كلية الهندسة جامعه حلوان

جامعة حلوان

Faculty of Engineering

Electronics & Communications Engineering Department

Helwan university

1st Term Evaluation report

TELEMATICS CONTROL UNIT IN MODERN VEHICLES

Supervised by:

- **Dr Mohamed El-Dakroury**

Project team members:

- Ahmed Mohamed Abdellah
- Ahmed Sherif Mohamed
- Mariam Mostafa Mohammed
- Nada Mohamed Ibrahim
- Samaa Khalid Mohamed

Table of Contents

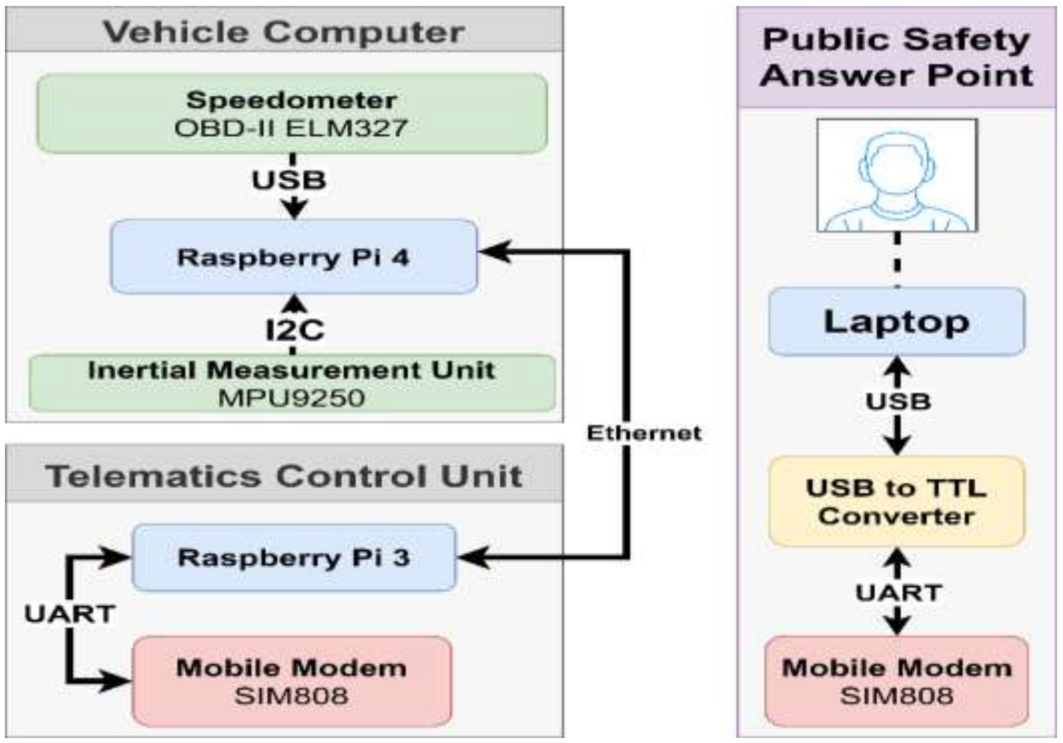
1. Problem Statement

Modern vehicles are becoming increasingly complex with a variety of electronic systems that need to communicate with each other. Telematics control units (TCUs) have become an essential part of modern vehicles, providing the connectivity required to exchange data between different systems. However, developing a TCU that can handle the latest technologies, such as automotive Ethernet and emergency call systems, along with a reliable automotive dead reckoning system can be a challenging task.

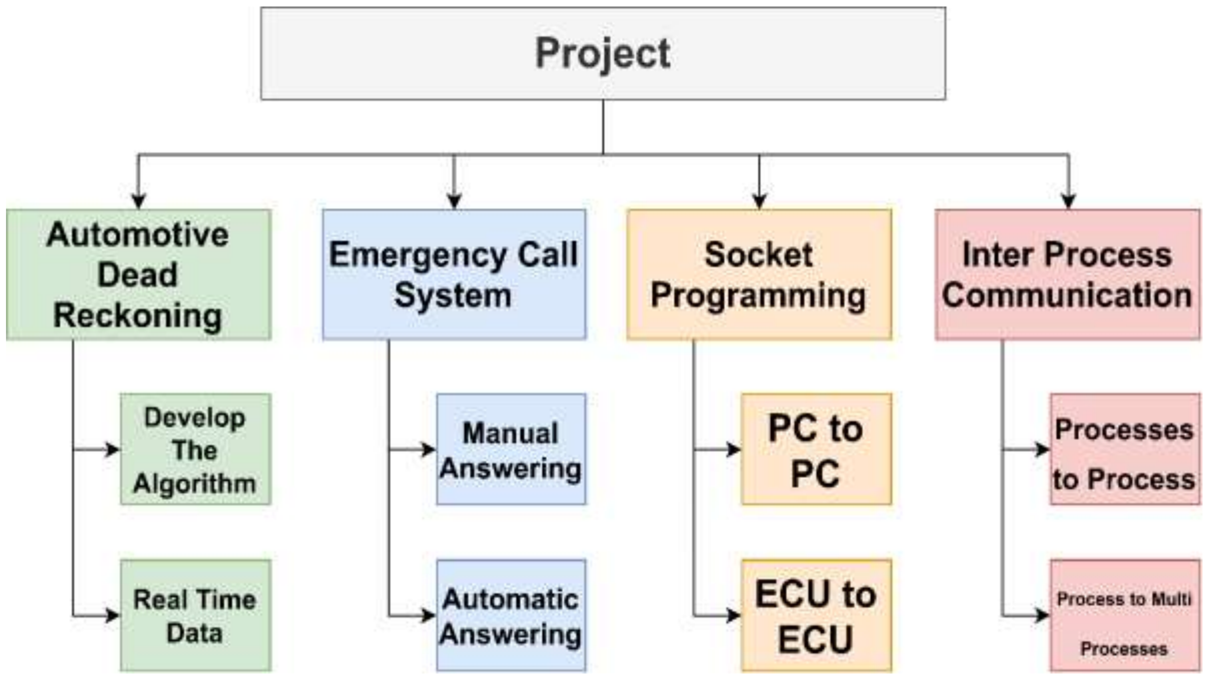
This graduation project aims to design and implement a TCU hardware circuit that emulates the behaviour of a zonal architecture modern vehicle, capable of supporting automotive Ethernet, emergency call systems, and automotive dead reckoning systems. The project will use Linux for the TCU and implement interprocess communication and socket programming to facilitate data exchange.

The primary challenge for this project is to ensure that the TCU hardware circuit can reliably handle the exchange of data between different systems in a modern vehicle. This involves dealing with the high-speed data transfer requirements of automotive Ethernet and the real-time communication demands of emergency call systems. Additionally, the implementation of a reliable automotive dead reckoning system presents a significant challenge, given the need to accurately calculate the vehicle's position and movement even in areas with limited or no GPS signal.

Therefore, the main problem this project seeks to address is the development of a TCU hardware circuit that can handle the latest technologies in modern vehicles, while providing reliable and accurate communication between the different electronic systems.



2. Work Breakdown Structure



Part 1: Automotive Dead Reckoning

Automotive dead reckoning involves the use of sensors and algorithms to calculate the vehicle's position and movement, even when there is limited or no GPS signal. This part of the project will involve designing and implementing the automotive dead reckoning system for the TCU hardware circuit.

Global Positioning System (GPS) aided Inertial Navigation Systems (INS) play an important role in modern navigation and guidance applications. They combine the advantages of two distinct technologies, namely GPS and INS, to provide accurate and reliable position and velocity information in real-time. GPS is a satellite-based navigation system that provides global coverage and accurate positioning capabilities, while INS is a self-contained navigation system that can provide position, velocity, and attitude information by integrating measurements from inertial sensors. The integration of these two systems results in a navigation solution that is accurate, robust, and reliable.

INS, also known as Inertial Measurement Units (IMUs), are based on a set of accelerometers and gyroscopes that measure acceleration and angular velocity, respectively. The information from these sensors is used to calculate the change in position, velocity, and attitude of a moving object over time. However, INS suffers from errors that accumulate over time, resulting in an increased uncertainty in position and velocity estimates. These errors are caused by several factors, including sensor noise, bias, and drift. GPS, on the other hand, provides absolute positioning information by measuring the time delay between the transmission of signals from satellites and their reception by a GPS receiver on the ground. However, GPS can suffer from signal degradation and loss of signal reception in urban canyons, tunnels, and other areas with limited sky view.

The combination of GPS and INS can overcome the limitations of each system and provide a more accurate, robust, and reliable navigation solution. GPS measurements can be used to correct the errors in the INS, thereby reducing the error accumulation over time. This is achieved by comparing the position and velocity estimates from INS with the GPS measurements and then adjusting the INS estimates accordingly. This process, known as GPS/INS integration, can be achieved through several algorithms, including the Extended Kalman Filter (EKF) and the Unscented Kalman Filter (UKF) and different approaches to the problem of localization like the lossy coupled approach and tightly coupled approach.

The following tasks will be involved in this part of the project:

- Research and identify the sensors and algorithms required for the automotive dead reckoning system.
- Develop and test the algorithm to calculate the vehicle's position and movement based on sensor data.
- Implement the algorithm in the TCU hardware circuit.
- Conduct testing and validation of the automotive dead reckoning system to ensure it is accurate and reliable.

Part 2: Emergency Call System

The emergency call system is a critical component of modern vehicles that allows the driver or passengers to quickly summon help in the event of an accident or other emergency. This part of the project will involve designing and implementing the emergency call system for the TCU hardware circuit.

The following tasks will be involved in this part of the project:

- Research and identify the requirements for the emergency call system, including the necessary hardware and software components.
- Develop and test the emergency call system software, including the user interface and communication protocols.
- Implement the emergency call system software in the TCU hardware circuit.
- Conduct testing and validation of the emergency call system to ensure it functions correctly and can communicate with external emergency services.

Part 3: Socket Programming

Socket programming is a method of communication between different processes or computers over a network. This part of the project will involve developing and implementing the socket programming functionality required for the TCU hardware circuit.

<u>The following tasks will be involved in this part of the project:</u>

- Research and identify the socket programming protocols and requirements for the TCU hardware circuit.
- Develop and test the socket programming functionality, including sending and receiving data packets.
- Implement the socket programming functionality in the TCU hardware circuit.
- Conduct testing and validation of the socket programming functionality to ensure it can communicate with other systems in a modern vehicle.

Part 4: Interprocess Communication

Interprocess communication is a method of communication between different processes or threads within the same computer. This part of the project will involve developing and implementing the interprocess communication functionality required for the TCU hardware circuit.

<u>The following tasks will be involved in this part of the project:</u>

- Research and identify the interprocess communication protocols and requirements for the TCU hardware circuit.
- Develop and test the interprocess communication functionality, including sending and receiving data packets between different processes or threads.
- Implement the interprocess communication functionality in the TCU hardware circuit.
- Conduct testing and validation of the interprocess communication functionality to ensure it can facilitate the exchange of data between different electronic systems in a modern vehicle.

Overall, each part of the project is critical to the development of a fully-functional TCU hardware circuit capable of handling the latest technologies in modern vehicles. The completion of each part will be necessary to successfully integrate and test the TCU hardware circuit as a whole.

Automotive Dead Reckoning

GPS/INS integration is used in a wide range of applications, including aircraft, ships, and missiles. In aviation, GPS/INS integration is used for navigation, landing, and takeoff. In ships, it is used for navigation and collision avoidance. In missiles, it is used for guidance and control.

The accuracy and reliability of GPS/INS integration are critical in these applications, as they can directly affect the safety, efficiency, and effectiveness of the systems. The use of GPS/INS integration has several advantages over using GPS or INS alone. First, it provides accurate and reliable position and velocity estimates in real-time, even in areas with limited GPS signal reception. Second, it provides high update rates, which is essential for applications that require fast and precise navigation, such as aviation and missile guidance. Third, it is self-contained and does not rely on external infrastructure, which makes it suitable for use in remote and hostile environments.

Inertial navigation is a technique for determining the position, velocity, and orientation of an object using accelerometers and gyroscopes. This method is entirely self-contained, which means it does not require any external information sources that may be affected by disturbances or jamming. Consequently, it is a popular navigation option for applications that require consistent and reliable service. It also provides a comprehensive six-degree-of-freedom navigation solution and has a fast update rate of 100 Hz or higher.

The importance of accurately locating the vehicle in an e-call system cannot be overstated. In a serious accident, every second counts, and the ability of the emergency services to respond quickly and effectively can mean the difference between life and death. Accurate location information allows emergency services to quickly dispatch the appropriate personnel and resources to the scene of the accident, improving the chances of survival for those involved. In addition to improving response times and potentially saving lives, accurate location information can also help emergency services to better manage resources in the event of a major incident. By having accurate real-time information on the location and severity of accidents, emergency services can quickly and efficiently allocate resources where they are needed most, reducing the likelihood of delays in responding to incidents

The importance of accurate location information is further emphasized when considering the increasingly complex and diverse nature of modern road networks. With more and more cars on the road, and with those cars increasingly equipped with sophisticated technology, accurately locating a vehicle in the event of an accident is not always straightforward. In urban areas, tall buildings, bridges and other structures can obstruct GPS signals, leading to inaccurate location information. In rural areas, poor network coverage can similarly hamper location accuracy.

GPS alone Problems:

-GPS signals can be degraded or blocked by both natural and manmade sources, resulting in inaccurate data or complete loss of GPS signal

-Determining vehicle position using GPS requires at least 4 satellites must be reachable and the signal is able to be received and decoded otherwise GPS navigation can't work.

-underground tunnels, Reflective high and dense buildings structures and Multilevel roads cause a significant challenge for GPS Signals to be transferred and consistent positional data of the vehicle to be observed due to GPS outage in such scenarios.

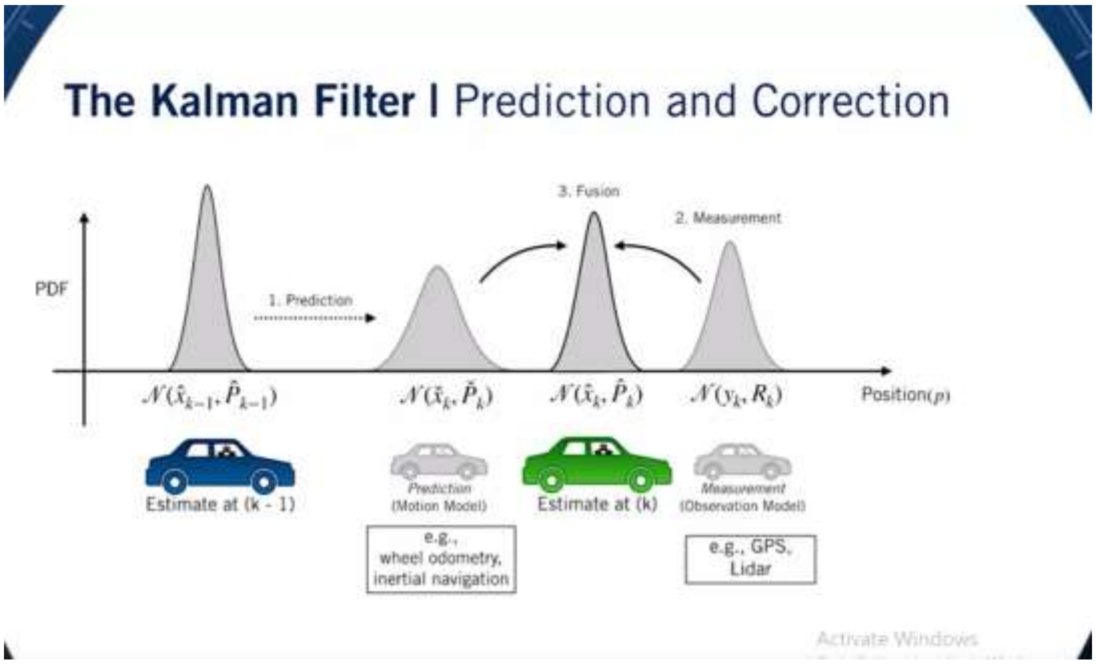
-Ionospheric delay and Multipath effects

-A clock synchronization error of 10^-6 sec in GPS gives a 300 m position error

Kalman Filters:

The Kalman filter remains an important tool to fuse measurements from several sensors to estimate in real-time the state of a robotic system such as a self-driving car. The Kalman filter is an algorithm that provides an optimal estimate of the state of a linear dynamic system, given measurements that are corrupted by noise. It is widely used in various fields, including control theory, signal processing, and navigation. The Kalman filter is a recursive algorithm that estimates the state of a dynamic system by taking into account the noise in the measurements. It is based on the idea of predicting the state of the system at each time step and then updating the prediction using the measurements. The algorithm uses a mathematical model of the system and the measurements to calculate the optimal estimate of the state. two-step process: prediction and update. In the prediction step, the filter uses the previous estimate of the state and the model of the system to predict the state of the system at the next time step. The prediction includes an estimate of the noise in the system, which is added to the predicted state. The predicted state and the predicted covariance matrix are then used as the initial values for the update step. In the update step, the filter uses the predicted state and the measurements to calculate the optimal estimate of the state. The update includes a correction factor that takes into account the difference between the predicted state and the measurements. The correction factor is weighted by a gain factor, which is calculated from the predicted covariance matrix and the covariance matrix of the measurement noise.

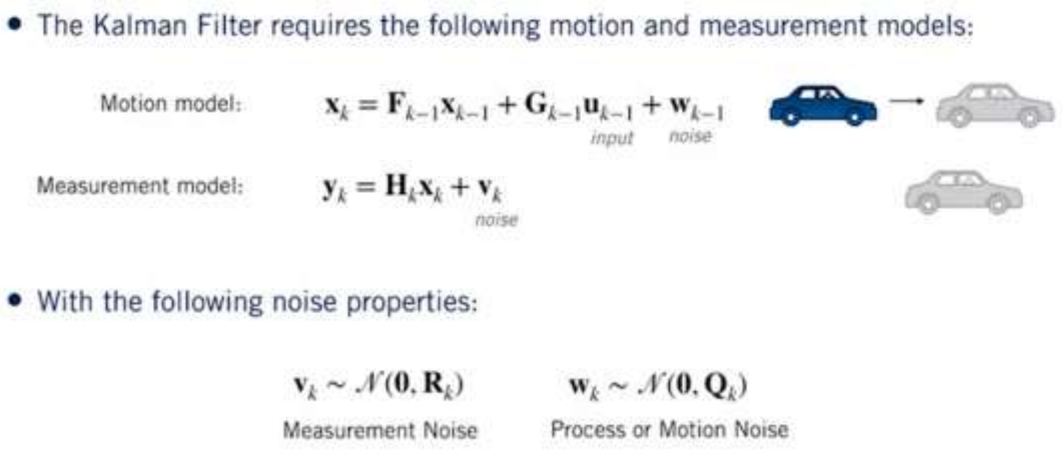
For example:



The Kalman Filter I Prediction and Correction

PDF
1. Prediction
3. Fusion
2. Measurement
$\mathcal{N}(\hat{x}_{k-1}, \hat{P}_{k-1})$
$\mathcal{N}(\check{x}_k, \check{P}_k)$
$\mathcal{N}(\hat{x}_k, \hat{P}_k)$
$\mathcal{N}(y_k, R_k)$
Position(p)
Estimate at (k - 1)
Prediction (Motion Model)
e.g., wheel odometry, inertial navigation
Estimate at (k)
Measurement (Observation Model)
e.g., GPS, Lidar
Activate Windows

Starting from an initial probabilistic estimate at time k minus 1, our goal is to use a motion model which could be derived from wheel odometry or inertial sensor measurements to predict our new state. Then, we'll use the observation model derived from GPS for example, to correct that prediction of vehicle position at time k. Each of these components, the initial estimate, the predicted state, and the final corrected state are all random variables that we will specify by their means and covariances. In this way, we can think of the Kalman filter as a technique to fuse information from different sensors to produce a final estimate of some unknown state, taking into account, uncertainty in motion and in our measurements



- The Kalman Filter requires the following motion and measurement models:

Motion model: $\mathbf{x}_k = \mathbf{F}_{k-1}\mathbf{x}_{k-1} + \mathbf{G}_{k-1}\underbrace{\mathbf{u}_{k-1}}_{input} + \underbrace{\mathbf{w}_{k-1}}_{noise}$

Measurement model: $\mathbf{y}_k = \mathbf{H}_k\mathbf{x}_k + \underbrace{\mathbf{v}_k}_{noise}$

- With the following noise properties:

$\mathbf{v}_k \sim \mathcal{N}(\mathbf{0}, \mathbf{R}_k)$ Measurement Noise

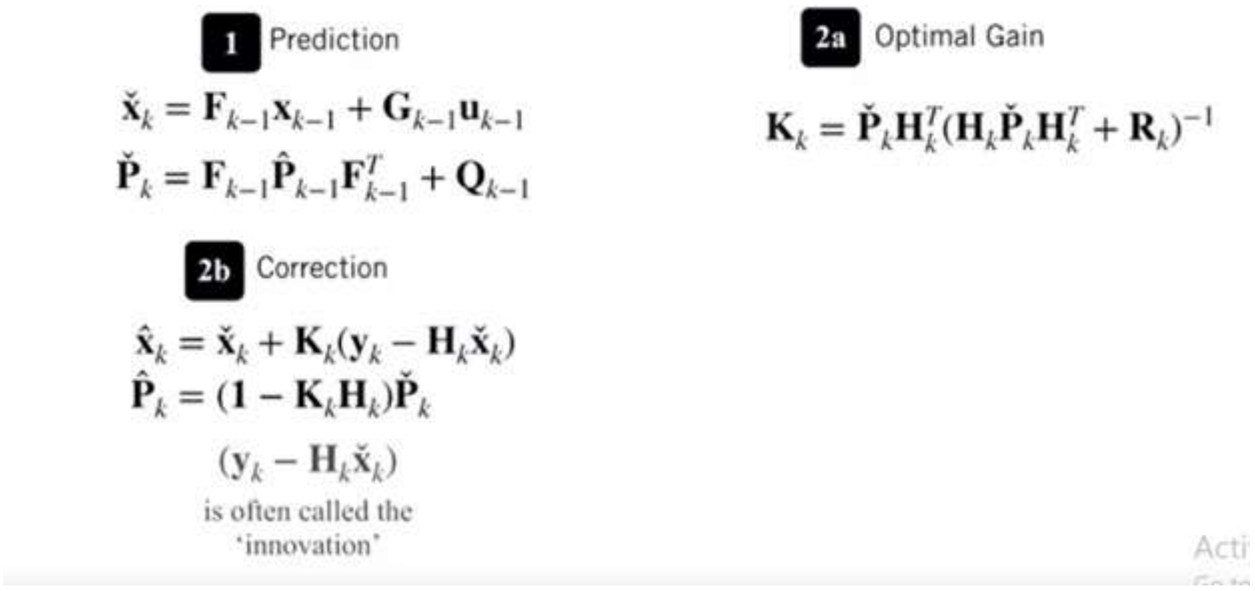
$\mathbf{w}_k \sim \mathcal{N}(\mathbf{0}, \mathbf{Q}_k)$ Process or Motion Noise

How LKF works?

1-First we use the process model to predict how our states, remember, that we're now typically talking about evolving states and non-state parameters evolved since the last time step, and will propagate our uncertainty:

2-Second, we'll use our measurement to correct that prediction based on our measurement residual or innovation and our optimal gain

3- Finally, we'll use the gain to also propagate the state covariance from our prediction to our corrected estimate



1 Prediction

$$\check{\mathbf{x}}_k = \mathbf{F}_{k-1}\mathbf{x}_{k-1} + \mathbf{G}_{k-1}\mathbf{u}_{k-1}$$
$$\check{\mathbf{P}}_k = \mathbf{F}_{k-1}\hat{\mathbf{P}}_{k-1}\mathbf{F}^T_{k-1} + \mathbf{Q}_{k-1}$$

2a Optimal Gain

$$\mathbf{K}_k = \check{\mathbf{P}}_k\mathbf{H}_k^T(\mathbf{H}_k\check{\mathbf{P}}_k\mathbf{H}_k^T + \mathbf{R}_k)^{-1}$$

2b Correction

$$\hat{\mathbf{x}}_k = \check{\mathbf{x}}_k + \mathbf{K}_k(\mathbf{y}_k - \mathbf{H}_k\check{\mathbf{x}}_k)$$
$$\hat{\mathbf{P}}_k = (\mathbf{1} - \mathbf{K}_k\mathbf{H}_k)\check{\mathbf{P}}_k$$

$(\mathbf{y}_k - \mathbf{H}_k\check{\mathbf{x}}_k)$
is often called the 'innovation'

<u>Why use Kalman filter?</u>

In statistics, "BLUE" stands for "Best Linear Unbiased Estimator". The Kalman filter is an example of a BLUE estimator, meaning that it is a linear estimator that is unbiased and has the smallest variance of all other unbiased linear estimators. “For mathematical proof please consider the references.

Non-linear Kalman filter:

While the original Kalman filter is designed to handle linear systems, it is often not sufficient for systems that exhibit nonlinear dynamics. In such cases, an extension of the Kalman filter known as the nonlinear Kalman filter must be used. And linear systems don’t exist in reality

Nonlinear Kalman filters come in many forms, each of which has its own strengths and weaknesses. In this essay, we will explore some of the most common forms of nonlinear Kalman filters, including the Extended Kalman Filter (EKF), the Unscented Kalman Filter (UKF), and the Particle Filter, Our focus will be on EKF and some variation of it known as indirect Kalman filter or Error state Kalman filter as it’s the most widely used filter in localization of a vehicle in self-driving car

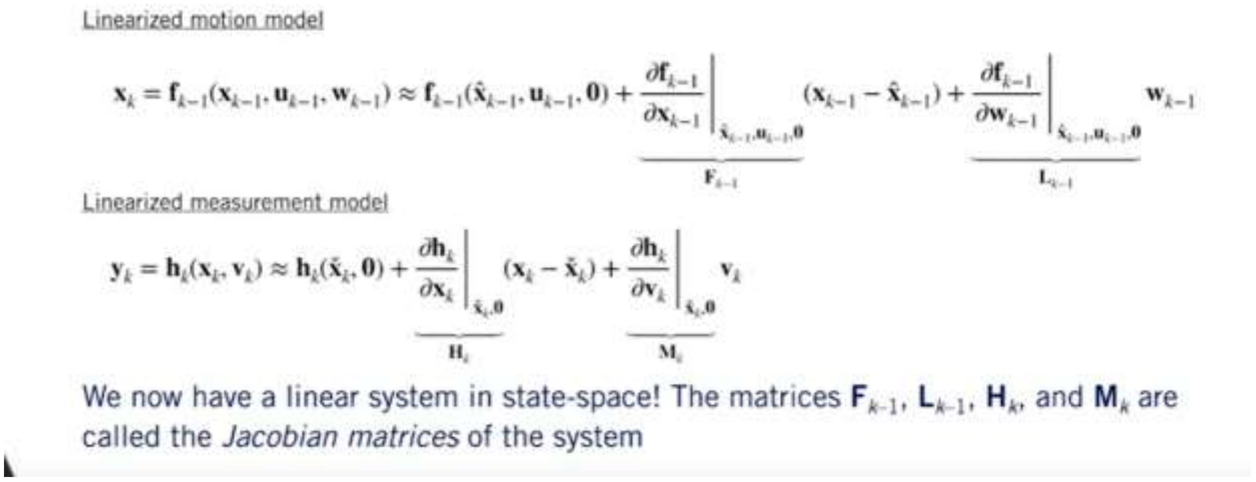
<u>EKF:</u>

The Extended Kalman Filter (EKF) is perhaps the most well-known and widely used nonlinear Kalman filter. The EKF is a simple extension of the original Kalman filter that linearizes the system dynamics and the measurement function around the current estimate of the system state. The linearization is accomplished by taking the first-order Taylor expansion of the nonlinear function. The EKF is easy to implement and computationally efficient, but it can suffer from numerical instability if the linearization is not accurate, and it can be sensitive to the choice of initial conditions.

<u>How this is done?</u>

Let’s assume we have a non-linear measurement model for real car position and velocity are

The function of the car orientation which is a non-linear parameter.



Linearized motion model

$$\mathbf{x}_k = \mathbf{f}_{k-1}(\mathbf{x}_{k-1}, \mathbf{u}_{k-1}, \mathbf{w}_{k-1}) \approx \mathbf{f}_{k-1}(\hat{\mathbf{x}}_{k-1}, \mathbf{u}_{k-1}, \mathbf{0}) + \underbrace{\frac{\partial \mathbf{f}_{k-1}}{\partial \mathbf{x}_{k-1}}\bigg|_{\hat{\mathbf{x}}_{k-1},\mathbf{u}_{k-1},\mathbf{0}}}_{\mathbf{F}_{k-1}} (\mathbf{x}_{k-1} - \hat{\mathbf{x}}_{k-1}) + \underbrace{\frac{\partial \mathbf{f}_{k-1}}{\partial \mathbf{w}_{k-1}}\bigg|_{\hat{\mathbf{x}}_{k-1},\mathbf{u}_{k-1},\mathbf{0}}}_{\mathbf{L}_{k-1}} \mathbf{w}_{k-1}$$

Linearized measurement model

$$\mathbf{y}_k = \mathbf{h}_k(\mathbf{x}_k, \mathbf{v}_k) \approx \mathbf{h}_k(\check{\mathbf{x}}_k, \mathbf{0}) + \underbrace{\frac{\partial \mathbf{h}_k}{\partial \mathbf{x}_k}\bigg|_{\check{\mathbf{x}}_k,\mathbf{0}}}_{\mathbf{H}_k} (\mathbf{x}_k - \check{\mathbf{x}}_k) + \underbrace{\frac{\partial \mathbf{h}_k}{\partial \mathbf{v}_k}\bigg|_{\check{\mathbf{x}}_k,\mathbf{0}}}_{\mathbf{M}_k} \mathbf{v}_k$$

We now have a linear system in state-space! The matrices $\mathbf{F}_{k-1}$, $\mathbf{L}_{k-1}$, $\mathbf{H}_k$, and $\mathbf{M}_k$ are called the *Jacobian matrices* of the system

Notice that the motion model is linearized around the previous corrected state “recent estimate”

And for the measurement model, we'll linearize our prediction of the current state based on the motion model. So now we have a linear system in state-space, and the matrices F, L, H, and M are called Jacobian matrices of the system. Computing these matrices correctly is the most important and difficult step in the extended Kalman filter algorithm, and it's also the most common place to make mistakes.

In vector calculus a Jacobian matrix is the matrix of all first order partial derivatives of a function:

$$\frac{\partial \mathbf{f}}{\partial \mathbf{x}} = \begin{bmatrix} \frac{\partial \mathbf{f}}{\partial x_1} & \cdots & \frac{\partial \mathbf{f}}{\partial x_n} \end{bmatrix} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \cdots & \frac{\partial f_m}{\partial x_n} \end{bmatrix}$$

for example:

$$\mathbf{f}(\mathbf{x}) = \begin{bmatrix} f_1 \\ f_2 \end{bmatrix} = \begin{bmatrix} x_1 + x_2 \\ x_1^2 \end{bmatrix} \longrightarrow \frac{\partial \mathbf{f}}{\partial \mathbf{x}} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 2x_1 & 0 \end{bmatrix}$$

EKF linearization of motion and measurement model can be summarized as follow:

Linearized motion model

$$\mathbf{x}_k = \mathbf{f}_{k-1}(\hat{\mathbf{x}}_{k-1}, \mathbf{u}_{k-1}, \mathbf{0}) + \mathbf{F}_{k-1}\left(\mathbf{x}_{k-1} - \hat{\mathbf{x}}_{k-1}\right) + \mathbf{L}_{k-1}\mathbf{w}_{k-1}$$

Linearized measurement model

$$\mathbf{y}_k = \mathbf{h}_k(\check{\mathbf{x}}_k, \mathbf{0}) + \mathbf{H}_k\left(\mathbf{x}_k - \check{\mathbf{x}}_k\right) + \mathbf{M}_k\mathbf{v}_k$$

Prediction

$$\check{\mathbf{x}}_k = \mathbf{f}_{k-1}(\hat{\mathbf{x}}_{k-1}, \mathbf{u}_{k-1}, \mathbf{0})$$
$$\check{\mathbf{P}}_k = \mathbf{F}_{k-1}\hat{\mathbf{P}}_{k-1}\mathbf{F}_{k-1}^T + \mathbf{L}_{k-1}\mathbf{Q}_{k-1}\mathbf{L}_{k-1}^T$$

Optimal gain

$$\mathbf{K}_k = \check{\mathbf{P}}_k\mathbf{H}_k^T(\mathbf{H}_k\check{\mathbf{P}}_k\mathbf{H}_k^T + \mathbf{M}_k\mathbf{R}_k\mathbf{M}_k^T)^{-1}$$

Correction

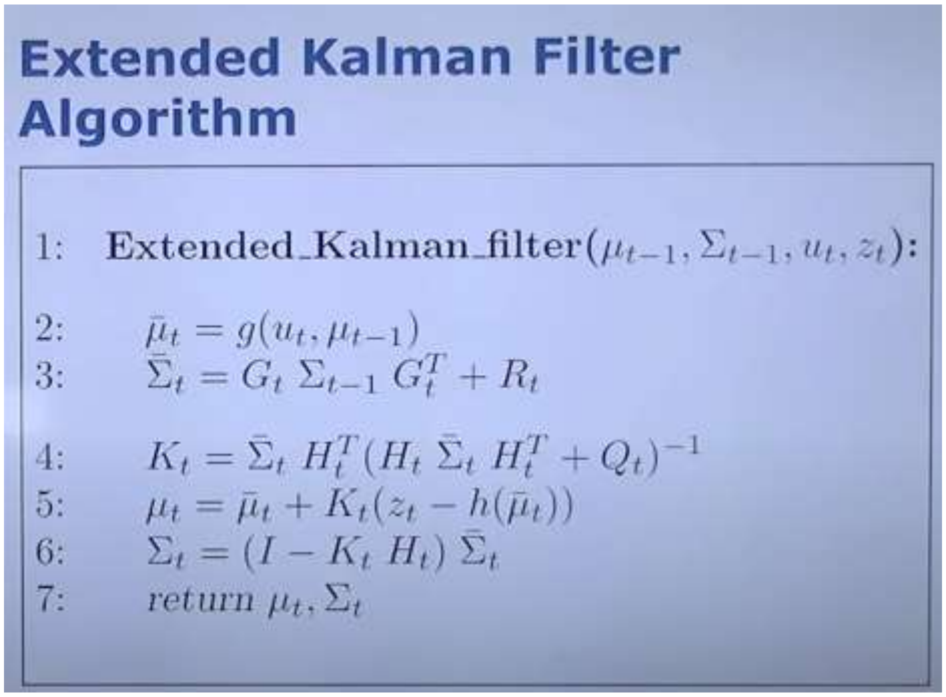
$$\hat{\mathbf{x}}_k = \check{\mathbf{x}}_k + \mathbf{K}_k(\mathbf{y}_k - \mathbf{h}_k(\check{\mathbf{x}}_k, \mathbf{0}))$$
$$\hat{\mathbf{P}}_k = (\mathbf{1} - \mathbf{K}_k\mathbf{H}_k)\check{\mathbf{P}}_k$$

$\check{\mathbf{x}}_k$	Prediction (given motion model) at time k
$\hat{\mathbf{x}}_k$	Corrected prediction (given measurement) at time k

A very important note is that the X_k-X_(k-1) in the prediction and correction steps of the mean of our state are zeros because their expected values are zeros “for mathematical proof consider the references”

Another way to look at the EKF is as follows:

Where μ is equivalent to x , ∑ to P ,z to y ,R to Q and Q to R.



Extended Kalman Filter Algorithm

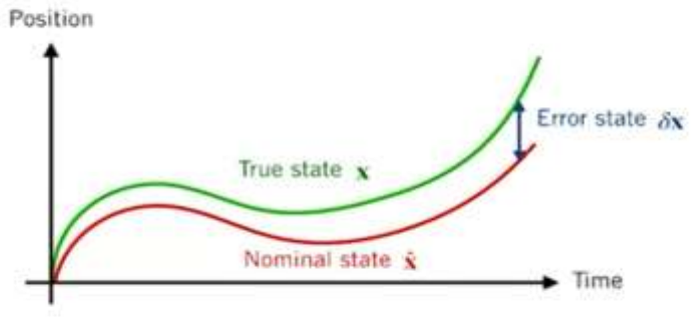
```
1:  Extended_Kalman_filter($\mu_{t-1}, \Sigma_{t-1}, u_t, z_t$):

2:      $\bar{\mu}_t = g(u_t, \mu_{t-1})$
3:      $\bar{\Sigma}_t = G_t\, \Sigma_{t-1}\, G_t^T + R_t$

4:      $K_t = \bar{\Sigma}_t\, H_t^T (H_t\, \bar{\Sigma}_t\, H_t^T + Q_t)^{-1}$
5:      $\mu_t = \bar{\mu}_t + K_t(z_t - h(\bar{\mu}_t))$
6:      $\Sigma_t = (I - K_t\, H_t)\, \bar{\Sigma}_t$
7:      return $\mu_t, \Sigma_t$
```

The error state Kalman filter “The Implemented filter in the algorithm “:

The Error State Kalman Filter is a type of extended Kalman filter that estimates the error between the predicted state and the actual state. Instead of directly estimating the state of the system, the ESKF estimates the deviation of the actual state from the predicted state. This deviation, or error state, is propagated through the nonlinear model equations to estimate the error in the system state at each time step. The estimated error state is then used to correct the predicted state, and the process is repeated for the next time step.

One of the main advantages of the Error State Kalman Filter is that it is a more flexible approach to nonlinear estimation. It can handle a wider range of nonlinear models compared to the traditional Kalman filter. The ES-EKF can also estimate a wide range of system states and parameters, making it useful in many different applications. The filter can handle systems with any number of state variables, provided that the model equations are nonlinear.

The process can be summarized with the following example:


Position
Error state $\delta\mathbf{x}$
True state $\mathbf{x}$
Nominal state $\hat{\mathbf{x}}$
Time

tracking the position of a car over time. The green line shows the true position of the car, which is the quantity we're trying to estimate. The red line is the nominal state, or our best guess of what the true state could be based on what we know about the car's motion model and acceleration and breaking inputs. Of course, our motion model is never perfect, and there is always some random process noise. These errors build up over time as we integrate the motion model. We can think of the error state as the place where all of these modelling errors and process noise accumulate over time, so that the error state is just the difference between the nominal state and the true state at any given time. If we can figure out what the error state is, we can use it as a correction to the nominal state to bring us closer to the true state.

The error state linearized motion and measurement model can be obtained by rearranging the EKF equations as follows:

$$\underset{\text{True State}}{\mathbf{x}} = \underset{\substack{\text{Nominal State}\\(\text{"Large"})}}{\hat{\mathbf{x}}} + \underset{\substack{\text{Error State}\\(\text{"Small"})}}{\delta\mathbf{x}}$$

Linearized motion model

$$\mathbf{x}_k = \mathbf{f}_{k-1}(\hat{\mathbf{x}}_{k-1}, \mathbf{u}_{k-1}, \mathbf{0}) + \mathbf{F}_{k-1}\left(\mathbf{x}_{k-1} - \hat{\mathbf{x}}_{k-1}\right) + \mathbf{L}_{k-1}\mathbf{w}_{k-1}$$

$$\Downarrow$$

$$\underbrace{\mathbf{x}_k - \mathbf{f}_{k-1}(\hat{\mathbf{x}}_{k-1}, \mathbf{u}_{k-1}, \mathbf{0})}_{\delta\mathbf{x}_k} = \mathbf{F}_{k-1}\underbrace{\left(\mathbf{x}_{k-1} - \hat{\mathbf{x}}_{k-1}\right)}_{\delta\mathbf{x}_{k-1}} + \mathbf{L}_{k-1}\mathbf{w}_{k-1}$$

Error state

Linearized measurement model

$$\mathbf{y}_k = \mathbf{h}_k(\check{\mathbf{x}}_k, \mathbf{0}) + \mathbf{H}_k\left(\mathbf{x}_k - \check{\mathbf{x}}_k\right) + \mathbf{M}_k\mathbf{v}_k$$

$$\Downarrow$$

$$\mathbf{y}_k = \mathbf{h}_k(\check{\mathbf{x}}_k, \mathbf{0}) + \mathbf{H}_k\underbrace{\left(\mathbf{x}_k - \check{\mathbf{x}}_k\right)}_{\delta\mathbf{x}_k} + \mathbf{M}_k\mathbf{v}_k$$

Error state

How does it work?

The ES-KF algorithm:

Loop:

1. Update nominal state with motion model

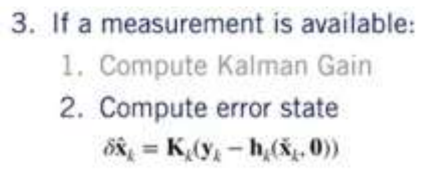
$$\check{\mathbf{x}}_k = \mathbf{f}_{k-1}(\mathbf{x}_{k-1}, \mathbf{u}_{k-1}, \mathbf{0})$$

2. Propagate uncertainty

$$\check{\mathbf{P}}_k = \mathbf{F}_{k-1}\mathbf{P}_{k-1}\mathbf{F}_{k-1}^T + \mathbf{L}_{k-1}\mathbf{Q}_{k-1}\mathbf{L}_{k-1}^T$$

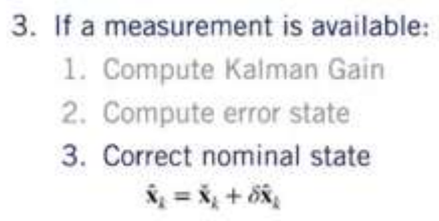
3. If a measurement is available:
    1. Compute Kalman Gain

$$\mathbf{K}_k = \check{\mathbf{P}}_k\mathbf{H}_k^T(\mathbf{H}_k\check{\mathbf{P}}_k\mathbf{H}_k^T + \mathbf{R})^{-1}$$



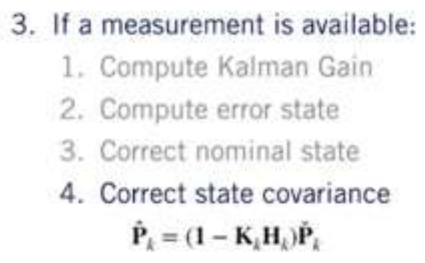
3. If a measurement is available:
    1. Compute Kalman Gain
    2. Compute error state

$$\delta\hat{\mathbf{x}}_k = \mathbf{K}_k(\mathbf{y}_k - \mathbf{h}_k(\check{\mathbf{x}}_k, \mathbf{0}))$$



3. If a measurement is available:
    1. Compute Kalman Gain
    2. Compute error state
    3. Correct nominal state

$$\hat{\mathbf{x}}_k = \check{\mathbf{x}}_k + \delta\hat{\mathbf{x}}_k$$



3. If a measurement is available:
    1. Compute Kalman Gain
    2. Compute error state
    3. Correct nominal state
    4. Correct state covariance

$$\hat{\mathbf{P}}_k = (\mathbf{1} - \mathbf{K}_k\mathbf{H}_k)\check{\mathbf{P}}_k$$

Why use ES-KF:

1-Better performance compared to vanilla EKF and less computation compared to UKF+ more suitable for the algorithm approach lossy coupled approach.

2-Easy to Work with rotational quantities like Quaternion

The proposed algorithm:

STATE Estimation of the vehicle:

We estimate the state of the vehicle by combining data from 3 sensors GPS, IMU and Speedometer using ES-EKF to estimate position, velocity and orientation of the vehicle assuming a non-holonomic model of the car.

Why use IMU&GPS and Speedometer?

1-their error dynamics are completely different and uncorrelated that means of one of these failed

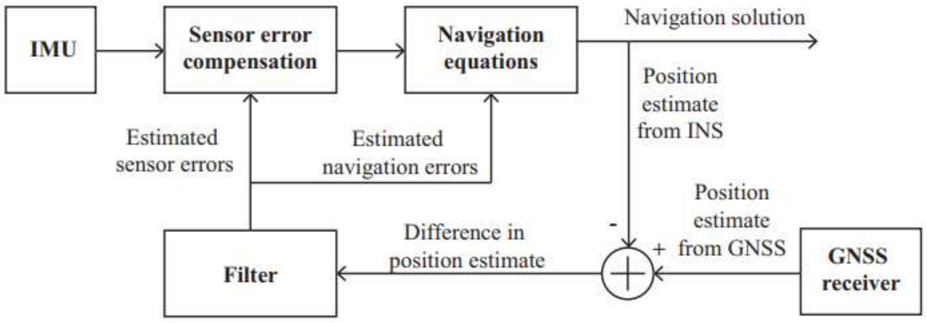
The others are unlikely to fail

2-IMU and speedometer provide smoothing of the GPS signal and continue to estimate the car state during GPS outage

3-GPS provides absolute positioning information to reduce imu drifts.

ES-EKF coupling:

In the algorithm design, we took the lossy coupled sensor fusion approach of GPS/INS and speedometer ES-EKF. Loosely coupled GPS/INS systems are because they are less computationally complex and more robust to errors than tightly coupled systems. In a loosely coupled GPS/INS system, the GPS and INS operate independently and their outputs are combined in a separate module. In contrast, in a tightly coupled system, the GPS and INS are tightly integrated, and their measurements are combined in a single Kalman filter. One reason why a loosely coupled approach is preferred is that GPS and INS measurements are subject to different types of errors. GPS measurements are affected by signal attenuation, multi-path interference, and signal obstructions, while INS measurements are affected by drift and bias in the accelerometers and gyroscopes. Because these errors have different characteristics, it is difficult to model them accurately in a single Kalman filter. In a loosely coupled system, the GPS and INS errors are modelled independently, and their estimates are combined in a separate module. This results in a more accurate estimate of the vehicle's position, velocity, and orientation. Another reason why a loosely coupled approach is preferred is that it is more computationally efficient than a tightly coupled system.


IMU
Sensor error compensation
Navigation equations
Navigation solution
Position estimate from INS
Estimated sensor errors
Estimated navigation errors
Position estimate from GNSS
-
+
Difference in position estimate
Filter
GNSS receiver

The main idea is that the INS is used as the backbone of the navigation system and constantly provides a 6-degree-of-freedom navigation solution. Whenever the GNSS-receiver produces a position estimate or the speedometer produces speed to estimate the difference between the position estimates of the two systems is calculated and the difference in speed estimate and used as the input for a filter that tries to estimate the errors of the INS navigation solution, as well as the errors in the IMU sensors. The estimated errors are then used to correct the navigation solution and to calibrate the sensors

Vehicle State:

-consists of position velocity and orientation quaternion :

$$\mathbf{x}_k = \begin{bmatrix} \mathbf{p}_k \\ \mathbf{v}_k \\ \mathbf{q}_k \end{bmatrix} \in R^{10}$$

-the model input will consist of specific forces and rotational rates from IMU:

$$\mathbf{u}_k \triangleq \begin{bmatrix} \mathbf{s}_k \\ \omega_k \end{bmatrix} \in R^6,$$

Here pk ∈ R3 [m], vk ∈ R3 [m/s], and qk ∈ R4 [−] denote the position, velocity, and attitude (quaternion representation) of the navigation system at time instant k. Further, sk ∈ R3 [m/s2] and ωk ∈ R3 [rad/s]

Motion model:

$$\mathbf{x}_k = f(\mathbf{x}_{k-1}, \mathbf{u}_k), \tag{2}$$

where

$$\begin{aligned}
\mathbf{p}_k &= \mathbf{p}_{k-1} + T_s\mathbf{v}_{k-1} + \frac{T_s^2}{2}\Big(\mathbf{R}_b^n(\mathbf{q}_{k-1})\mathbf{s}_k - \mathbf{g}\Big) && (3)\\
\mathbf{v}_k &= \mathbf{v}_{k-1} + T_s\Big(\mathbf{R}_b^n(\mathbf{q}_{k-1})\mathbf{s}_k - \mathbf{g}\Big) && (4)\\
\mathbf{q}_k &= \Big(\cos(0.5\,T_s\,\|\boldsymbol{\omega}_k\|)\mathbf{I}_4 + \frac{1}{\|\boldsymbol{\omega}_k\|}\sin(0.5\,T_s\,\|\boldsymbol{\omega}_k\|)\boldsymbol{\Omega}_k\Big)\mathbf{q}_{k-1} && (5)
\end{aligned}$$

and

$$\boldsymbol{\Omega}_k = \begin{bmatrix} 0 & [\boldsymbol{\omega}_k]_z & -[\boldsymbol{\omega}_k]_y & [\boldsymbol{\omega}_k]_x \\ -[\boldsymbol{\omega}_k]_z & 0 & [\boldsymbol{\omega}_k]_x & [\boldsymbol{\omega}_k]_y \\ [\boldsymbol{\omega}_k]_y & -[\boldsymbol{\omega}_k]_x & 0 & [\boldsymbol{\omega}_k]_z \\ -[\boldsymbol{\omega}_k]_x & -[\boldsymbol{\omega}_k]_y & -[\boldsymbol{\omega}_k]_z & 0 \end{bmatrix}. \tag{6}$$

Here, Ts denotes the sampling period of the data and Rn b (q) denotes the directional cosine matrix (rotation matrix) that rotates a vector from the body coordinate frame (b-frame) to the navigation coordinate frame (n-frame). Further, g denotes the gravity vector expressed in the navigation coordinate system. Noteworthy is that the attitude in the mechanized INS equations is described and propagated using the quaternion vector q. It is possible to use other attitude representations such as Euler angles or directional cosine matrices, but the quaternion representation is generally more numerically stable.

The linearized Error state motion model will be:

$$\delta\mathbf{x}_k \triangleq \begin{bmatrix} \delta\mathbf{p}_k \\ \delta\mathbf{v}_k \\ \boldsymbol{\epsilon}_k \end{bmatrix} \in R^9,$$

Where the position perturbation (error) δpk and velocity perturbation (error) δvk are defined as δpk = pk − pˆk and δvk = vk − vˆk, respectively. The attitude perturbation vector ϱk is defined as the small (Euler) angle sequence that rotates the attitude vector qˆk into qk "basically the error in

roll-Pitch and yaw" and if we added the estimation of bias error of IMU as shown in the figure of the system above assuming IMU measurement can be described as follows:

$$\tilde{\mathbf{u}}_k = \mathbf{u}_k - \delta\mathbf{u}_k + \mathbf{w}_k^{(1)}$$

Where u_k is the true measurement δuk bias error and wk1 denotes the measurement noise which is assumed to be additive white noise with covariance matrix Q1.

As the error dynamics of the state of the system and IMU bias error can be described as follows:

$$\delta\mathbf{x}_k = \mathbf{F}_{k-1}\delta\mathbf{x}_{k-1} + \mathbf{L}_{k-1}\underset{\uparrow}{\mathbf{n}_{k-1}}$$

$$\delta\mathbf{u}_k = \delta\mathbf{u}_{k-1} + \mathbf{w}_k^{(2)}.$$

Where Fk ,Lk are the Jacobian matrix , n_k is the measurement noise assumed to be gaussian with zero means =wk1:

$$\begin{aligned}\mathbf{n}_k &\sim \mathcal{N}(\mathbf{0}, \mathbf{Q}_k) \\ &\sim \mathcal{N}\left(\mathbf{0}, \Delta t^2 \begin{bmatrix} \sigma^2_{\text{acc}} & \\ & \sigma^2_{\text{gyro}} \end{bmatrix}\right)\end{aligned}$$

Now if we added the two models in a single model the complete model will be:

$$\mathbf{z}_k = \mathbf{F}(\mathbf{x}_k, \mathbf{u}_k)\mathbf{z}_{k-1} + \mathbf{G}_k(\mathbf{x}_k)\mathbf{w}_k,$$

where

$$\mathbf{z}_k \triangleq \begin{bmatrix} \delta\mathbf{x}_k \\ \delta\mathbf{u}_k \end{bmatrix} \in R^{15}, \quad \mathbf{w}_k \triangleq \begin{bmatrix} \mathbf{w}_k^{(1)} \\ \mathbf{w}_k^{(2)} \end{bmatrix} \in R^{12}$$

So the jacobian matrices can be calculated as follows “for full derivation please consider the references “:

$$\mathbf{F}(\mathbf{x}_k, \mathbf{u}_k) = \begin{bmatrix} \mathbf{I}_3 & T_s\mathbf{I}_3 & \mathbf{0}_{3,3} & \mathbf{0}_{3,3} & \mathbf{0}_{3,3} \\ \mathbf{0}_{3,3} & \mathbf{I}_3 & T_s[\mathbf{R}_b^n(\mathbf{q}_k)\mathbf{s}_k]_\times & T_s\mathbf{R}_b^n(\mathbf{q}_k) & \mathbf{0}_{3,3} \\ \mathbf{0}_{3,3} & \mathbf{0}_{3,3} & \mathbf{I}_3 & \mathbf{0}_{3,3} & -T_s\mathbf{R}_b^n(\mathbf{q}_k) \\ \mathbf{0}_{3,3} & \mathbf{0}_{3,3} & \mathbf{0}_{3,3} & \mathbf{I}_3 & \mathbf{0}_{3,3} \\ \mathbf{0}_{3,3} & \mathbf{0}_{3,3} & \mathbf{0}_{3,3} & \mathbf{0}_{3,3} & \mathbf{I}_3 \end{bmatrix}$$

and

$$\mathbf{G}_k(\mathbf{x}_k) = \begin{bmatrix} \mathbf{0}_{3,3} & \mathbf{0}_{3,3} & \mathbf{0}_{3,3} & \mathbf{0}_{3,3} \\ T_s\mathbf{R}_b^n(\mathbf{q}_k) & \mathbf{0}_{3,3} & \mathbf{0}_{3,3} & \mathbf{0}_{3,3} \\ \mathbf{0}_{3,3} & T_s\mathbf{R}_b^n(\mathbf{q}_k) & \mathbf{0}_{3,3} & \mathbf{0}_{3,3} \\ \mathbf{0}_{3,3} & \mathbf{0}_{3,3} & \mathbf{I}_3 & \mathbf{0}_{3,3} \\ \mathbf{0}_{3,3} & \mathbf{0}_{3,3} & \mathbf{0}_{3,3} & \mathbf{I}_3 \end{bmatrix},$$

The measurement model:

-The GPS measurement is modelled as follows assuming the GPS output data is in the same frame as the navigation model “NED frame”:

$$\begin{aligned}\mathbf{y}_k &= \mathbf{h}(\mathbf{x}_k) + \boldsymbol{\nu}_k \\ &= \mathbf{H}_k\mathbf{x}_k + \boldsymbol{\nu}_k = [\mathbf{1}\ \mathbf{0}\ \mathbf{0}]\,\mathbf{x}_k + \boldsymbol{\nu}_k \\ &= \mathbf{p}_k + \boldsymbol{\nu}_k\end{aligned}$$

$$\boldsymbol{\nu}_k \sim \mathcal{N}(\mathbf{0}, \mathbf{R}_{\text{GNSS}})$$

The speedometer measurement can be modelled as follows:

$$y_{k2} = h(x_k) + v_{k2} = H_{k_2} x_k + v_{k2}$$

$$H_{k2} = [0 \; BR_n^p \; 0]$$

Where RnP rotation matrix from navigation frame to vehicle frame as the speed of the speedometer is measured in that frame. where B=[1 0 0] as the vehicle only moves in the forward direction

Vehicle motion constrained:

Even though the GNSS-aided INS estimates the sensor biases the position error growth during the GNSS-signal outages is substantial and the navigation solution soon becomes useless. One way to reduce the error growth rate is to include a model of the vehicle's motion dynamics in the information fusion process. There are many ways to include motion dynamics models in the information process. One way is to define a set of motion constraints and enforce these constraints on the navigation solution using so-called constrained filtering.

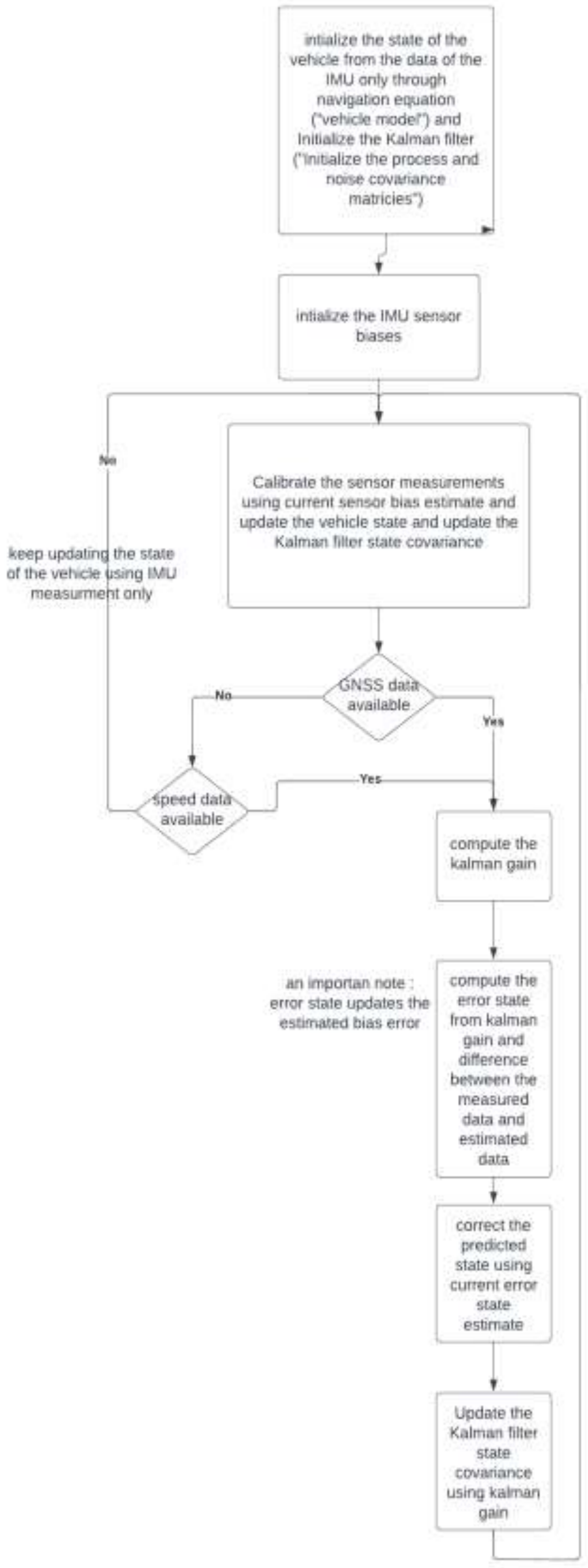
Two non-holonomic limitations control how a wheeled vehicle moves on a surface. A vehicle has no side slip and no motion in the direction normal to the road surface under ideal driving circumstances, and the speed in Thus, the vehicle coordinate frame's (p-frame) y- and z-axes should both be equal to zero.

Non-holonomic constraints can be modelled as follows:

$$y_{k3} = h(x_k) + v_{k2} = H_{k_3} x_k + v_{k3} = 0$$

$H_{k3} = [0 \; AR_n^p \; 0]$ Where $A = [0\ 1\ 0; 0\ 0\ 1]$

With the previous derivations in mind, the algorithm will work as follows:


intialize the state of the vehicle from the data of the IMU only through navigation equation ("vehicle model") and Initialize the Kalman filter ("Initialize the process and noise covariance matricies")
intialize the IMU sensor biases
Calibrate the sensor measurements using current sensor bias estimate and update the vehicle state and update the Kalman filter state covariance
No
keep updating the state of the vehicle using IMU measurment only
GNSS data available
No
Yes
speed data available
Yes
compute the kalman gain
an importan note : error state updates the estimated bias error
compute the error state from kalman gain and difference between the measured data and estimated data
correct the predicted state using current error state estimate
Update the Kalman filter state covariance using kalman gain

Progress:

-The algorithm is done in Matlab and tested with several car trips

-the next step is to convert it to C++ code and to be deployed in raspberry pi.

Emergency & breakdown call system

➢ Emergency call (Ecall)

In the event of a collision in which a car's airbags are deployed, ECall automatically contacts emergency services. It uses GPS to relay the time, your location, what type of car you're in, and what fuel it uses to the authorities, while a microphone in the car allows you to speak directly to call handlers when the system is activated.

All this allows ambulance, police, and fire crews to reach you as quickly as they can, having the required information.

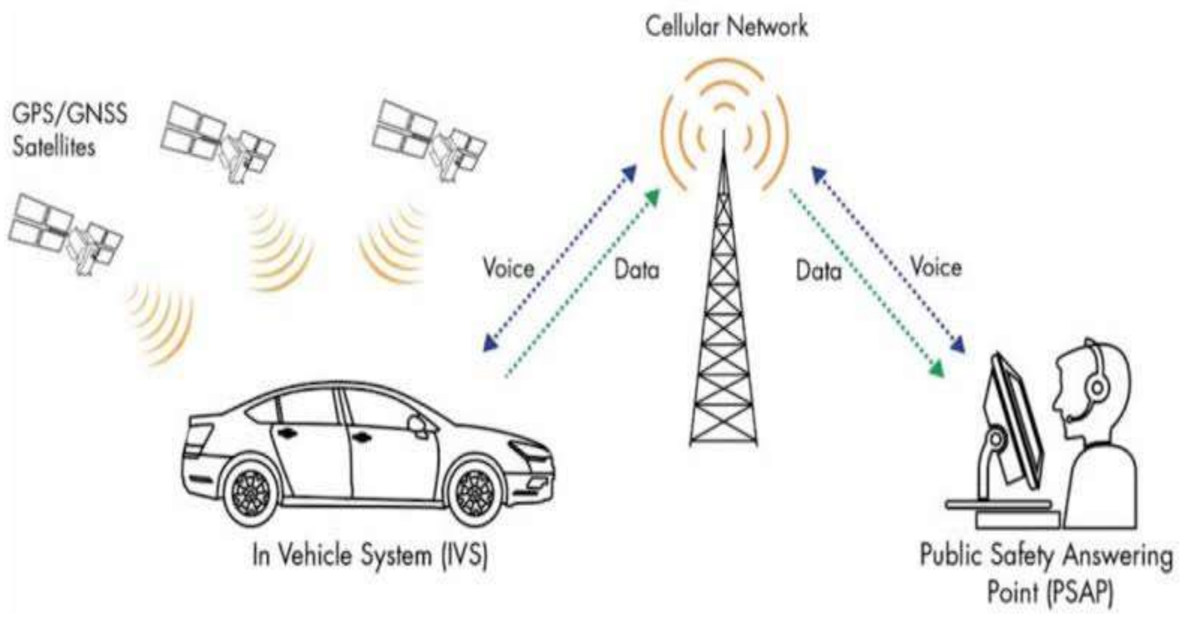
ECall is claimed to reduce emergency-service response times by 50% in the countryside, and 60% in built-up areas, potentially bringing accelerated medical attention and helping to prevent injuries from developing into something worse.

➢ Breakdown call (Bcall)

BCall is short for breakdown call. It refers to a service, which allows the driver to call local road assistance in case of a breakdown. At the same time, a BCall makes it possible for the call centre to gather information about the vehicle.

It can also be used in cases of having a medical emergency or seeing someone else in need of help.

➢ Ecall & Bcall work steps

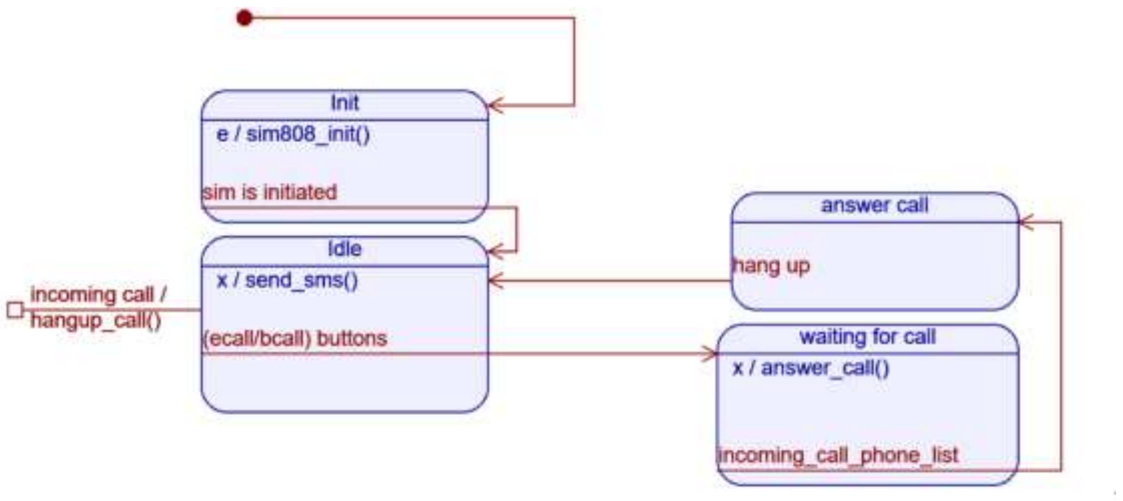


When the car’s airbags are deployed it is an indication of an accident or when the SOS button is pressed the vehicle sends a request to get its position the satellite sends the position in a NMEA protocol frame, and then the vehicle sends a minimum setup dada (MSD) to the public switched telephone network (PSTN) that direct the MSD to the public safety answering point (PSAP), when the PSAP receives the MSD it sends a call to the PSTN that direct the call to the vehicle which answers the call and by which the PSAP determine the situation of the passengers and sends the required help.

➢ SIM808 module

It is the used hardware to achieve the Ecall and Bcall systems in the vehicle, it is a GSM, GPRS, and GPS three-in-one function module, it supports GSM/GPRS Quad-Band network and combines GPS technology for satellite navigation.
 It is controlled by AT Command, supports 2G 3G 4G SIM Card, supports GPS NMEA protocol and uses UART protocol.

➢ State machine



when the SIM808 begins to work it is initiated to ensure that it is working correctly and ready to start the application, if it is initiated it starts the idle state, where it hangs up and refuses incoming calls to not disturb the function of the SIM808 in case of an accident or SOS button is pressed.

In the case of an accident or the SOS button is pressed the SIM808 sends MSD to the PSAP number and enters the waiting for a call state which hangs up all incoming calls but the PSAP numbers, it answers it and that turns it to the answer call state which exit it and goes back to the idle state when the call is finished.

➢ minimum setup data (MSD)

Minimum Set Data, MSD, is the data that should be collected and sent by the vehicle.

It has the following information:

- Vehicle registration
- Vehicle type
- Propulsion type
- Position data
- Time of the incident
- Trigger type: automatic – manual
- Position confidence: high confidence position – low confidence position

Socket Programming

Is a method to connect two nodes over a network to establish a means of communication between those two nodes. A node represents a computer or a physical device with an internet connection. A socket is an endpoint used for connecting to a node. The signals required to implement the connection between two nodes are sent and received using the sockets on each node respectively.

The processes that use a socket can reside on the same system or different systems on different networks. Sockets are useful for both stand-alone and network applications. Sockets allow you to exchange information between processes on the same machine or across a network, distribute work to the most efficient machine, and easily allow access to centralized data. Socket application program interfaces (APIs) are the network standard for TCP/IP. A wide range of operating systems support socket APIs.

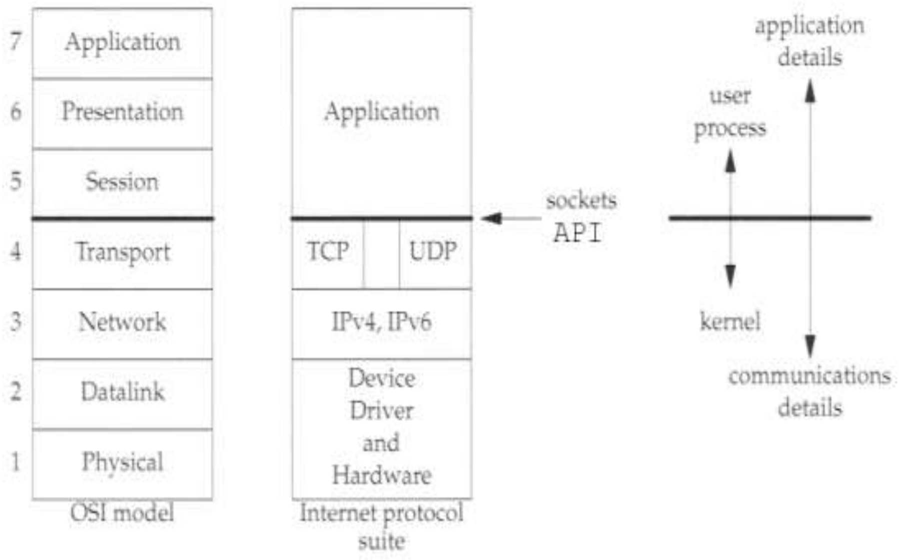
The socket programming will provide the ability of the implement analytics, streaming in binary, document collaboration, and so on. On the positive side, The IO can control the connection in sockets. For this reason, both the server and also client-side consist of IO libraries. It makes real-time application feasible in mobile devices in Systems.

Benefits of Socket Programming

Socket Programming consists of the essential communication of protocols, for example, TCP/UDP and also raw sockets, and so on., There various protocols thatare level-small communication when it is compared to HTTP/SMTP, etc. because of, becauseIt depends on the requirement mostly it perform half duplex data. Protocols allow both video and audio streaming. As well as, the First socket is a desire for server applications that are connectionless to the internet.

The Client / Server Model Two network applications can begin simultaneously, but it is impractical to require it. Therefore, it makes sense to design communicating network applications to perform complementary network operations in sequence, rather than simultaneously. The server executes first and waits to receive; the client executes second and sends the first network packet to the server. After theinitial contact, either the client or the server is capable of sending and receiving data.

Where is the socket programming interface about the protocol stack?



Features of a TCP connection:

- Connection Oriented Reliability
1. Handles lost packets
2. Handles packet sequencing
3. Handles duplicated packets
- Full Duplex

Feature	TCP	UDP
Connection status	Requires an established connection to transmit data (connection should be closed once transmission is complete)	Connectionless protocol with no requirements for opening, maintaining or terminating a connection
Guaranteed delivery	Can guarantee delivery of data to the destination router	Cannot guarantee delivery of data to the destination

Retransmission of data	Retransmission of lost packets is possible	No retransmission of lost packets
Error checking	Extensive error checking acknowledgement of data	Basic checking-error mechanism using checksums
Speed	Slower than UDP	Faster than TCP

Identifications of sockets

IP address

An IP address identifies a machine in an IP network and is used to determine the destination of a data packet.

The importance of IP addresses follows from the fact that each host on the Internet has a unique IP address. Thus, although the Internet is made up of many networks of networks with many different types of architectures and transport mediums, it is the IP address that provides a cohesive structure so that at least theoretically, (there are routing issues involved as well), any two hosts on the Internet can communicate with each other

Port number

Sockets are uniquely identified by anInternet address, end-to-end protocol, and port number. That is why when a socket is first created it is vital to match it with a valid IP address and a port number.

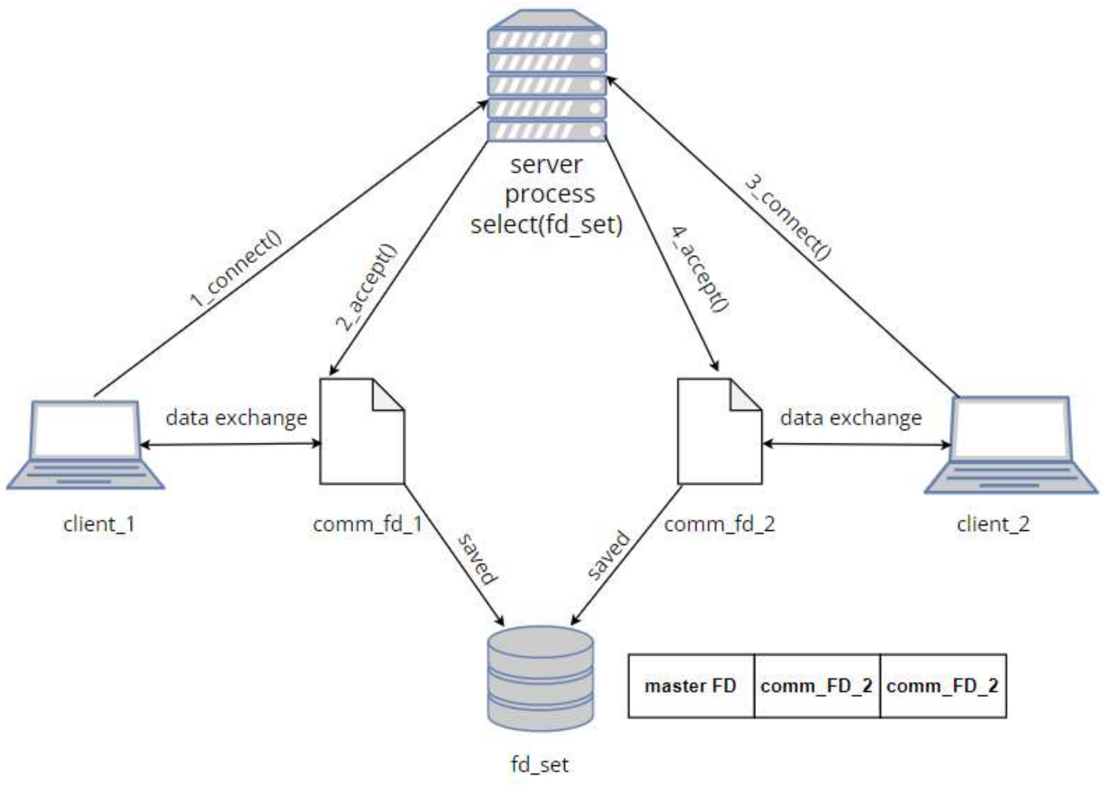
Ports are software objects to multiplex data between different applications. When a host receives a packet, it travels up the protocol stack and finally reaches the application layer. Now consider a user running an FTP client, a telnet client, and a web browser concurrently. To which application should the packet be delivered? Well, part of the packet contains a value holding a port number, and it is this number which determines to which application the packet should be delivered. So when a client first tries to contact a server, which port number should the client specify? For many common services, standard port numbers are defined.

Ports 0 – 1023, are reserved and servers or clients that you create will not be able to bind to these ports unless you have root privilege.

Ports 1024 – 65535 are available for use by your programs, but beware other network applications may be running and using these port numbers as well so do not make assumptions about the availability of specific port numbers.

How the connection is fully established

- When theServer boots up, it creates a connection socket (also called "master socket" using socket ())
- M is the mother of all Client Handles. Client handles are also called "data sockets".
- Once Client handles are created for each client, theServer carries out Communication (actual data exchange) with the client using theClient handle (and not M).
- M is only used to create new client handles. M is not used for data exchange with already connected clients.
- accept() is the system call used on theserver side to create client handles.
- In Linux Terminology, handles are called "file descriptors" which are just positive integer numbers. Client handles are called" communication file descriptors & and M is called "Master socket file descriptors.



Steps involved in establishing a socket on the *client* side

1. Create a socket with the socket() system call
2. Connect the socket to the address of the server using the connect() system call
3. Send and receive data. There are several ways to do this, but the simplest is to use the read() and write() system calls.

The steps involved in establishing a socket on the *server* side are as follows:

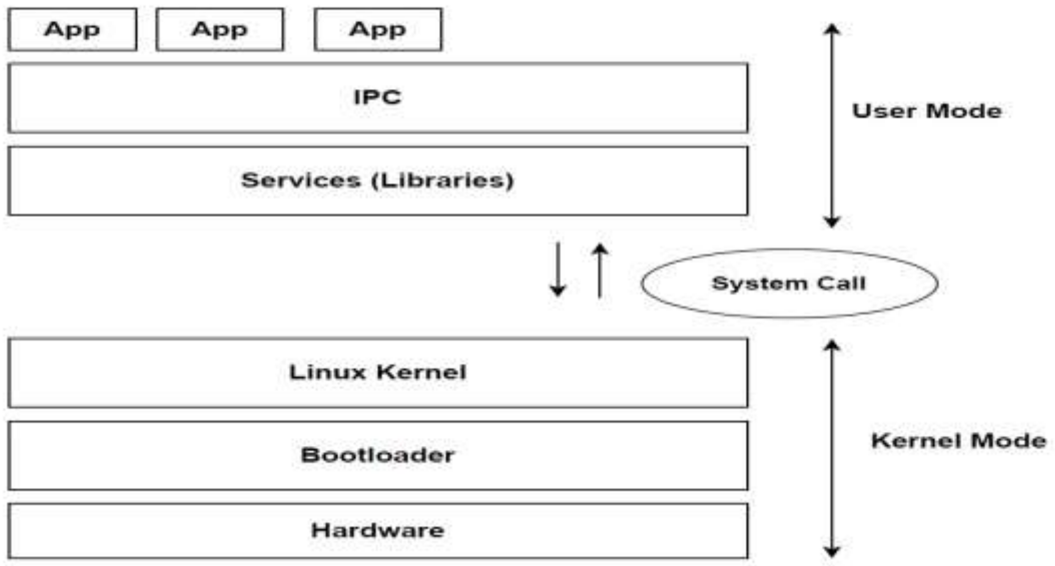
1. Create a socket with the socket() system call
2. Bind the socket to an address using the bind() system call. For a server socket on the Internet, an address consists of a port number on the host machine.
3. Listen for connections with the to-listen system call
4. Accept a connection with the accept() system call. This call typically blocks until a client connects with the server.
5. Send and receive data

Why Linux for telematics?

- Re-use & Features Extendibility: Many features, protocols and hardware are supported. Allow focusing on the added value of your product.
- Cost: are Development tools free too. But of course, deploying Linux costs time and effort.
- Time to market
- Full control: You decide when to update components in your system. No vendor lock-in.
- Quality: Your system is built on high-quality foundations (kernel, compiler, C-library, base utilities). Many Open-Source applications have good quality.

Generic Architecture for Embedded Linux:

- **Linux Levels:**
  - Linux uses only two different modes, kernel mode, and user mode.
  - The switch from user to kernel mode is made by the system call.
  - No intervention from the kernel unless the process for example tried to access memory that doesn't belong to it or did wrong calculations (divide by 0), so an exception will be raised and the Kernel here would take over this.



1. Kernel mode

- Embedded Linux = Linux kernel +embedded board
- Bootloader: The first piece of code running by the processor that can be modified by developers. It is responsible for:
    - ➢ Basic hardware initialization
    - ➢ Loading of an application binary, usually an operating system kernel, from flash storage, from the work, or from another type of non-volatile storage.
    - ➢ Possibly decompression of the application binary
    - ➢ Execution of the application

  Besides these basic functions, most bootloaders provide a shell or menu:
    - ➢ Menu to select the operating system to load
    - ➢ Shell with commands to load data from storage or network, inspect memory, perform hardware testing/diagnostics
- Linux kernel
    - ➢ Manages all HW/SW resources.
    - ➢ Provide a set of portable, architecture and hardware-independent APIs to allow user space applications and libraries to use the hardware resources.
    - ➢ Handle concurrent accesses and usage of hardware resources from different applications.
    - ➢ Some subsystems:

1. Device drivers/kernel modules: it is responsible for handling a hardware device to isolate the operating system and user application from details of this hardware device, without the need to modify the kernel. Device drivers are not limited to the support of hardware

devices, some device drivers perform complete software functions and don't perform any hardware-device action.

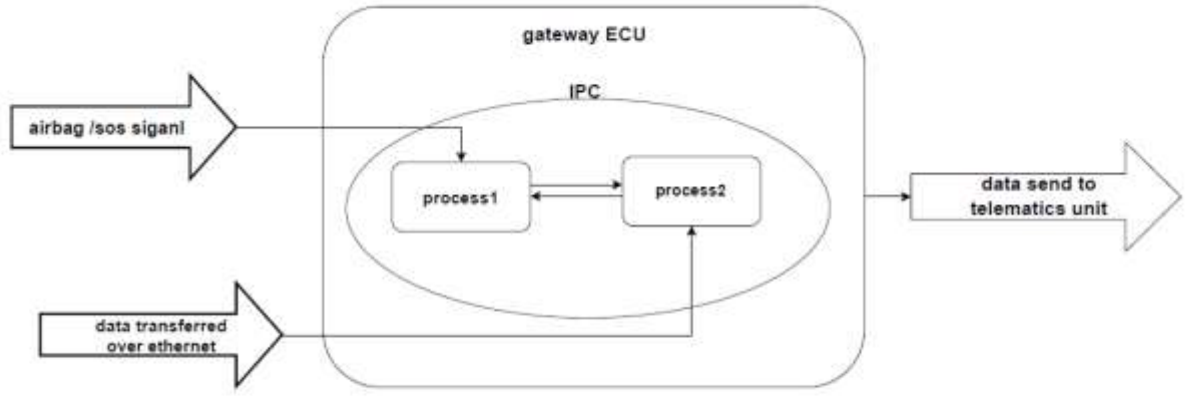
Examples:

- ➢ Random number generator: /dev/random
- ➢ Zeros generator: /dev/zero

2. Network interface driver: An "interface" means communication with an HW device through an interface bus like "eth0" not a file. Communication between the kernel and a network device driver is completely different from that used with device drivers. Instead of reading and writing, the kernel calls functions related to packet transmission.
3. Process Schedule: it is the core component of the kernel, which computes and decides when and for how long a process gets CPU time. The decision of which process to run depends on the priority of the process. Priorities are fundamentally classified into dynamic (applied dynamically by the kernel) and static (real-time processes by the user and the kernel does not change their priorities dynamically regardless of any historic behaviour) priorities.
4. File Systems: Allow stored data to be organized into directory structures and also have the job of linking other meta-information (owners, access rights, etc...).

2. User mode

- Linux runs processes /applications at the same time
- Processes need various resources to communicate, share data, and synchronize their execution to achieve desired results
- These are provided by the operating system's kernel as services called interposes communication (IPC)
- Linux provides different IPC methods:
  - ➢ Signals
  - ➢ Pipes and FIFOs
  - ➢ Message queue
  - ➢ Shared memory



Example: Gateway ECU

More than one processes run at the same time on one machine. Process_1 is responsible for receiving airbag/SOS signal & process_2 is responsible for receiving data transferred over Ethernet, both of them must be sent to the telematics unit and must synchronize data between them using IPC mechanisms.

REFERENCES:

1- **aided-navigation-gps-with-high-rate-sensors_compress by jay A. Farrell**

2- **inertial navigation system with geodetic applications by Christopher Jekel**

3- **Quaternion kinematics for the error-state Kalman filter by Joan Sol`**

4- **course 1: udemy-data fusion with linear Kalman filter**
https://www.udemy.com/course/data-fusion-with-linear-kalman-filter/

5- **course 2: udemy-advanced Kalman filtering and sensor fusion**
https://www.udemy.com/course/advanced-kalman-filtering-and-sensor-fusion/

6- **course 3: Coursera- state estimation and localization for self driving car**
https://www.coursera.org/learn/state-estimation-localization-self-driving-cars

7- **course 4: Udacity sensor fusion nano degree.**

8- Master MATLAB through Guided Problem Solving