IMU, GNSS and camera sensor fusion algorithm

Silvio Cardo

June 28, 2024

In the following it is described a concept of a sensor fusion system, considering data coming from IMU, GNSS and camera to estimate the position and orientation of a vehicle.

Introduction

We consider a 6 d.o.f IMU providing 3-axis accelerations and 3-axis angular velocities, a single antenna GNSS receiver providing latitude and longitude position information and a front camera providing images of the environment. The proposed solution is structured as follows:

1. firstly a calibration phase is performed in order to evaluate IMU data offsets and camera intrinsic and extrinsic parameters;
2. then a pre-processing phase eliminates the estimated IMU offsets and prepares the sensed data for the fusion algorithm;
3. camera images are used to perform Visual Odometry to obtain estimated vehicle pose updates;
4. finally a multi-sensor fusion is performed to obtain vehicle position and orientation by means of an Extended Kalman Filter, predicting the state with IMU data and updating it with camera odometry and GNSS positions.

Calibration

IMU Calibration

Raw IMU measurements are typically biased, distorted and subjected to additive noise. For this reason they need to be calibrated, ideally at each system startup or periodically.
For accelerometer signals (a_x, a_y, a_z) corresponding biases are estimated in a standstill condition over a flat surface, placing the vehicle in 4 different directions along 2 orthogonal axes. After collecting the measurement data, for each of the 4 data logs the mean of the acceleration is taken and it is averaged with the other means. The result is an estimation of the bias for each acceleration.
Similarly, for gyroscope signals (ψ, θ, ϕ) in a stationary condition the angular velocities are logged and the mean of each signal provides an estimate of the bias.

Camera Calibration

Camera calibration consists in estimating camera intrinsic and extrinsic parameters used to model the transformation of points in 3D-environment to camera 2D image plane.

The relationship between a 3D point $[X_w, Y_w, Z_w]^T$ in the world coordinate system and its corresponding 2D point $[u, v]^T$ in the image plane is described by the Forward Imaging Model:

$$\lambda \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \mathbf{P_{int}} \mathbf{P_{ext}} \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix}$$

where λ is a scale factor, $\mathbf{P_{ext}}$ is the extrinsic parameters matrix transforming world coordinates into camera coordinates combining a roto-translation $\mathbf{R}|\mathbf{t}$:

$$\mathbf{P_{ext}} = \begin{bmatrix} \mathbf{R_{3x3}} & \mathbf{t_{3x1}} \\ \mathbf{0_{1x3}} & 1 \end{bmatrix}$$

while $\mathbf{P_{int}}$ is the intrinsic parameters matrix that describes points from camera frame into points in the 2D image plane (perspective projection):

$$\mathbf{P_{int}} = \begin{bmatrix} f_x & 0 & o_x & 0 \\ 0 & f_y & o_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

where f_x, f_y are the focal lengths in pixels along the x and y directions and o_x, o_y are the coordinates of the center of the image plane (principal point). Camera calibration consists in estimating the projection matrix $\mathbf{P} = \mathbf{P_{int}} \mathbf{P_{ext}}$.
The calibration procedure consists of the following steps:

1. **Images capture**: Capture multiple images of a calibration pattern with known geometry (e.g. a chessboard) from different angles and positions.

2. **Detect points**: Detect the corners of the chessboard matching the coordinates in the world frame ($\mathbf{M_i}$) with the corresponding ones ($\mathbf{m_i}$) in the image plane. The relation for each pair of points i of each image j is $\mathbf{M_{ij}} = \mathbf{P_j} \mathbf{m_{ij}}$, given that the intrinsic parameters are the same for each image and extrinsic ones differ from image to image $\mathbf{P_j} = \mathbf{P_{int}} \mathbf{P^j_{ext}}$.

3. **Compute homography**: Compute the map between the detected 2D points and the known 3D coordinates. By rearranging the points into a single matrix H the problem can be rewritten as $H\mathbf{P_j} = 0$.

4. **Derive projection matrix**: From the previous problem the intrinsic parameters are extracted $\mathbf{P_{int}}$ and the extrinsic parameters are estimated for each image $\mathbf{P^j_{ext}}$.

Distortion due to lenses affects how the 3D points in the camera frame are projected to the camera image plane, therefore it should be taken into account during the calibration process. For example, for a radial distortion, the perspective projection can be rewritten with a first order model as:

$$u_{\text{distorted}} = u(1 + k_1(x + y)^2)$$
$$v_{\text{distorted}} = v(1 + k_1(x + y)^2)$$

where x, y are the corresponding points in the camera frame and u, v the corresponding ones in the image plane. k_1, k_2 affect the intrinsic parameters and can be estimated by solving the minimization problem:

$$\min \sum_j \sum_i \|\mathbf{m}_{ij} - f(\mathbf{P_{int}}, k1, k2, \mathbf{P^j_{ext}}, \mathbf{M_{ij}})\|$$

The estimated projection matrix will be used in the Virtual Odometry (VO) algorithm to elaborate camera images in order to derive the corresponding points in the world coordinate frame.

Pre-processing

All the measurements should be transformed into a common world reference frame. We consider Nort-East-Down reference frame (NED) as the global reference frame.

IMU data pre-processing

Correct for the static offsets evaluated in the calibration phase for each accelerometer and gyroscope measurement. Inertial measurements are affected by high-frequency noise, therefore a Low-pass-filter (LPF) should be applied at this stage at the expense of an inducted delay.
The accelerations and angular velocities are provided by the sensor in the body frame of the IMU. These measurements have to be transformed to the global coordinates frame at this stage.

GNSS data pre-processing

From GNSS we consider latitude and longitude as positioning measures, ignoring altitude for simplicity. The geographic coordinates must be transformed into Cartesian coordinates in the NED reference frame.

Camera images pre-processing

Camera images should be prepared before the Visual Odometry algorithm in order to maximise its performance. Different considerations can be made on this step depending on the quality of the images and the computational power available.

Fusion Algorithm

Sensor fusion is performed with an Extended Kalman Filter (EKF). The state vector is composed by the position of the vehicle, the orientation in terms of Euler angles and the lateral and longitudinal velocity (all with respect to the global NED frame).
The prediction step of the algorithm updates the state vector with IMU accelerations and angular velocities. The update step of the algorithm takes into account GNSS global positions and VO poses. GNSS measures contribute to positions update while VO data contribute to both positions and orientation updates.

System description

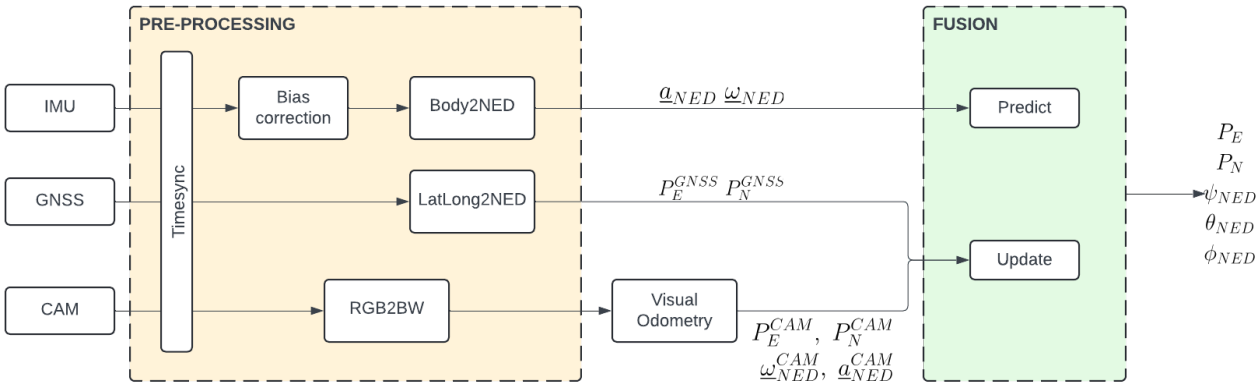

Figure 1: Architecture diagram.

In the figure above it is represented the architecture of the proposed system.
Sensors data is collected and preprocessed at first. A time-synchronization module ensures that the sensors data is synchronized before the pre-processing takes place. IMU accelerations and angular velocities biases are corrected and the measurements are transformed into the global reference frame to obtain $\underline{a}_{NED}, \underline{\omega}_{NED}$. GNSS latitude and longitude are transformed from geographical coordinates to global NED reference frame obtaining P_E^{GNSS}, P_N^{GNSS}. Camera images are transformed to greyscale images. At this point, IMU and GNSS signals are ready to be used by the fusion algorithm, but camera data still needs to be processed to retrieve useful information for the update step of the EKF. To this end, the greyscale images are fed to the Visual Odometry algorithm that provides the vehicle position and orientation based on the images $P_{CAM,E}, P_{CAM,N}, \psi_{CAM}, \theta_{CAM}, \phi_{CAM}$. IMU data is used then to predict the state of the EKF algorithm while GNSS and VO data is used to update the state. Finally, vehicle global positioning and orientation are estimated.

Pre-processing

Time-synchronization

Measurements are collected from different sources and are provided at different sample rates depending on the specific sensor. For this reason it cannot be assumed that the information received from different sensors refers to the time instant. A common solution is to consider each sensor data along with its own timestamp referred to its internal clock, identify a time-master among the sensors (in this configuration the GNSS could be a good choice since it provides a global timestamp) and shift the data in time with respect to the time-master.

IMU pre-processing

Accelerometer measurements provided by IMU sensor can be modelled as follows:

$$\underline{a}_{IMU} = \underline{a} + \beta(t) + \epsilon(t)$$

with $\underline{a}$ being pure accelerations vector (including roto-translation effects and gravity), $\beta(t)$ time-varying bias and a zero-mean Gaussian noise $\epsilon(t)$.
At first, $\epsilon(t)$ noise must be removed with a LPF. A proposed solution is a simple first order IIR filter of the form $y_k = \alpha y_{k-1} + (1-\alpha)x_k$ averaging the signal input x_k with the filter output at

the previous step y_{k-1}.
The bias term $\beta(t)$ is composed of a static term β_0 (estimated during the offline calibration phase) and a dynamic term due to ageing which it is usually described as a degrading term that affects the measure over time with a specific time factor λ (in $m/s^2/s$ - usually reported in the datasheet of the component). The dynamic term also accounts for a bias inducted by temperature changes, this can be addressed by collecting IMU measurements over different temperature values and evaluating the bias at rest. With a pre-saved set of offsets depending on the temperature $L(T)$ we can compensate for this term as well. Finally, the accelerations can be pre-processed as follows:

$$\underline{a}_{preproc} = LPF\left[\underline{a}_{IMU}\right] - \beta_{0,\underline{a}} - \lambda_{\underline{a}}\Delta t - L_{\underline{a}}(T)$$

Gyroscope measurements provided by IMU sensor can be modelled as follows:

$$\underline{\omega}_{IMU} = \underline{\omega} + \beta(t) + \eta(t)$$

where $\underline{\omega}$ identifies the pure angular velocities, $\beta(t)$ time-varying bias and a zero-mean Gaussian noise $\eta(t)$. With similar considerations made previously on the accelerometer measurements, we can pre-process the angular velocities as follows:

$$\underline{\omega}_{preproc} = LPF\left[\underline{\omega}_{IMU}\right] - \beta_{0,\underline{\omega}} - \lambda_{\underline{\omega}}\Delta t - L_{\underline{\omega}}(T)$$

Accelerometer and gyroscope data provided by the IMU sensor are referred to the body reference frame assumed to be placed on the CoG of the vehicle. In order to be used in the sensor fusion algorithm the measurements must be transformed to the global reference frame. Considering $\underline{a}_{preproc} = [a_x, a_y, a_z]^T$ being the accelerations and $\underline{\omega}_{preproc} = [r, q, p]^T$ as yaw, pitch and roll with respect to the body reference frame, we obtain global measurements with suitable rotation matrices:

$$\underline{\omega}_{NED} = \begin{bmatrix} \dot{\psi}_{\text{IMU}} \\ \dot{\phi}_{\text{IMU}} \\ \dot{\theta}_{\text{IMU}} \end{bmatrix} = R^{gyro}_{B\to NED} \begin{bmatrix} r \\ q \\ p \end{bmatrix},\ \underline{a}_{NED} = \begin{bmatrix} a_N \\ a_E \\ a_D \end{bmatrix} = R^{acc}_{B\to NED} \begin{bmatrix} a_x \\ a_y \\ a_z \end{bmatrix}$$

$\underline{\omega}_{NED}, \underline{a}_{NED}$ will be used to feed the prediction step of the EKF algorithm.

GNSS pre-processing

Latitude and longitude measurements from the GNSS receiver must be transformed into Cartesian coordinates in the NED reference frame before feeding the EKF algorithm. As an approximation we can choose NED reference frame to be the local tangent plane to the Earth, therefore a suitable non-linear transformation can be constructed to derive the Cartesian coordinates from latitude and longitude as:

$$\begin{bmatrix} P_E^{GNSS} \\ P_N^{GNSS} \end{bmatrix} = C\left(c_{\text{lat}}\ , c_{\text{long}}\ \right)$$

P_E^{GNSS}, P_N^{GNSS} will be used in the EKF algorithm to feed the update step.

Camera pre-processing

The pre-processing impemented in this step strongly depends on the camera provided images and on the Visual Odometry algorithm.
Nevertheless, images can be transformed from RGB to greyscale, reducing the payload size and enhancing processing speed. To reduce additionally the processing time of the images, downsampling can be applied. This technique reduces the size of the image for faster processing but compromises quality, possibly affecting feature tracking in the Visual Odometry algorithm.

Visual Odometry

Vehicle trajectory can be derived by means of Visual Odometry from camera images. This technique consists in using subsequent camera frames at first to extract features and then to match and track these features at each frame. In this way it is possible to reconstruct camera pose changes at each timestamp, evaluate pose changes in the world frame and finally to derive the vehicle trajectory in terms of position and orientation.
In general VO provides a rotation matrix and a translation vector to be applied to the previous pose and derive vehicle motion. In this setting it is assumed a VO algorithm that updates a trajectory based only on camera images, without any corrective feedback from the current estimated pose of EKF.
VO provides then a measurement vector of the form: $[P_{CAM,E}, P_{CAM,N}, \psi_{CAM}, \theta_{CAM}, \phi_{CAM}]^T$ including updated vehicle position and orientation with respect to NED global frame.
VO performed using a single camera is affected by scale ambiguity, meaning that the pose is computed at an arbitrary scale, due to the fact that depth information cannot be retrieved with a single camera. This limitation will be overcome with the fusion with IMU measurement that solve the scale ambiguity.
An advantage of using VO is that in situations where GNSS data is not reliable (e.g. in tunnels or city centre with canyoning effects) the EKF algorithm can still rely on an additional measurement to update the vehicle position.

Fusion algorithm (EKF)

The proposed fusion algorithm consists in an Extended Kalman Filter, with the following state vector:

$$\underline{x} = [P_E, P_N, \psi, \theta, \phi, V_E, V_N]^T$$

where P_E, P_N denote the vehicle position, ψ, θ, ϕ indicate the yaw, pitch and roll angles and V_E, V_N the velocities, all with respect to the global NED frame.
A possible implementation in C++ could be to consider the following class with private members: $\underline{x}$ state, P error covariance matrix, Q, R the process and measurement noise matrices, F, H the Jacobians of the process and sensor models and the innovation covariance matrix S. Assuming a cycle time of $10ms$. Public methods are used to invoke the steps of the EKF.

```
class EKFSensorFusion {
  private:
  std::array<float,7> x_; // State vector
  std::array<std::array<float,7>,7> P_; // Error covariance matrix
  std::array<std::array<float,7>,7> Q_; // Process noise covariance matrix
  std::array<std::array<float,7>,7> R_; // Measurement noise covariance matrix
  std::array<std::array<float,7>,7> F_; // Jacobian matrix for the process model f()\\
  std::array<std::array<float,7>,7> H_; // Jacobian matrix for the measurement model h()
  std::array<std::array<float,7>,7> S_; // Innovation covariance matrix S
  static constexpr int DELTA_T = 0.01; // 10ms sample time

  void evaluateJacobianF(); // Evaluate Jacobian F based on current state
  void evaluateJacobianH(); // Evaluate Jacobian H based on current state

  public:
  // Initialize the filter
  void initialization(const std::array<float,7> &p0, const std::array<float,7> &q0, const
  std::array<float,7> &r0);

  // Predict the state and covariance
  void predict(const std::array<float,3> &gyro_meas, const std::array<float,3> &acc_meas);

  // Update the state with measurement
  void update(const std::array<float,2> &gnss_meas, const std::array<float,5> &vo_meas);
};

```

Listing 1: EKFSensorFusion class

Initialisation

The state initialization can be obtained considering the vehicle on the origin of the global NED reference frame along the East axis, therefore $\underline{x} = \mathbf{0}_{7\times1}$.
Error covariance matrix $\mathbf{P}[0]$ can be chosen diagonal with small positive values. Process noise covariance matrix $\mathbf{Q}$ can be chosen as constant and diagonal, with entries depending on the IMU datasheet reported variance for each gyroscope and accelerometer signals. Similarly, measurements' noise covariance matrix $\mathbf{R}$ can be chosen constant and diagonal. $\mathbf{Q}, \mathbf{R}$ affect stability and convergence of the EKF so a specific tuning must be performed to guarantee desired performance. Δt will denote the sample time of the EKF algorithm used to represent the discretized formulae.

```
void EKFSensorFusion::initialization(const std::array<float,7> &p0, const std::array<float
  ,7> &q0, const std::array<float,7> &r0) {
  x_ = std::array<float,7>{0,0,0,0,0,0,0}; // initialize each state at 0
  // Use of an utility function that returns A such that diag(A) = input array
  P_ = initMatrixFromArray(p0);
  Q_ = initMatrixFromArray(q0);
  R_ = initMatrixFromArray(r0);
}

```

Listing 2: Initialization Function

Predict

The prediction step takes into account IMU accelerometer and gyroscope measures $\underline{\omega}_{NED}, \underline{a}_{NED}$ to estimate the state as follows (discretised EKF prediction step):

$$\underline{x}_{k|k-1} = f(\underline{x}_{k-1|k-1}) + \underline{w}_k$$

where $\underline{w}_k$ is the gaussian noise at the process and f is the non-linear process function using IMU data to update the state, discretized via Euler integration with Δt:

$$\begin{cases} P_{E,k} = P_{E,k-1} + V_{E,k-1}\Delta t \\ P_{N,k} = P_{N,k-1} + V_{N,k-1}\Delta t \\ \psi_k = \psi_{k-1} + \dot{\psi}_{\text{meas}} \cdot \Delta t \\ \theta_k = \theta_{k-1} + \dot{\theta}_{\text{meas}} \cdot \Delta t \\ \phi_k = \phi_{k-1} + \dot{\phi}_{\text{meas}} \cdot \Delta t \\ V_{E,k} = V_{E,k-1} + a_E\Delta t + a_D\Delta t \\ V_{N,k} = V_{N,k-1} + a_N\Delta t + a_D\Delta t \end{cases}$$

where positions are updated based on the velocity contribution on each axes, increments on the angular velocities are given by the integration of the measured rates and velocities depend on the effects of the measured accelerations.
From f we can compute the Jacobian $F_k = \frac{\delta f}{\delta \underline{x}}$. The error covariance matrix is updated by the process, providing information on how well the estimation is performing. At this stage the error will increase due to the uncertainty of the process:

$$P_{k|k-1} = F_k P_{k-1|k-1} F_k^T + Q$$

```
void EKFSensorFusion::predict(const std::array<float,3> &gyro_meas, const std::array<float,3> &acc_meas) {
  /* Extract measurements */
  float psi_dot = gyro_meas[0];
  float theta_dot = gyro_meas[1];
  float phi_dot = gyro_meas[2];

  float aE = acc_meas[0];
  float aN = acc_meas[1];

  /* Predict */
  // x[k] = x[k-1] + DELTA_T*f(x)
  x_[0] = x_[0] + DELTA_T*x[5]; // P_E
  x_[1] = x_[1] + DELTA_T*x[6]; // P_N
  x_[2] = x_[2] + DELTA_T*psi_dot; // psi
  x_[3] = x_[3] + DELTA_T*theta_dot; // theta
  x_[4] = x_[4] + DELTA_T*phi_dot; // phi
  x_[5] = x_[5] + DELTA_T*(aE+aD); // V_E
  x_[6] = x_[6] + DELTA_T*(aN+aD); // V_N

  // Compute F as the Jacobian of f()
  // Note: since the dimension of f() function is big (=7) and it depends on several trigonometric functions,
  // the explicit formulation of the Jacobian is complex. The evaluation of the Jacobian matrix has been hidden
  // in this wrapper function for the sake of simplicity.
  evaluateJacobianF();

  // Error covariance matrix prediction (discretized)
  // P[k] = P[k-1] + DELTA_T(F*P[k-1] + P[k-1]*F^T + Q)
  // Note: we use here some helper functions to compute basic matrix operations (addition, multiplication, transposition, inversion)
  P_ = add(P_, multiply(DELTA_T, add(multiply(F_, P_), add(multiply(P_, transpose(F_)), Q_))));
}

```

Listing 3: Predict Function

Update

In the update step the GNSS positions and camera Visual Odometry are fused to create the innovation that updates the state vector getting a correction of the vehicle position and orientation. In fact, the measurements vector containing GNSS and Visual Odometry data $\left[P^{GNSS}_{E,k-1}, P^{GNSS}_{N,k-1}, P^{CAM}_{E,k-1}, P^{CAM}_{N,k-1}, \psi^{CAM}_{k-1}, \theta^{CAM}_{k-1}, \phi^{CAM}_{k-1}\right]^T$ is defined as:

$$\underline{z}_k = h(\underline{x}_{k|k-1}) + \underline{v}_k$$

where $\underline{v}_k$ is the Gaussian noise at the sensors and the sensor model matches the state vector $h(\underline{x}_{k|k-1}) = \underline{x}_{k|k-1}$. From h we can compute the Jacobian $H_k = \frac{\delta h}{\delta \underline{x}}$. Notice that, based on the current formulation of the problem, the Jacobian $H_k = I_{7\times 7}$. At this point the Kalman gain can be computed and used in the state update formula:

$$K_k = P_{k|k-1}H_k^T S_k^{-1} = P_{k|k-1}S_k^{-1}$$
$$\underline{x}_{k|k} = \underline{x}_{k|k-1} + K_k(\underline{z}_k - h(\underline{x}_{k|k-1}))$$

where $S_k = H_k P_{k|k-1} H_k^T + R_k = P_{k|k-1} + R_k$ is the innovation covariance matrix.
Finally, the prediction error is updated and will decrease at this step because of the new

incoming corrective data:

$$P_{k|k} = (I - K_k H_k) P_{k|k-1} = (I - K_k) P_{k|k-1}$$

```
void EKFSensorFusion::update(const std::array<float,2> &gnss_meas, const std::array<float
  ,5> &vo_meas) {
  /* Extract measurements */
  std::array<float,7> meas = vconcat(gnss_meas, vo_meas);

  // Compute H as the Jacobian of h()
  evaluateJacobianH();

  // Innovation covariance matrix S[k] = H*P[k]*H^T + R = P[k] + R
  S_ = add(P_, R_);

  // Kalman gain
  K_ = multiply(P_, inverse(S_));

  // State update x[k+1] = x[k] + K*(z - h(x))
  x_ = add(x_, multiply(K_, sub(meas, x_)));

  // Error covariance matrix update
  P_ = multiply(subtract(idMatrix, K_), P_);
}

```

Listing 4: Update Function

Considerations

- **Real-time.** Specific choices to make the implementation real-time can be made directly on the data structures used in the code. For example, should be avoided high-level libraries like "Eigen" to manage matrix operations since they may use dynamic memory allocation which is not suitable for real-time applications. Moreover, the use of std::array instead of a std::vector ensures static memory allocation.
  Input sensors have different sample times which must be accounted for. We could consider IMU data provided at 100Hz, slower GNSS data provided at 10Hz and camera at 30fps. A suitable choice of the cycle time of each software component in the architecture is critical. $\Delta t = 10$ms could be a good choice based on the sensor data available and commonly adopted in the Automotive industry.

- **Error handling.** Sensors like IMU and GNSS receivers usually provide health status information about their estimations. Based on the GNSS status we could prevent the use of degraded positioning information in the update step of the EKF, consisting, for example, in a "condition evaluator" block that makes sure the innovation given by the GNSS is not considered.
  Moreover, IMU data is used in the core prediction step, if a secondary IMU is available it could be used to prevent degradation of inertial measurements. In the VO algorithm a status indicating the number of tracked features or the quality of the estimated motion could be useful to prevent the usage of not reliable pose estimation in the update step of the EKF. In general, a logic that monitors health status signals for each sensor would be useful not only to prevent the use of degraded measurements but also to resort to a fallback pose estimation solution, potentially less accurate but that can still work with body signals from CAN network (e.g. wheel speed sensors).

- **Quaternions.** Orientation of the vehicle can be represented conveniently with quaternions instead of Euler angles. The formulation of the EKF would change by having an additional dimension in the state $\underline{x} = [P_E, P_N, q0, q1, q2, q3, V_E, V_N]^T$, leading to and increased matrices dimension and therefore an increased complexity of the state update computation. Nevertheless, quaternions would ease the transformation of body frame angular velocities and accelerations into NED global frame avoiding the usage of rotation matrices and matrix multiplication.

- **System scalability.** As already reported, scalability in terms of efficiency of the system can be achieved using greyscale images instead of RBG images for VO and quaternions to represent orientation, easing the measurements transformations in the preprocessing phase. In terms of reliability, redundancy of the IMU could prevent unreliable estimations in case of faults and time-synchronization ensures that the input data is coherent and can be reliable. More accurate positions can be obtained with a differential-GPS receiver, but are rarely used in common automotive industrial applications due to high cost. Additional sensors can be included as well, for example a stereo camera would provide depth information useful to have better pose estimations.