMemberPitch) = dFy_dP;
transferFunctionJacobian_(StateMemberY, StateMemberYaw) = dFy_dY;
transferFunctionJacobian_(StateMemberZ, StateMemberRoll) = dFz_dR;
transferFunctionJacobian_(StateMemberZ, StateMemberPitch) = dFz_dP;
transferFunctionJacobian_(StateMemberRoll, StateMemberRoll) = dFR_dR;
transferFunctionJacobian_(StateMemberRoll, StateMemberPitch) = dFR_dP;
transferFunctionJacobian_(StateMemberPitch, StateMemberRoll) = dFP_dR;
transferFunctionJacobian_(StateMemberYaw, StateMemberRoll) = dFY_dR;
transferFunctionJacobian_(StateMemberYaw, StateMemberPitch) = dFY_dP;

FB_DEBUG("Transfer function is:\n" << transferFunction_ <<
         "\nTransfer function Jacobian is:\n" << transferFunctionJacobian_ <<
         "\nProcess noise covariance is:\n" << processNoiseCovariance_ <<
         "\nCurrent state is:\n" << state_ << "\n");

Eigen::MatrixXd *processNoiseCovariance = &processNoiseCovariance_;
```

```
if (useDynamicProcessNoiseCovariance_)
{
  computeDynamicProcessNoiseCovariance(state_, delta);
  processNoiseCovariance = &dynamicProcessNoiseCovariance_;
}

// (1) Apply control terms, which are actually accelerations
state_(StateMemberVroll) += controlAcceleration_(ControlMemberVroll) * delta;
state_(StateMemberVpitch) += controlAcceleration_(ControlMemberVpitch) * delta;
state_(StateMemberVyaw) += controlAcceleration_(ControlMemberVyaw) * delta;

state_(StateMemberAx) = (controlUpdateVector_[ControlMemberVx] ?
  controlAcceleration_(ControlMemberVx) : state_(StateMemberAx));
state_(StateMemberAy) = (controlUpdateVector_[ControlMemberVy] ?
  controlAcceleration_(ControlMemberVy) : state_(StateMemberAy));
state_(StateMemberAz) = (controlUpdateVector_[ControlMemberVz] ?
  controlAcceleration_(ControlMemberVz) : state_(StateMemberAz));

// (2) Project the state forward: x = Ax + Bu (really, x = f(x, u))
state_ = transferFunction_ * state_;

// Handle wrapping
wrapStateAngles();

FB_DEBUG("Predicted state is:\n" << state_ <<
         "\nCurrent estimate error covariance is:\n" <<  estimateErrorCovariance_ << "\n");

// (3) Project the error forward: P = J * P * J' + Q
estimateErrorCovariance_ = (transferFunctionJacobian_ *
                            estimateErrorCovariance_ *
                            transferFunctionJacobian_.transpose());
estimateErrorCovariance_.noalias() += delta * (*processNoiseCovariance);

FB_DEBUG("Predicted estimate error covariance is:\n" << estimateErrorCovariance_ <<
         "\n\n--------------------- /Ekf::predict ----------------------\n");
}
```

1. 실제로 가속도인 제어 텀을 적용

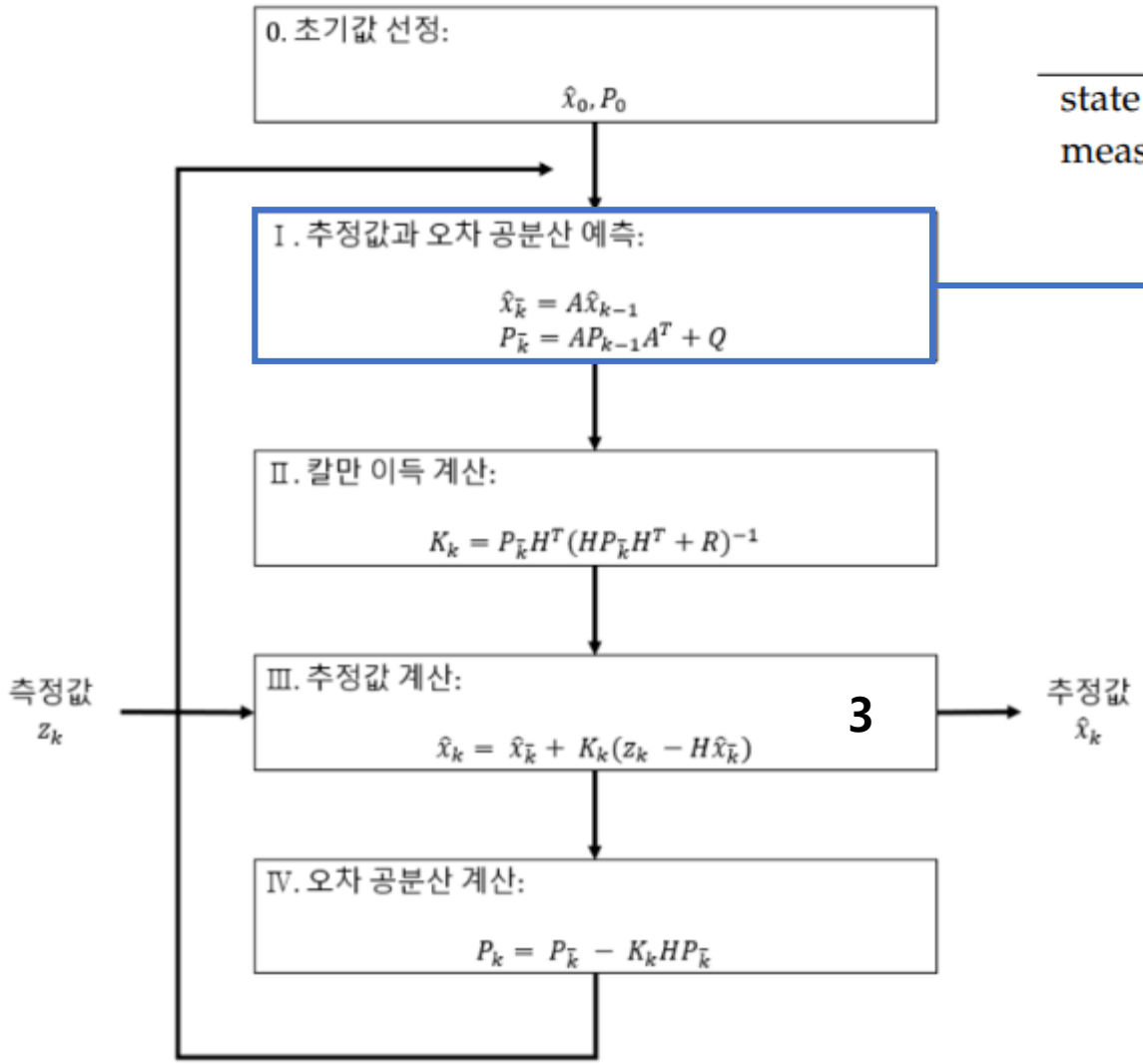
- 여기서 controlAcceleration_은 측정을 처리하고 유효한 컨트롤을 가질 때 마다 업데이트 되는 변수
- controlUpdateVector_는 사용 중인 제어 변수
- ControlMemberVx가 사용 중이라면,

StateMemberAx = controlAcceleration_(ControlMemberVx)

사용 X, StateMemberAx = state_(StateMemberAx)

2. 추정값 예측 : x = Ax + Bu (really, x = f(x, u))

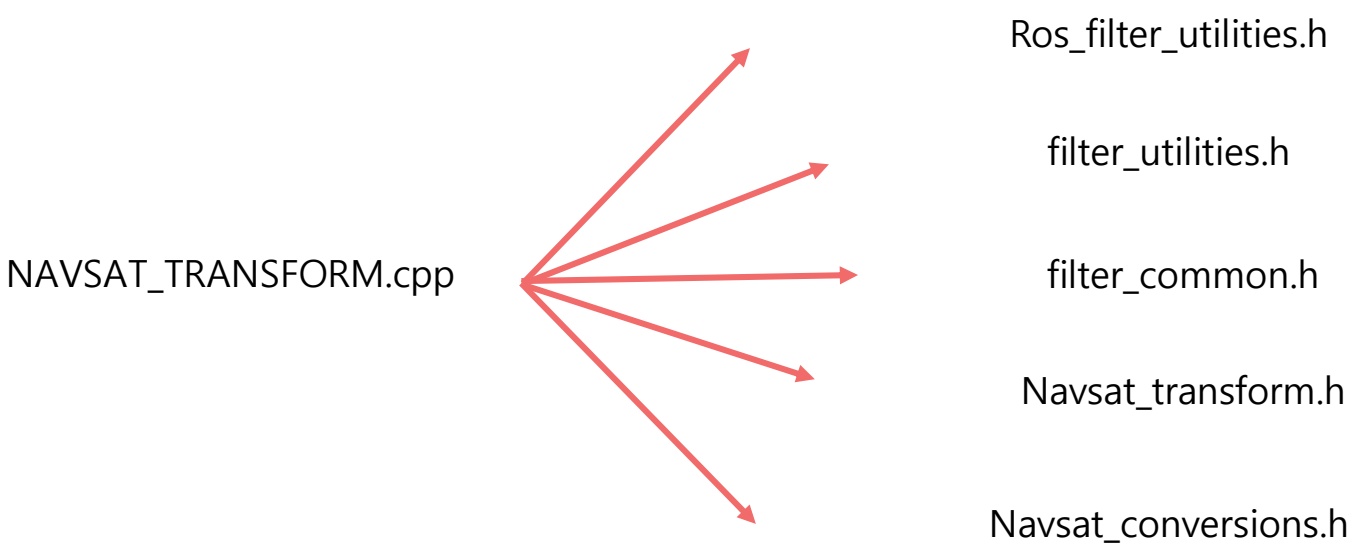
3. 공분산 예측 : P = J * P * J' + Q



	Kalman filter	EKF
state prediction (line 2)	$A_t\ \mu_{t-1} + B_t\ u_t$	$g(u_t, \mu_{t-1})$
measurement prediction (line 5)	$C_t\ \bar{\mu}_t$	$h(\bar{\mu}_t)$

Predict


NAVSAT_TRANSFORM
_NODELET.cpp
NAVSAT_TRANSF
ORM_NODE.cpp
navsat_transform.h


NAVSAT_TRANSFORM.cpp
Ros_filter_utilities.h
filter_utilities.h
filter_common.h
Navsat_transform.h
Navsat_conversions.h

Navsat_transform.h

Class NavSatTransform

Public :

- 생성자, 소멸자

Private :

- Datum/toLL/fromLL/setUTMzone callback
- Void getRobotOriginCartesianPose
- Void getRobotOriginWorldPose
- gpsFix/imu/odom callback
- prepareFilteredGps/prepareGpsOdometry
- setTransformGps/setTransformOdometry
- nav_msgs::Odometry cartesianToMap
  : 포즈를 utm에서 맵 프레임으로
- void mapToLL
  : 지점을 지도 프레임에서 위도/경도로
- odom/imu/gps 등 사용,게시,업뎃 여부 등
- GeographicLib::LocalCartesian gps_local_cartesian
- 공분산, 업데이트 타임, 타임 아웃, 오리엔테이션
- Pose
- Sub, pub, srv

Navsat_conversions.h

위도, 경도 > UTM 좌표 변환

- #define WGS84 Parameters : gps common reference
- #define UTM Parameters : gps common reference
- static inline void UTM : 측지 위치 > UTM
- static inline void LLtoUTM : 위도/경도 > UTM(easting, northing, UTM zone)
- static inline void UTMtoLLd : UTMUTM(easting, northing, UTM zone) > 위도/경도

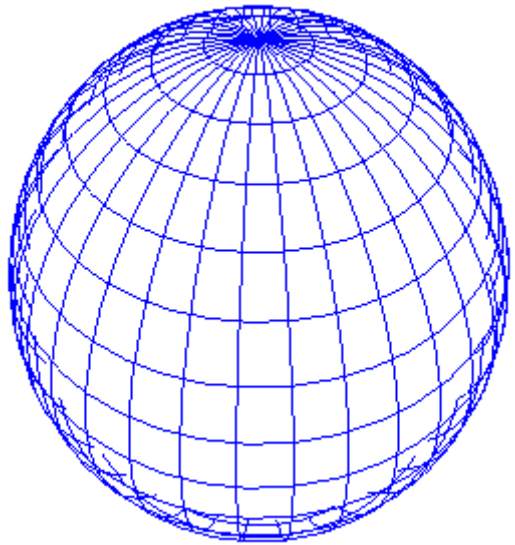


그림. Geodetic(측지위치=지리좌표) : 타원체 > 넓은 지역

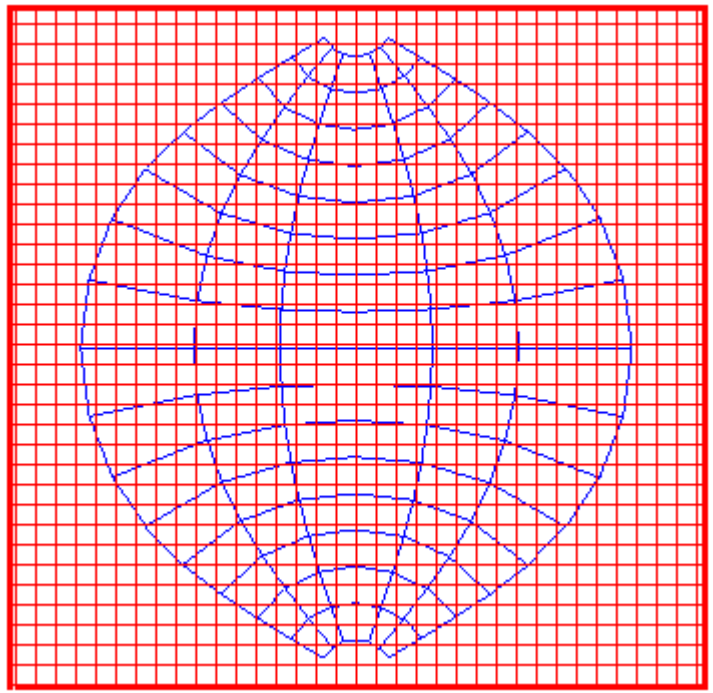


그림. UTM : 평면 직각 좌표계 > 좁은 지역 적합

Navsat_conversion.h

- WGS84와 UTM Parameter 설정 : Gps common에 정의되어 있음

```
#ifndef ROBOT_LOCALIZATION_NAVSAT_CONVERSIONS_H
#define ROBOT_LOCALIZATION_NAVSAT_CONVERSIONS_H

#include <cmath>
#include <string>

#include <stdio.h>
#include <stdlib.h>

#include <GeographicLib/MGRS.hpp>
#include <GeographicLib/UTMUPS.hpp>
namespace RobotLocalization
{
namespace NavsatConversions
{
const double RADIANS_PER_DEGREE = M_PI/180.0;
const double DEGREES_PER_RADIAN = 180.0/M_PI;

const double grid_size = 100000.0;

#define WGS84_A   6378137.0   // major axis
#define WGS84_B   6356752.31424518  // minor axis
#define WGS84_F   0.0033528107    // ellipsoid flattening
#define WGS84_E   0.0818191908    // first eccentricity
#define WGS84_EP  0.0820944379    // second eccentricity

//! @brief gps_common reference
// UTM Parameters
#define UTM_K0    0.9996              // scale factor
#define UTM_FE    500000.0            // false easting
#define UTM_FN_N  0.0                 // false northing, northern hemisphere
#define UTM_FN_S  10000000.0          // false northing, southern hemisphere
#define UTM_E2    (WGS84_E*WGS84_E)   // e^2
#define UTM_E4    (UTM_E2*UTM_E2)     // e^4
#define UTM_E6    (UTM_E4*UTM_E2)     // e^6
#define UTM_EP2   (UTM_E2/(1-UTM_E2))  // e'^2
```

```cpp
static inline void UTM(double lat, double lon, double *x, double *y)
{
  // constants
  static const double m0 = (1 - UTM_E2/4 - 3*UTM_E4/64 - 5*UTM_E6/256);
  static const double m1 = -(3*UTM_E2/8 + 3*UTM_E4/32 + 45*UTM_E6/1024);
  static const double m2 = (15*UTM_E4/256 + 45*UTM_E6/1024);
  static const double m3 = -(35*UTM_E6/3072);

  // compute the central meridian
  int cm = ((lon >= 0.0)
    ? (static_cast<int>(lon) - (static_cast<int>(lon)) % 6 + 3)
    : (static_cast<int>(lon) - (static_cast<int>(lon)) % 6 - 3));

  // convert degrees into radians
  double rlat = lat * RADIANS_PER_DEGREE;
  double rlon = lon * RADIANS_PER_DEGREE;
  double rlon0 = cm * RADIANS_PER_DEGREE;

  // compute trigonometric functions
  double slat = sin(rlat);
  double clat = cos(rlat);
  double tlat = tan(rlat);

  // decide the false northing at origin
  double fn = (lat > 0) ? UTM_FN_N : UTM_FN_S;

  double T = tlat * tlat;
  double C = UTM_EP2 * clat * clat;
  double A = (rlon - rlon0) * clat;
  double M = WGS84_A * (m0*rlat + m1*sin(2*rlat)
      + m2*sin(4*rlat) + m3*sin(6*rlat));
  double V = WGS84_A / sqrt(1 - UTM_E2*slat*slat);

  // compute the easting-northing coordinates
  *x = UTM_FE + UTM_K0 * V * (A + (1-T+C)*pow(A, 3)/6
             + (5-18*T+T*T+72*C-58*UTM_EP2)*pow(A, 5)/120);
  *y = fn + UTM_K0 * (M + V * tlat * (A*A/2
               + (5-T+9*C+4*C*C)*pow(A, 4)/24
               + ((61-58*T+T*T+600*C-330*UTM_EP2)
             * pow(A, 6)/720)));

  return;
}
```

- 측지 위치를 UTM 위치로 변환하는 유틸리티 기능 (3->2차원)

1. 상수 정의
2. 본초 자오선 계산 (UTM Zone은 경도 6도 간격)
3. 도 > 라디안
4. 삼각 함수 계산
5. 원점에서 false northing을 결정

- False easting : 지도 투영의 모든 x 좌표에 추가된 선형 값으로 매핑되는 지리적 영역의 값이 음수가 되지 않도록 함
- 음수 처리를 피하기 위해 각 구역의 중앙 자오선은 동쪽으로 500,000미터로 설정
- False nortning : 지도 투영의 모든 y 좌표에 추가된 선형 값으로 매핑되는 지리적 영역의 값이 음수가 되지 않도록 함
- 점이 북반구에 있는 경우 False Northing이 0인 반면 남반구의 한 지점은 10,000,000미터의 False Northing이 있음

6. easting-northing 좌표 계산

```cpp
static inline void LLtoUTM(const double Lat, const double Long,
                           double &UTMNorthing, double &UTMEasting,
                           std::string &UTMZone, double &gamma)
{
  int zone;
  bool northp;
  double k_unused;
  GeographicLib::UTMUPS::Forward(Lat, Long, zone, northp, UTMEasting, UTMNorthing, gamma,
                                 k_unused, GeographicLib::UTMUPS::zonespec::MATCH);
  GeographicLib::MGRS::Forward(zone, northp, UTMEasting, UTMNorthing, -1, UTMZone);
}

static inline void LLtoUTM(const double Lat, const double Long,
                           double &UTMNorthing, double &UTMEasting,
                           std::string &UTMZone)
{
  double gamma = 0.0;
  LLtoUTM(Lat, Long, UTMNorthing, UTMEasting, UTMZone, gamma);
}

static inline void UTMtoLL(const double UTMNorthing, const double UTMEasting,
                           const std::string &UTMZone, double& Lat, double& Long,
                           double& /*gamma*/)
{
  int zone;
  bool northp;
  double x_unused;
  double y_unused;
  int prec_unused;
  GeographicLib::MGRS::Reverse(UTMZone, zone, northp, x_unused, y_unused, prec_unused, true);
  GeographicLib::UTMUPS::Reverse(zone, northp, UTMEasting, UTMNorthing, Lat, Long);
}

static inline void UTMtoLL(const double UTMNorthing, const double UTMEasting,
                           const std::string &UTMZone, double& Lat, double& Long)
{
  double gamma;
  UTMtoLL(UTMNorthing, UTMEasting, UTMZone, Lat, Long, gamma);
}

}  // namespace NavsatConversions
}  // namespace RobotLocalization

#endif  // ROBOT_LOCALIZATION_NAVSAT_CONVERSIONS_H
```

- LL>UTM : 위도/경도를 UTM 좌표로 변환
- UTM>LL : UTM 좌표를 위도/경도로 변환

UTM.cpp

```cpp
#include "robot_localization/navsat_transform.h"
#include "robot_localization/filter_common.h"
#include "robot_localization/filter_utilities.h"
#include "robot_localization/navsat_conversions.h"
#include "robot_localization/ros_filter_utilities.h"

#include <tf2_geometry_msgs/tf2_geometry_msgs.h>
#include <XmlRpcException.h>

#include <string>

namespace RobotLocalization
{
  NavSatTransform::NavSatTransform(ros::NodeHandle nh, ros::NodeHandle nh_priv) :
    broadcast_cartesian_transform_(false),
    broadcast_cartesian_transform_as_parent_frame_(false),
    gps_updated_(false),
    has_transform_gps_(false),
    has_transform_imu_(false),
    has_transform_odom_(false),
    odom_updated_(false),
    publish_gps_(false),
    transform_good_(false),
    use_manual_datum_(false),
    use_odometry_yaw_(false),
    use_local_cartesian_(false),
    zero_altitude_(false),
    magnetic_declination_(0.0),
    yaw_offset_(0.0),
    base_link_frame_id_("base_link"),
    gps_frame_id_(""),
    utm_zone_(0),
    world_frame_id_("odom"),
    transform_timeout_(ros::Duration(0)),
    tf_listener_(tf_buffer_)
  {
    ROS_INFO("Waiting for valid clock time...");
    ros::Time::waitForValid();
    ROS_INFO("Valid clock time received. Starting node.");

    latest_cartesian_covariance_.resize(POSE_SIZE, POSE_SIZE);
    latest_odom_covariance_.resize(POSE_SIZE, POSE_SIZE);

    double frequency;
    double delay = 0.0;
    double transform_timeout = 0.0;
```

- navsat_transform의 NavSatTransform 클래스의 NavSatTransform의 값들 초기화
- 최신 GPS/UTM/LocalCartesian/odom 데이터에 대한 공분산 사이즈 resize
- 주파수, 딜레이, 변환 타임아웃 선언 및 초기화

- 필요한 parameter 로드 (private parameter로 접근)

  param : getparam과 비슷 but, parameter가 검색할 수 없는 경우 기본값을 지정 가능

- 더 이상 사용되지 않는 parameter 확인

```
// Load the parameters we need
nh_priv.getParam("magnetic_declination_radians", magnetic_declination_);
nh_priv.param("yaw_offset", yaw_offset_, 0.0);
nh_priv.param("broadcast_cartesian_transform", broadcast_cartesian_transform_, false);
nh_priv.param("broadcast_cartesian_transform_as_parent_frame",
              broadcast_cartesian_transform_as_parent_frame_, false);
nh_priv.param("zero_altitude", zero_altitude_, false);
nh_priv.param("publish_filtered_gps", publish_gps_, false);
nh_priv.param("use_odometry_yaw", use_odometry_yaw_, false);
nh_priv.param("wait_for_datum", use_manual_datum_, false);
nh_priv.param("use_local_cartesian", use_local_cartesian_, false);
nh_priv.param("frequency", frequency, 10.0);
nh_priv.param("delay", delay, 0.0);
nh_priv.param("transform_timeout", transform_timeout, 0.0);
nh_priv.param("cartesian_frame_id", cartesian_frame_id_, std::string(use_local_cartesian_ ? "local_enu" : "utm"));
transform_timeout_.fromSec(transform_timeout);

// Check for deprecated parameters
if (nh_priv.getParam("broadcast_utm_transform", broadcast_cartesian_transform_))
{
  ROS_WARN("navsat_transform, Parameter 'broadcast_utm_transform' has been deprecated. Please use"
           "'broadcast_cartesian_transform' instead.");
}
if (nh_priv.getParam("broadcast_utm_transform_as_parent_frame", broadcast_cartesian_transform_as_parent_frame_))
{
  ROS_WARN("navsat_transform, Parameter 'broadcast_utm_transform_as_parent_frame' has been deprecated. Please use"
           "'broadcast_cartesian_transform_as_parent_frame' instead.");
}
```

- tf 경고를 억제해야 하는지 확인

  nh.getparam은 parameter server에서 값을 가져옴

- 필요한 메시지와 서비스를 구독

  service(이름, 기능을 할 함수) : 요청시 작동

```
// Check if tf warnings should be suppressed
nh.getParam("/silent_tf_failure", tf_silent_failure_);

// Subscribe to the messages and services we need
datum_srv_ = nh.advertiseService("datum", &NavSatTransform::datumCallback, this);

to_ll_srv_ = nh.advertiseService("toLL", &NavSatTransform::toLLCallback, this);
from_ll_srv_ = nh.advertiseService("fromLL", &NavSatTransform::fromLLCallback, this);
set_utm_zone_srv_ = nh.advertiseService("setUTMZone", &NavSatTransform::setUTMZoneCallback, this);
```

- If, 첫 번째 GPS 메시지 또는 set_datum에서 데이텀을 가져오고 datum이라는 parameter를 확인한다면 srv 동작

  hasparam : parameter의 존재를 확인할 수 있음

- Datum : 지구타원체가 결정되고 나면, 지구타원체가 어떤 점을 중심으로 할지가 결정되어야하는데 그 기준점
- Ostringstream : 문자열 format을 조합하여 저장 / Istringstream : 문자열 포맷을 parsing

  즉, 읽은 datum_config[0~2]를 하나씩 쪼개서 datum_lat, lon, yaw로 parsing

ROS launch file에서 yaml file 을 load해주는 편리한 도구

예외 발생 가능성 있음

Asset : 어디에 에러가 났는지

Setprecision
부동 소수점에 대한 사용자 지정 정밀도 설정

```
if (use_manual_datum_ && nh_priv.hasParam("datum"))
{
  XmlRpc::XmlRpcValue datum_config;

  try
  {
    double datum_lat;
    double datum_lon;
    double datum_yaw;

    nh_priv.getParam("datum", datum_config);

    // Handle datum specification. Users should always specify a baseLinkFrameId_ in the
    // datum config, but we had a release where it wasn't used, so we'll maintain compatibility.
    ROS_ASSERT(datum_config.getType() == XmlRpc::XmlRpcValue::TypeArray);
    ROS_ASSERT(datum_config.size() >= 3);

    if (datum_config.size() > 3)
    {
      ROS_WARN_STREAM("Deprecated datum parameter configuration detected. Only the first three parameters "
          "(latitude, longitude, yaw) will be used. frame_ids will be derived from odometry and navsat inputs.");
    }

    std::ostringstream ostr;
    ostr << std::setprecision(20) << datum_config[0] << " " << datum_config[1] << " " << datum_config[2];
    std::istringstream istr(ostr.str());
    istr >> datum_lat >> datum_lon >> datum_yaw;
```

```
    // Try to resolve tf_prefix
    std::string tf_prefix = "";
    std::string tf_prefix_path = "";
    if (nh_priv.searchParam("tf_prefix", tf_prefix_path))
    {
      nh_priv.getParam(tf_prefix_path, tf_prefix);
    }

    // Append the tf prefix in a tf2-friendly manner
    FilterUtilities::appendPrefix(tf_prefix, world_frame_id_);
    FilterUtilities::appendPrefix(tf_prefix, base_link_frame_id_);

    robot_localization::SetDatum::Request request;
    request.geo_pose.position.latitude = datum_lat;
    request.geo_pose.position.longitude = datum_lon;
    request.geo_pose.position.altitude = 0.0;
    tf2::Quaternion quat;
    quat.setRPY(0.0, 0.0, datum_yaw);
    request.geo_pose.orientation = tf2::toMsg(quat);
    robot_localization::SetDatum::Response response;
    datumCallback(request, response);
  }
  catch (XmlRpc::XmlRpcException &e)
  {
    ROS_ERROR_STREAM("ERROR reading sensor config: " << e.getMessage() <<
                     " for process_noise_covariance (type: " << datum_config.getType() << ")");
  }
}

odom_sub_ = nh.subscribe("odometry/filtered", 1, &NavSatTransform::odomCallback, this);
gps_sub_  = nh.subscribe("gps/fix", 1, &NavSatTransform::gpsFixCallback, this);

if (!use_odometry_yaw_ && !use_manual_datum_)
{
  imu_sub_ = nh.subscribe("imu/data", 1, &NavSatTransform::imuCallback, this);
}

gps_odom_pub_ = nh.advertise<nav_msgs::Odometry>("odometry/gps", 10);

if (publish_gps_)
{
  filtered_gps_pub_ = nh.advertise<sensor_msgs::NavSatFix>("gps/filtered", 10);
}
```

예외 처리

- appendPrefix : 접두사 추가
- Bool 함수 datumCallback의 SetDatum에서 요청

1. 요청된 geo_pose의 position 위도 = datum_lat
2. 요청된 geo_pose의 position 경도 = datum_lon
3. 요청된 geo_pose의 position 고도 = 0.0

- Quat은 쿼터니안, quat의 RPY(0 0 datum_yaw) 세팅
- 요청된 geo_pose의 orientatio은 quat의 type으로 tf
- Bool 함수 datumCallback의 SetDatum에서 응답
- Datumcallback의 요청과 응답

- Odom, GPS subscriber 및 publish 정의
- 만약 첫 번째 GPS 메시지 또는 set_datum에서 데이터를 가져오고, odom or IMU 소스에서 yaw을 가져오지 않으면 imu sub
- 만약 GPS 메시지 pub하면 필터링된 GPS pub

주기적 업데이트 호출 타이머

```
  // Sleep for the parameterized amount of time, to give
  // other nodes time to start up (not always necessary)
  ros::Duration start_delay(delay);
  start_delay.sleep();

  periodicUpdateTimer_ = nh.createTimer(ros::Duration(1./frequency), &NavSatTransform::periodicUpdate, this);
}

NavSatTransform::~NavSatTransform()
{
}

//  void NavSatTransform::run()
void NavSatTransform::periodicUpdate(const ros::TimerEvent& event)
{
  if (!transform_good_)
  {
    computeTransform();

    if (transform_good_ && !use_odometry_yaw_ && !use_manual_datum_)
    {
      // Once we have the transform, we don't need the IMU
      imu_sub_.shutdown();
    }
  }
  else
  {
    nav_msgs::Odometry gps_odom;
    if (prepareGpsOdometry(gps_odom))
    {
      gps_odom_pub_.publish(gps_odom);
    }

    if (publish_gps_)
    {
      sensor_msgs::NavSatFix odom_gps;
      if (prepareFilteredGps(odom_gps))
      {
        filtered_gps_pub_.publish(odom_gps);
      }
    }
  }
}
```

- **주기적 업데이트를 위해 호출되는 콜백 함수**
- 방향 계산 실수시
- UTM 프레임에서 Odom 프레임으로의 변환을 계산

- 만약 방향 계산 굿, odom/imu에서 yaw 안 가져오고 첫 번째 GPS 메시지 또는 set_datum에서 데이텀을 안가져오면 Imu sub shut down

- 방향 계산 굿
- 만약, 전송하기 전에 GPS odom 메시지를 준비했다면 odometry/gps에 gps_odom 구독
- 만약, gps를 pub시 odom_gps을 msg출력

- 만약, odom 데이터를 다시 GPS로 변환하고 브로드캐스트했다면, 필터링된 GPS 데이터 게시자를 pub

- 다음과 같은 경우에만 이 작업을 수행

1. 이전에 odom_frame->cartesian_frame 변환을 계산하지 않았을 때

2. 필요한 데이터를 받았을 때 > if, 방향이 이상하고, 사용가능한 odom/gps/imu 수신했으면

- 데카르트 자세는 로봇의 GPS 센서 위치에 주어지므로 로봇 원점의 데카르트 자세를 가져와야 함
- 만약 gps msg 혹은 set_datum에서 데이텀 안가지고 오면 getRobotOriginCartesianPose
- gps msg 혹은 set_datum에서 데이텀 주어지면 > 변환된 데카르트 포즈 = 변환된 보정된 데카르트 포즈

즉, gps_cartesian_pose = robot_odom_pose

```
void NavSatTransform::computeTransform()
{
  // Only do this if:
  // 1. We haven't computed the odom_frame->cartesian_frame transform before
  // 2. We've received the data we need
  if (!transform_good_ &&
      has_transform_odom_ &&
      has_transform_gps_ &&
      has_transform_imu_)
  {
    // The cartesian pose we have is given at the location of the GPS sensor on the robot. We need to get the
    // cartesian pose of the robot's origin.
    tf2::Transform transform_cartesian_pose_corrected;
    if (!use_manual_datum_)
    { getRobotOriginCartesianPose(gps_cartesian_pose,       robot_odom_pose,                transform_time)
      getRobotOriginCartesianPose(transform_cartesian_pose_, transform_cartesian_pose_corrected, ros::Time(0));
    }
    else
    {
      transform_cartesian_pose_corrected = transform_cartesian_pose_;
    }
```

- 인간이 축에 대한 회전(=오리엔테이션)을 생각하기는 쉽지만 쿼터니언의 관점에서 생각하기는 어려움

  (X축에 대한 롤링) / (Y축에 대한 후속 피치) / (Z축에 대한 후속 요) 관점에서 대상 회전을 계산한 다음 쿼터니언으로 변환하는 것

- imu yaw = imu yaw + (magnetic declination + yaw 오프셋 + utm 자오선 수렴) > Rpy를 쿼터니안으로 변경

1. IMU는 데이터가 magnetic declination에 대해 수정되지 않았을 가능성이 있음

   북반구를 기준으로 지구 상의 현재 위치에서 진북극(지리상의 북극점) 방향과 자기북극 방향(나침반의 빨간 바늘이 가리키는 방향) 사이의 각도

2. IMU 중간 처리 노드에서 설명할 수 없는 다른 오프셋을 설명하기 위해 navsat_transform_node로 작업할 수 있도록 하는 yaw 오프셋을 노출

3. UTM 그리드가 True East/North와 정렬되지 않는 차이를 설명하기 위해 UTM을 사용할 때 자오선 수렴 각도를 추가

```
// Get the IMU's current RPY values. Need the raw values (for yaw, anyway).
tf2::Matrix3x3 mat(transform_orientation_);

// Convert to RPY
double imu_roll;
double imu_pitch;
double imu_yaw;
mat.getRPY(imu_roll, imu_pitch, imu_yaw);

imu_yaw += (magnetic_declination_ + yaw_offset_ + utm_meridian_convergence_);

ROS_INFO_STREAM("Corrected for magnetic declination of " << std::fixed << magnetic_declination_ <<
                ", user-specified offset of " << yaw_offset_ <<
                " and meridian convergence of " << utm_meridian_convergence_ << "." <<
                " Transform heading factor is now " << imu_yaw);

// Convert to tf-friendly structures
tf2::Quaternion imu_quat;
imu_quat.setRPY(0.0, 0.0, imu_yaw);
```

```
tf2::Transform cartesian_pose_with_orientation;
1 cartesian_pose_with_orientation.setOrigin(transform_cartesian_pose_corrected.getOrigin());
cartesian_pose_with_orientation.setRotation(imu_quat);

2 // Remove roll and pitch from odometry pose
// Must be done because roll and pitch is removed from cartesian_pose_with_orientation
double odom_roll, odom_pitch, odom_yaw;
3 tf2::Matrix3x3(transform_world_pose_.getRotation()).getRPY(odom_roll, odom_pitch, odom_yaw);
tf2::Quaternion odom_quat;
4 odom_quat.setRPY(0.0, 0.0, odom_yaw);
tf2::Transform transform_world_pose_yaw_only(transform_world_pose_);
transform_world_pose_yaw_only.setRotation(odom_quat);

5 cartesian_world_transform_.mult(transform_world_pose_yaw_only, cartesian_pose_with_orientation.inverse());

cartesian_world_trans_inverse_ = cartesian_world_transform_.inverse();

ROS_INFO_STREAM("Transform world frame pose is: " << transform_world_pose_);
ROS_INFO_STREAM("World frame->cartesian transform is " << cartesian_world_transform_);

transform_good_ = true;

// Send out the (static) Cartesian transform in case anyone else would like to use it.
if (broadcast_cartesian_transform_)
{
  geometry_msgs::TransformStamped cartesian_transform_stamped;
  cartesian_transform_stamped.header.stamp = ros::Time::now();
  cartesian_transform_stamped.header.frame_id = (broadcast_cartesian_transform_as_parent_frame_ ?
                                                 cartesian_frame_id_ : world_frame_id_);
  cartesian_transform_stamped.child_frame_id = (broadcast_cartesian_transform_as_parent_frame_ ?
                                                world_frame_id_ : cartesian_frame_id_);
  cartesian_transform_stamped.transform = (broadcast_cartesian_transform_as_parent_frame_ ?
                                           tf2::toMsg(cartesian_world_trans_inverse_) :
                                           tf2::toMsg(cartesian_world_transform_));
  cartesian_transform_stamped.transform.translation.z = (zero_altitude_ ?
                                                         0.0 : cartesian_transform_stamped.transform.translation.z);
  cartesian_broadcaster_.sendTransform(cartesian_transform_stamped);
}
```

- 1 보정된 데카르트 포즈(=로봇 원점 포즈)의 origin을 오리엔테이션 데카르트 포즈의 origin으로 설정
- 쿼터니안 값을 가지는 imu를 오리엔테이션 데카르트 포즈의 rotation으로 set
- 2 오도메트리 포즈에서 롤과 피치 제거
- cartesian_pose_with_orientation에서 롤과 피치가 제거되었으므로 반드시 수행 > imu quat으로 rot가짐
- 3 최신 IMU 오리엔테이션(transform_world_pose_)을 3x3행렬로 나타내고 odom_roll,pitch,yaw를 RPY로 갖고, new 로테이션을 함
- 4 odom_quat은 tf한 transform_world_pose_yaw_only의 new 로테이션으로 set
- 5 데카르트 월드 변환 = odom_quat X 쿼터니안 값을 가지는 imu의 inverse

```cpp
bool NavSatTransform::datumCallback(robot_localization::SetDatum::Request& request,
                                    robot_localization::SetDatum::Response&)
{
  // If we get a service call with a manual datum, even if we already computed the transform using the robot's
  // initial pose, then we want to assume that we are using a datum from now on, and we want other methods to
  // not attempt to transform the values we are specifying here.
  use_manual_datum_ = true;

  transform_good_ = false;

  sensor_msgs::NavSatFix *fix = new sensor_msgs::NavSatFix();
  fix->latitude = request.geo_pose.position.latitude;
  fix->longitude = request.geo_pose.position.longitude;
  fix->altitude = request.geo_pose.position.altitude;
  fix->header.stamp = ros::Time::now();
  fix->position_covariance[0] = 0.1;
  fix->position_covariance[4] = 0.1;
  fix->position_covariance[8] = 0.1;
  fix->position_covariance_type = sensor_msgs::NavSatStatus::STATUS_FIX;
  sensor_msgs::NavSatFixConstPtr fix_ptr(fix);
  setTransformGps(fix_ptr);

  nav_msgs::Odometry *odom = new nav_msgs::Odometry();
  odom->pose.pose.orientation.x = 0;
  odom->pose.pose.orientation.y = 0;
  odom->pose.pose.orientation.z = 0;
  odom->pose.pose.orientation.w = 1;
  odom->pose.pose.position.x = 0;
  odom->pose.pose.position.y = 0;
  odom->pose.pose.position.z = 0;
  odom->header.frame_id = world_frame_id_;
  odom->child_frame_id = base_link_frame_id_;
  nav_msgs::OdometryConstPtr odom_ptr(odom);
  setTransformOdometry(odom_ptr);

  sensor_msgs::Imu *imu = new sensor_msgs::Imu();
  imu->orientation = request.geo_pose.orientation;
  imu->header.frame_id = base_link_frame_id_;
  sensor_msgs::ImuConstPtr imu_ptr(imu);
  imuCallback(imu_ptr);

  return true;
}
```

- **데이텀 서비스에 대한 콜백 (요청, 응답)**
- 수동 데이텀으로 서비스 요청을 받으면 로봇의 초기 포즈를 사용하여 변환을 이미 계산했더라도, 우리는 지금부터 데이터를 사용하고 있다고 가정하고 여기에서 지정하는 값을 변환하지 않는 방법을 원함
- Fix 동적 메모리 할당, 포인터 역참조 메모리 접근
  - 위도, 경도, 고도
  - stamp, position 공분산, position 공분산 타입
- Odom 동적 메모리 할당, 포인터 역참조 메모리 접근
  - 오리엔테이션 : x, y, z, w / 위치 : x, y, z
  - Frame_id, child_frame_id
- imu 동적 메모리 할당, 포인터 역참조 메모리 접근
  - Imu 오리엔테이션 = geo_pose 오리엔테이션 요청
  - Imu frame_id=base_link_frame_id

```cpp
bool NavSatTransform::toLLCallback(robot_localization::ToLL::Request& request,
                                   robot_localization::ToLL::Response& response)
{
  if (!transform_good_)
  {
    ROS_ERROR("No transform available (yet)");
    return false;
  }
  tf2::Vector3 point;
  tf2::fromMsg(request.map_point, point);
  mapToLL(point, response.ll_point.latitude, response.ll_point.longitude, response.ll_point.altitude);

  return true;
}

bool NavSatTransform::fromLLCallback(robot_localization::FromLL::Request& request,
                                     robot_localization::FromLL::Response& response)
{
  double altitude = request.ll_point.altitude;
  double longitude = request.ll_point.longitude;
  double latitude = request.ll_point.latitude;

  tf2::Transform cartesian_pose;

  double cartesian_x;
  double cartesian_y;
  double cartesian_z;

  if (use_local_cartesian_)
  {
    gps_local_cartesian_.Forward(latitude, longitude, altitude, cartesian_x, cartesian_y, cartesian_z);
  }
  else
  {
    int zone_tmp;
    bool nortp_tmp;
    try
    {
      GeographicLib::UTMUPS::Forward(latitude, longitude, zone_tmp, nortp_tmp, cartesian_x, cartesian_y, utm_zone_);
    }
    catch (const GeographicLib::GeographicErr& e)
    {
      ROS_ERROR_STREAM_THROTTLE(1.0, e.what());
      return false;
    }
  }
```

- **toLLCallback : 위도, 경도 서비스에 대한 콜백**
- Ros type request.map_point 를 another type인 point로
- mapToLL 함수 사용

- **fromLLCallback : 위도, 경도로 부터의 서비스에 대한 콜백**
- 만약, UTM 좌표를 데카르트 좌표로 사용한다면

UTM에서 데카르트 좌표 사용하기 위해 projection (3차원>2차원)

- UTM 좌표를 데카르트 좌표로 사용하지 않는다면

Try 예외 발생 가능성 : GeographicLib::UTMUPS

지리적 좌표를 UTM으로 변환

Catch(const GeographicLib::GeographicErr& e)

:()괄호 안 발생시, Error를 logger에 기록

```
  cartesian_pose.setOrigin(tf2::Vector3(cartesian_x, cartesian_y, altitude));

  nav_msgs::Odometry gps_odom;

  if (!transform_good_)
  {
    ROS_ERROR("No transform available (yet)");
    return false;
  }

  response.map_point = cartesianToMap(cartesian_pose).pose.pose.position;

  return true;
}

bool NavSatTransform::setUTMZoneCallback(robot_localization::SetUTMZone::Request& request,
                                         robot_localization::SetUTMZone::Response& response)
{
  double x_unused;
  double y_unused;
  int prec_unused;
  GeographicLib::MGRS::Reverse(request.utm_zone, utm_zone_, northp_, x_unused, y_unused, prec_unused, true);
  ROS_INFO("UTM zone set to %d %s", utm_zone_, northp_ ? "north" : "south");
  return true;
}

nav_msgs::Odometry NavSatTransform::cartesianToMap(const tf2::Transform& cartesian_pose) const
{
  nav_msgs::Odometry gps_odom{};

  tf2::Transform transformed_cartesian_gps{};

  transformed_cartesian_gps.mult(cartesian_world_transform_, cartesian_pose);
  transformed_cartesian_gps.setRotation(tf2::Quaternion::getIdentity());

  // Set header information stamp because we would like to know the robot's position at that timestamp
  gps_odom.header.frame_id = world_frame_id_;
  gps_odom.header.stamp = gps_update_time_;

  // Now fill out the message. Set the orientation to the identity.
  tf2::toMsg(transformed_cartesian_gps, gps_odom.pose.pose);
  gps_odom.pose.pose.position.z = (zero_altitude_ ? 0.0 : gps_odom.pose.pose.position.z);

  return gps_odom;
}
```

- 데카르트 좌표 origin설정 (벡터 데카르트 x,y,고도)
- map point 응답 = 데카르트to맵 함수(데카르트 포즈를 변수로 받는)의 position

- **UTM 영역 서비스에 대한 콜백**
- MGRS좌표를 UTM좌표로 반환하는 영역으로 영역, 북반구, 동쪽 x(미터), 북쪽 y(미터)로 변환

- **전달된 데카르트 포즈를 utm에서 맵 프레임으로 변환**
- 변환된 데카르트 gps

= 데카르트->odom 변환 유지 X 데카르트 포즈

- 변환된 데카르트 gps는 쿼터니안 형식의 항등항렬을 Rotation으로 세팅

항등행렬 : 주대각선의 원소가 모두 1, 나머지 원소는 0인 정사각 행렬

```cpp
void NavSatTransform::mapToLL(const tf2::Vector3& point, double& latitude, double& longitude, double& altitude) const
{
  tf2::Transform odom_as_cartesian{};

  tf2::Transform pose{};
  pose.setOrigin(point);
  pose.setRotation(tf2::Quaternion::getIdentity());

  odom_as_cartesian.mult(cartesian_world_trans_inverse_, pose);
  odom_as_cartesian.setRotation(tf2::Quaternion::getIdentity());

  if (use_local_cartesian_)
  {
    double altitude_tmp = 0.0;
    gps_local_cartesian_.Reverse(odom_as_cartesian.getOrigin().getX(),
                                 odom_as_cartesian.getOrigin().getY(),
                                 0.0,
                                 latitude,
                                 longitude,
                                 altitude_tmp);
    altitude = odom_as_cartesian.getOrigin().getZ();
  }
  else
  {
    GeographicLib::UTMUPS::Reverse(utm_zone_,
                                   northp_,
                                   odom_as_cartesian.getOrigin().getX(),
                                   odom_as_cartesian.getOrigin().getY(),
                                   latitude,
                                   longitude);
    altitude = odom_as_cartesian.getOrigin().getZ();
  }
}
```

- **지도 프레임에서 위도/경도로 변환**
- 데카르트에서 odom , pose TF
- Pose에서 쿼터니안 항등행렬을 rotation set
- 데카르트에서 odom

= 필터링된 GPS 브로드캐스트에 대한

odom->UTM x pose

- UTM 좌표를 데카르트 좌표로 사용한다면

로컬 데카르트 x , y , z (미터)에서 측지 좌표 lat , lon (도), h (미터)로 변환

- UTM 좌표를 데카르트 좌표로 사용 X

UTM 좌표를 지리적 좌표로 변환 utmzone, 북반구, x,y,위도 경도를 변수로

```cpp
void NavSatTransform::getRobotOriginCartesianPose(const tf2::Transform &gps_cartesian_pose,
                                                  tf2::Transform &robot_cartesian_pose,
                                                  const ros::Time &transform_time)
{
  robot_cartesian_pose.setIdentity();

  // Get linear offset from origin for the GPS
  tf2::Transform offset;
  bool can_transform = RosFilterUtilities::lookupTransformSafe(tf_buffer_,
                                                               base_link_frame_id_,
                                                               gps_frame_id_,
                                                               transform_time,
                                                               ros::Duration(transform_timeout_),
                                                               offset,
                                                               tf_silent_failure_);

  if (can_transform)
  {
    // Get the orientation we'll use for our Cartesian->world transform
    tf2::Quaternion cartesian_orientation = transform_orientation_;
    tf2::Matrix3x3 mat(cartesian_orientation);

    // Add the offsets
    double roll;
    double pitch;
    double yaw;
    mat.getRPY(roll, pitch, yaw);
    yaw += (magnetic_declination_ + yaw_offset_ + utm_meridian_convergence_);
    cartesian_orientation.setRPY(roll, pitch, yaw);

    // Rotate the GPS linear offset by the orientation
    // Zero out the orientation, because the GPS orientation is meaningless, and if it's non-zero, it will make the
    // the computation of robot_cartesian_pose erroneous.
    offset.setOrigin(tf2::quatRotate(cartesian_orientation, offset.getOrigin()));
    offset.setRotation(tf2::Quaternion::getIdentity());

    // Update the initial pose
    robot_cartesian_pose = offset.inverse() * gps_cartesian_pose;
  }
  else
  {
    if (gps_frame_id_ != "")
    {
      ROS_WARN_STREAM_ONCE("Unable to obtain " << base_link_frame_id_ << "->" << gps_frame_id_ <<
        " transform. Will assume navsat device is mounted at robot's origin");
    }

    robot_cartesian_pose = gps_cartesian_pose;
  }
}
```

- **월드 프레임에서 navsat 센서의 포즈가 주어지면 차량의 중심에서 오프셋을 제거하고 해당 중심의 '데카르트' 프레임 포즈를 반환**

- gps frame id > base link frame id 변환 얻을려고 함
  실패시 tf_silent_failure_ 얻음
- TF 성공시) robot_cartesian_pose = offset.inverse() * gps_cartesian_pose;
- TF 실패시) 로봇 데카르트 포즈 = gps 데카르트 포즈

```cpp
void NavSatTransform::getRobotOriginWorldPose(const tf2::Transform &gps_odom_pose,
                                              tf2::Transform &robot_odom_pose,
                                              const ros::Time &transform_time)
{
  robot_odom_pose.setIdentity();

  // Remove the offset from base_link
  tf2::Transform gps_offset_rotated;
  bool can_transform = RosFilterUtilities::lookupTransformSafe(tf_buffer_,
                                                               base_link_frame_id_,
                                                               gps_frame_id_,
                                                               transform_time,
                                                               transform_timeout_,
                                                               gps_offset_rotated,
                                                               tf_silent_failure_);

  if (can_transform)
  {
    tf2::Transform robot_orientation;
    can_transform = RosFilterUtilities::lookupTransformSafe(tf_buffer_,
                                                            world_frame_id_,
                                                            base_link_frame_id_,
                                                            transform_time,
                                                            transform_timeout_,
                                                            robot_orientation,
                                                            tf_silent_failure_);

    if (can_transform)
    {
      // Zero out rotation because we don't care about the orientation of the
      // GPS receiver relative to base_link
      gps_offset_rotated.setOrigin(tf2::quatRotate(robot_orientation.getRotation(), gps_offset_rotated.getOrigin()));
      gps_offset_rotated.setRotation(tf2::Quaternion::getIdentity());
      robot_odom_pose = gps_offset_rotated.inverse() * gps_odom_pose;
    }
    else
    {
      ROS_WARN_STREAM_THROTTLE(5.0, "Could not obtain " << world_frame_id_ << "->" << base_link_frame_id_ <<
        " transform. Will not remove offset of navsat device from robot's origin.");
    }
  }
  else
  {
    ROS_WARN_STREAM_THROTTLE(5.0, "Could not obtain " << base_link_frame_id_ << "->" << gps_frame_id_ <<
      " transform. Will not remove offset of navsat device from robot's origin.");
  }
}
```

- **월드 프레임에서 navsat 센서의 포즈가 주어지면 차량의 중심에서 오프셋을 제거하고 해당 중심의 '월드' 프레임 포즈를 반환**

- gps frame id > base link frame id 변환 얻을려고 함
  실패시 tf_silent_failure_ 얻음

{

world_frame_id > base_link_frame_id 변환 얻을려고 함
  실패시 tf_silent_failure_ 얻음

robot_odom_pose = gps_offset_rotated.inverse() *
gps_odom_pose;

}

- **gps 수정 데이터에 대한 콜백**
- Bool Good gps (상태 fix X, 위도 경도 고도가 nan이 아님)
- Good gps일 때, 만약 데이터를 사용할 수 있다면 GPS 원점 주변의 로컬 데카르트 투영법
- Good gps가 아닐 때, UTM 좌표를 데카르트 좌표로 사용하지 않으면 고정 utm_zone_을 사용하여 UTM으로 변환
- 이후, 데카르트 좌표로 저장된 최신 GPS 데이터의 origin은(벡터형 데카르트 x,y,고도)로 set
- 최신 데카르트 좌표로 저장된 최신 GPS 데이터의 공분산은 zero로 set

```cpp
void NavSatTransform::gpsFixCallback(const sensor_msgs::NavSatFixConstPtr& msg)
{
  gps_frame_id_ = msg->header.frame_id;

  if (gps_frame_id_.empty())
  {
    ROS_WARN_STREAM_ONCE("NavSatFix message has empty frame_id. Will assume navsat device is mounted at robot's "
      "origin.");
  }

  // Make sure the GPS data is usable
  bool good_gps = (msg->status.status != sensor_msgs::NavSatStatus::STATUS_NO_FIX &&
                   !std::isnan(msg->altitude) &&
                   !std::isnan(msg->latitude) &&
                   !std::isnan(msg->longitude));

  if (good_gps)
  {
    // If we haven't computed the transform yet, then
    // store this message as the initial GPS data to use
    if (!transform_good_ && !use_manual_datum_)
    {
      setTransformGps(msg);
    }

    double cartesian_x = 0.0;
    double cartesian_y = 0.0;
    double cartesian_z = 0.0;
    if (use_local_cartesian_)
    {
      gps_local_cartesian_.Forward(msg->latitude, msg->longitude, msg->altitude,
                                   cartesian_x, cartesian_y, cartesian_z);
    }
    else
    {
      // Transform to UTM using the fixed utm_zone_
      int zone_tmp;
      bool northp_tmp;
      try
      {
        GeographicLib::UTMUPS::Forward(msg->latitude, msg->longitude,
                                       zone_tmp, northp_tmp, cartesian_x, cartesian_y, utm_zone_);
      }
      catch (const GeographicLib::GeographicErr& e)
      {
        ROS_ERROR_STREAM_THROTTLE(1.0, e.what());
        return;
      }
    }
    latest_cartesian_pose_.setOrigin(tf2::Vector3(cartesian_x, cartesian_y, msg->altitude));
    latest_cartesian_covariance_.setZero();

    // Copy the measurement's covariance matrix so that we can rotate it later
    for (size_t i = 0; i < POSITION_SIZE; i++)
    {
      for (size_t j = 0; j < POSITION_SIZE; j++)
      {
        latest_cartesian_covariance_(i, j) = msg->position_covariance[POSITION_SIZE * i + j];
      }
    }

    gps_update_time_ = msg->header.stamp;
    gps_updated_ = true;
  }
}
```

```cpp
void NavSatTransform::imuCallback(const sensor_msgs::ImuConstPtr& msg)
{
  // We need the baseLinkFrameId_ from the odometry message, so
  // we need to wait until we receive it.
  if (has_transform_odom_)
  {
    /* This method only gets called if we don't yet have the
     * IMU data (the subscriber gets shut down once we compute
     * the transform), so we can assumed that every IMU message
     * that comes here is meant to be used for that purpose. */
    tf2::fromMsg(msg->orientation, transform_orientation_);

    // Correct for the IMU's orientation w.r.t. base_link
    tf2::Transform target_frame_trans;
    bool can_transform = RosFilterUtilities::lookupTransformSafe(tf_buffer_,
                                                                 base_link_frame_id_,
                                                                 msg->header.frame_id,
                                                                 msg->header.stamp,
                                                                 transform_timeout_,
                                                                 target_frame_trans,
                                                                 tf_silent_failure_);

    if (can_transform)
    {
      double roll_offset = 0;
      double pitch_offset = 0;
      double yaw_offset = 0;
      double roll = 0;
      double pitch = 0;
      double yaw = 0;
      RosFilterUtilities::quatToRPY(target_frame_trans.getRotation(), roll_offset, pitch_offset, yaw_offset);
      RosFilterUtilities::quatToRPY(transform_orientation_, roll, pitch, yaw);

      ROS_DEBUG_STREAM("Initial orientation is " << transform_orientation_);

      // Apply the offset (making sure to bound them), and throw them in a vector
      tf2::Vector3 rpy_angles(FilterUtilities::clampRotation(roll - roll_offset),
                              FilterUtilities::clampRotation(pitch - pitch_offset),
                              FilterUtilities::clampRotation(yaw - yaw_offset));

      tf2::Matrix3x3 mat;
      mat.setRPY(0.0, 0.0, yaw_offset);
      rpy_angles = mat * rpy_angles;
      transform_orientation_.setRPY(rpy_angles.getX(), rpy_angles.getY(), rpy_angles.getZ());

      ROS_DEBUG_STREAM("Initial corrected orientation roll, pitch, yaw is (" <<
                       rpy_angles.getX() << ", " << rpy_angles.getY() << ", " << rpy_angles.getZ() << ")");

      has_transform_imu_ = true;
    }
  }
}
```

- **IMU 데이터에 대한 콜백 :아직 IMU 데이터가 없는 경우에만 호출**
- 즉 header.frame_id > base_link_frame_id_ 변환 얻을려고 함

  실패시 tf_silent_failure_ 얻음
- tf할 수 있으면 {

input : quat > 변환할 쿼터니언

output : r,p,y > 변환된 r,p,y

- 함수 : 쿼터니안, r, p, y를 멤버변수로 받는 quatToRPY 멤버 함수

1. r,p,y offset 멤버 변수로 받아 target_frame_trans(imu 오리엔테이션)을 변환하여 쿼터니언으로 얻음

2. r,p,y 멤버 변수로 받아 transform_orientation(최신 imu오리엔테이션)을 변환하여 쿼터니언으로 얻음

- 이와 같이 rpy_angle은 로테이션 범위를 정해줌
- 이제 요 오프셋 값으로 롤과 피치를 회전해야 함
- IMU가 옆을 향하여 장착될 때, IMU의 세계 프레임에 대한 피치는 로봇에 대한 롤 }

```cpp
void NavSatTransform::odomCallback(const nav_msgs::OdometryConstPtr& msg)
{
  world_frame_id_ = msg->header.frame_id;
  base_link_frame_id_ = msg->child_frame_id;

  if (!transform_good_ && !use_manual_datum_)
  {
    setTransformOdometry(msg);
  }

  tf2::fromMsg(msg->pose.pose, latest_world_pose_);
  latest_odom_covariance_.setZero();
  for (size_t row = 0; row < POSE_SIZE; ++row)
  {
    for (size_t col = 0; col < POSE_SIZE; ++col)
    {
      latest_odom_covariance_(row, col) = msg->pose.covariance[row * POSE_SIZE + col];
    }
  }

  odom_update_time_ = msg->header.stamp;
  odom_updated_ = true;
}
```

- **odom 데이터에 대한 콜백**

: world frame id가 header frame id

: base link frame id가 child frame id

latest_odom_covariance_(row, col) = msg->

pose.covariance[row * POSE_SIZE + col]

```cpp
bool NavSatTransform::prepareFilteredGps(sensor_msgs::NavSatFix &filtered_gps)
{
  bool new_data = false;

  if (transform_good_ && odom_updated_)
  {
    mapToLL(latest_world_pose_.getOrigin(), filtered_gps.latitude, filtered_gps.longitude, filtered_gps.altitude);

    // Rotate the covariance as well
    tf2::Matrix3x3 rot(cartesian_world_trans_inverse_.getRotation());
    Eigen::MatrixXd rot_6d(POSE_SIZE, POSE_SIZE);
    rot_6d.setIdentity();

    for (size_t rInd = 0; rInd < POSITION_SIZE; ++rInd)
    {
      rot_6d(rInd, 0) = rot.getRow(rInd).getX();
      rot_6d(rInd, 1) = rot.getRow(rInd).getY();
      rot_6d(rInd, 2) = rot.getRow(rInd).getZ();
      rot_6d(rInd+POSITION_SIZE, 3) = rot.getRow(rInd).getX();
      rot_6d(rInd+POSITION_SIZE, 4) = rot.getRow(rInd).getY();
      rot_6d(rInd+POSITION_SIZE, 5) = rot.getRow(rInd).getZ();
    }

    // Rotate the covariance
    latest_odom_covariance_ = rot_6d * latest_odom_covariance_.eval() * rot_6d.transpose();

    // Copy the measurement's covariance matrix back
    for (size_t i = 0; i < POSITION_SIZE; i++)
    {
      for (size_t j = 0; j < POSITION_SIZE; j++)
      {
        filtered_gps.position_covariance[POSITION_SIZE * i + j] = latest_odom_covariance_(i, j);
      }
    }

    filtered_gps.position_covariance_type = sensor_msgs::NavSatFix::COVARIANCE_TYPE_KNOWN;
    filtered_gps.status.status = sensor_msgs::NavSatStatus::STATUS_GBAS_FIX;
    filtered_gps.header.frame_id = base_link_frame_id_;
    filtered_gps.header.stamp = odom_update_time_;

    // Mark this GPS as used
    odom_updated_ = false;
    new_data = true;
  }

  return new_data;
}
```

Set the default build target

- **필터링된 GPS : odom 데이터를 다시 GPS로 변환하고 브로드캐스트**
- 좋은 방향을 계산했고 새로운 odom 데이터가 있으면,
- 전달된 지점을 지도 프레임에서 위도/경도로 변환,
- 여기서 변환될 map point는 latest_world_pose_.getOrigin()임

- 공분산도 회전 > 정의

최신 오돔 공분산 = 항등행렬 rot_6d * 최신 odom 공분산(문자로 표현된 코드 실행) * rot_6d의 전치행렬

- Filtered_gps 정의

```cpp
bool NavSatTransform::prepareGpsOdometry(nav_msgs::Odometry &gps_odom)
{
  bool new_data = false;

  if (transform_good_ && gps_updated_ && odom_updated_)
  {
    gps_odom = cartesianToMap(latest_cartesian_pose_);

    tf2::Transform transformed_cartesian_gps;
    tf2::fromMsg(gps_odom.pose.pose, transformed_cartesian_gps);

    // Want the pose of the vehicle origin, not the GPS
    tf2::Transform transformed_cartesian_robot;
    getRobotOriginWorldPose(transformed_cartesian_gps, transformed_cartesian_robot, gps_odom.header.stamp);

    // Rotate the covariance as well
    tf2::Matrix3x3 rot(cartesian_world_transform_.getRotation());
    Eigen::MatrixXd rot_6d(POSE_SIZE, POSE_SIZE);
    rot_6d.setIdentity();

    for (size_t rInd = 0; rInd < POSITION_SIZE; ++rInd)
    {
      rot_6d(rInd, 0) = rot.getRow(rInd).getX();
      rot_6d(rInd, 1) = rot.getRow(rInd).getY();
      rot_6d(rInd, 2) = rot.getRow(rInd).getZ();
      rot_6d(rInd+POSITION_SIZE, 3) = rot.getRow(rInd).getX();
      rot_6d(rInd+POSITION_SIZE, 4) = rot.getRow(rInd).getY();
      rot_6d(rInd+POSITION_SIZE, 5) = rot.getRow(rInd).getZ();
    }

    // Rotate the covariance
    latest_cartesian_covariance_ = rot_6d * latest_cartesian_covariance_.eval() * rot_6d.transpose();

    // Now fill out the message. Set the orientation to the identity.
    tf2::toMsg(transformed_cartesian_robot, gps_odom.pose.pose);
    gps_odom.pose.pose.position.z = (zero_altitude_ ? 0.0 : gps_odom.pose.pose.position.z);

    // Copy the measurement's covariance matrix so that we can rotate it later
    for (size_t i = 0; i < POSE_SIZE; i++)
    {
      for (size_t j = 0; j < POSE_SIZE; j++)
      {
        gps_odom.pose.covariance[POSE_SIZE * i + j] = latest_cartesian_covariance_(i, j);
      }
    }

    // Mark this GPS as used
    gps_updated_ = false;
    new_data = true;
  }

  return new_data;
}
```

- **전송하기 전에 GPS odom 메시지 준비**

- 좋은 방향을 계산했고 새로운 gps/odom 있으면,
- gps_odom=변환에 사용할 데카르트 좌표계의 포즈를 utm 에서 맵 프레임으로 변환
- GPS가 아닌 차량 원점의 pose를 원하기 때문에, getRobotOriginWorldPose을 통해 원점 pose구함

- 공분산도 회전 > 정의
- 최신 데카르트 공분산 = 항등행렬 rot_6d * 최신 데카르트 공분산(문자로 표현된 코드 실행) * rot_6d의 전치행렬

```cpp
void NavSatTransform::setTransformGps(const sensor_msgs::NavSatFixConstPtr& msg)
{
  double cartesian_x = 0;
  double cartesian_y = 0;
  double cartesian_z = 0;
  if (use_local_cartesian_)
  {
    const double hae_altitude = 0.0;
    gps_local_cartesian_.Reset(msg->latitude, msg->longitude, hae_altitude);
    gps_local_cartesian_.Forward(msg->latitude, msg->longitude, msg->altitude, cartesian_x, cartesian_y, cartesian_z);

    // UTM meridian convergence is not meaningful when using local cartesian, so set it to 0.0
    utm_meridian_convergence_ = 0.0;
  }
  else
  {
    double k_tmp;
    double utm_meridian_convergence_degrees;
    GeographicLib::UTMUPS::Forward(msg->latitude, msg->longitude, utm_zone_, northp_,
                                   cartesian_x, cartesian_y, utm_meridian_convergence_degrees, k_tmp);
    utm_meridian_convergence_ = utm_meridian_convergence_degrees * NavsatConversions::RADIANS_PER_DEGREE;
  }

  ROS_INFO_STREAM("Datum (latitude, longitude, altitude) is (" << std::fixed << msg->latitude << ", " <<
                  msg->longitude << ", " << msg->altitude << ")");
  ROS_INFO_STREAM("Datum " << ((use_local_cartesian_)? "Local Cartesian" : "UTM") <<
                  " coordinate is (" << std::fixed << cartesian_x << ", " << cartesian_y << ") zone " << utm_zone_);

  transform_cartesian_pose_.setOrigin(tf2::Vector3(cartesian_x, cartesian_y, msg->altitude));
  transform_cartesian_pose_.setRotation(tf2::Quaternion::getIdentity());
  has_transform_gps_ = true;
}
```

- **변환을 계산하는 데 사용할 GPS 데이터를 설정하는 데 사용**

- UTM 좌표를 데카르트 좌표로 사용한다면,

UTM 자오선 수렴은 지역 데카르트를 사용할 때 의미가 없으므로 0.0으로 설정

- UTM 좌표를 데카르트 좌표로 사용 X,

utm 자오선 수렴 = utm 자오선 수렴 각도 * 각도당 라디안

```cpp
void NavSatTransform::setTransformOdometry(const nav_msgs::OdometryConstPtr& msg)
{
  tf2::fromMsg(msg->pose.pose, transform_world_pose_);
  has_transform_odom_ = true;

  ROS_INFO_STREAM_ONCE("Initial odometry pose is " << transform_world_pose_);

  // Users can optionally use the (potentially fused) heading from
  // the odometry source, which may have multiple fused sources of
  // heading data, and so would act as a better heading for the
  // Cartesian->world_frame transform.
  if (!transform_good_ && use_odometry_yaw_ && !use_manual_datum_)
  {
    sensor_msgs::Imu *imu = new sensor_msgs::Imu();
    imu->orientation = msg->pose.pose.orientation;
    imu->header.frame_id = msg->child_frame_id;
    imu->header.stamp = msg->header.stamp;
    sensor_msgs::ImuConstPtr imuPtr(imu);
    imuCallback(imuPtr);
  }
}

}  // namespace RobotLocalization
```

- **변환을 계산하는 데 사용할 odom 데이터를 설정하는 데 사용**
- 사용자는 선택적으로 odom 소스에서 (잠재적으로 융합된) 헤딩(방향)을 사용할 수 있음
- 이 방향은 여러 통합된 방향 데이터 소스를 가질 수 있으므로 Cartesian->world_frame 변환에 대한 더 나은 방향 역할을 함