Sensor Fusion

Sensor Fusion

The characteristics of sensor reveal the fact that each sensor can only perceive part of environment features, which is not sufficient enough to comprehensively represent the driving environment. Furthermore, confidence level of sensor data still remains to be determined. The techniques of sensor fusion and integration are concerned with improving the sensing capacity of system by synergistic using redundant and complementary information of multiple sensors, they are able to obtain more accurate environment features that are impossible to perceive with individual sensor. Sensor fusion has great application potential in ADAS system. However, when developing the actual multi-sensor platform, a series of problems arise and need to be resolved, including not only the conventional issues of sensor fusion and integration but also some special cruxes in ADAS design. Figure 1 shows the scheme of sensor fusion for sensing driving environment, it includes three fusion levels. With the common geometry and time frames, sensor fusion is implemented with the following levels:

1. **Registration level:**

   sensor data are registered to determine the correspondence among sensor data in both spatial and temporal dimensions.

2. **First fusion level:**

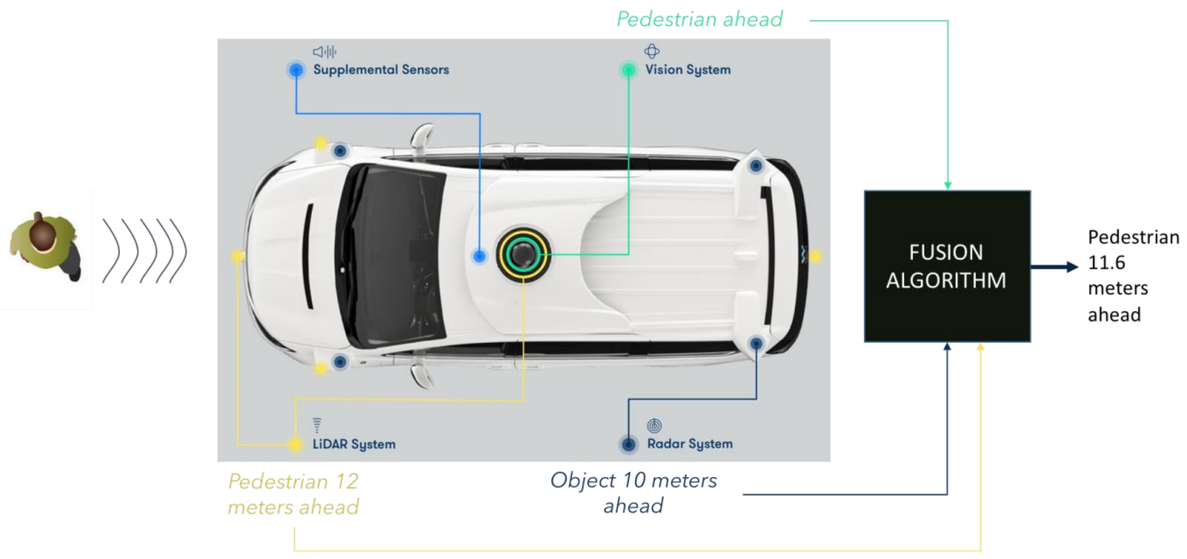
   Sensor data are fused to produce more accurate position-velocity information for detected objects, it uses the redundant position velocity information of different sensors (radar, lidar, and vision)

3. **Second level fusion:**

   complementary information are fused to infer new knowledge about the driving environment. The position velocity information of detected objects (from the first fusion level) and road geometry information (from vision) are fused to produce the primary perception map in which objects are characterized as being either stationary/moving or inside/outside the lane

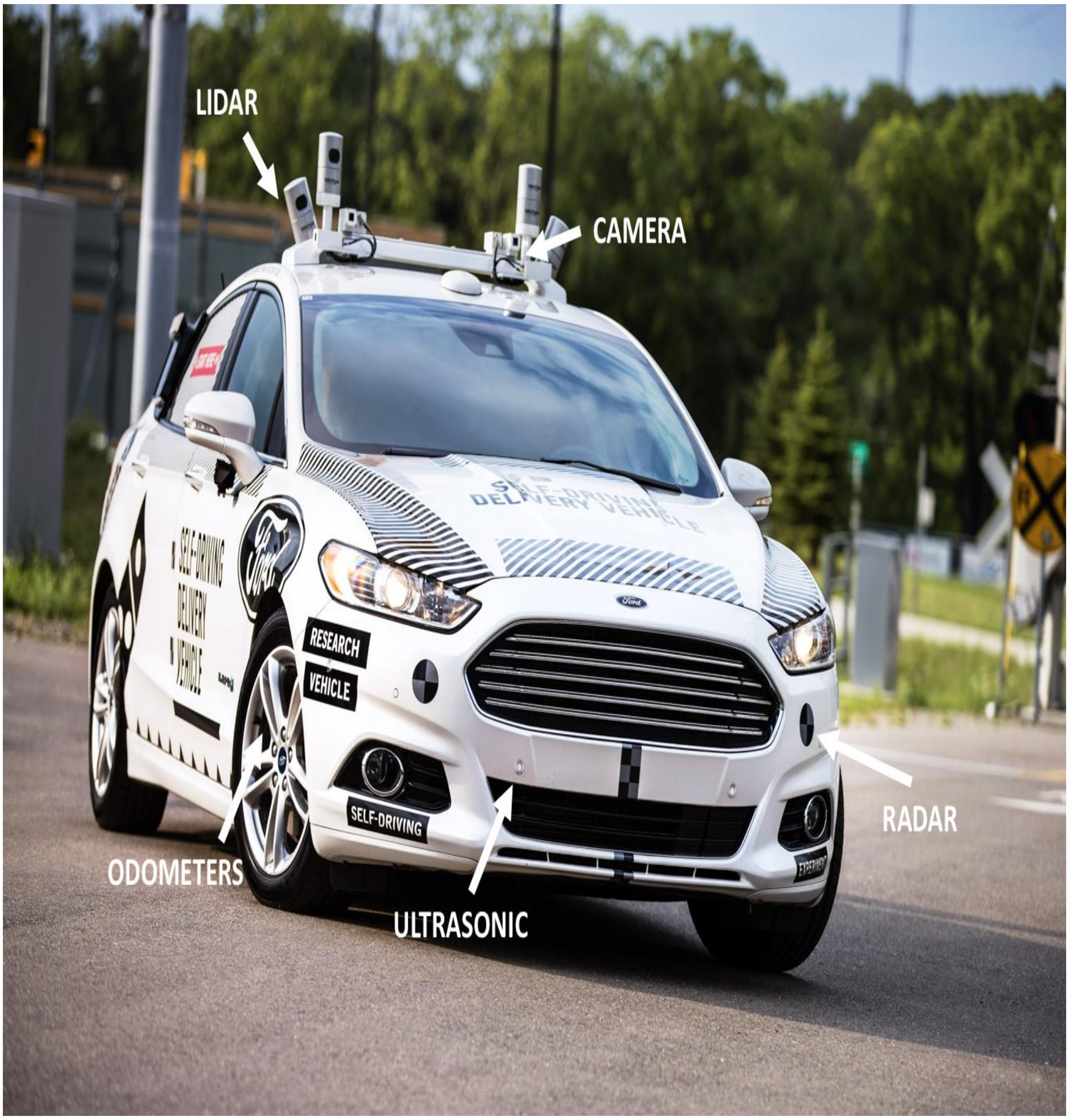
4. **Third level fusion:**

   object type information (from lidar) and object shape (from vision) are fused to classified the objects as objects interest or not in the perception map.



Sensors

The autonomous vehicle uses a large number of sensors to understand its environment, to locate and move.

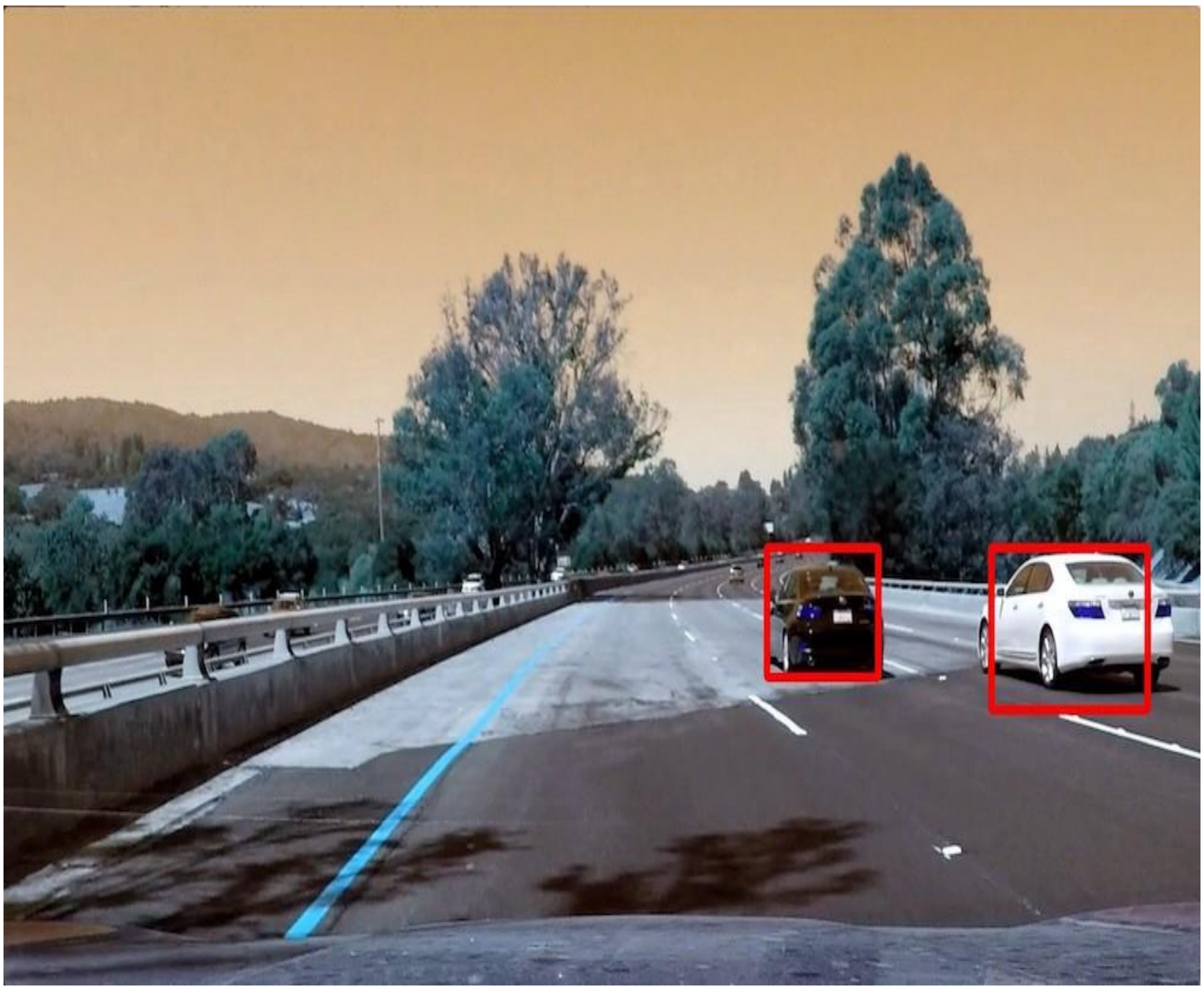


Different sensors on a self-driving car

Camera

The camera transcribes the driver’s vision. It is very often used to understand the environment with artificial intelligence by classifying roads, pedestrians, signs…

Vehicle detection using a camera



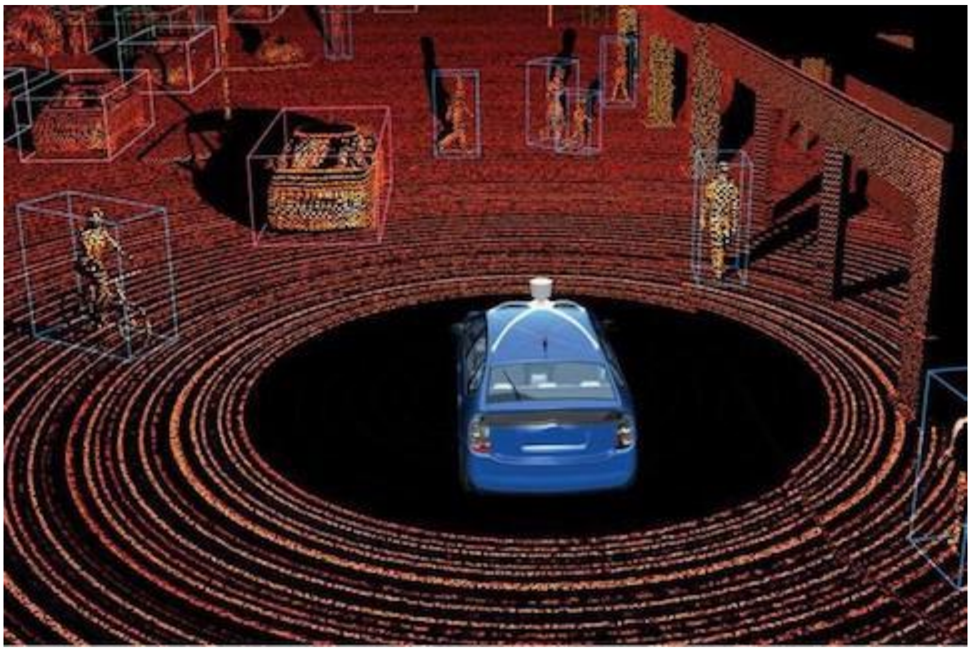
Radar (Radio Detection and Ranging)

The radar emits radio waves to detect objects within a radius of several meters. Radars have been in our cars for years to detect vehicles in blind spots or to avoid collisions. They have better results on moving objects than on static objects. Unlike other sensors that calculate the difference in position between two measurements, the radar uses the Doppler effect by measuring the change in the next wave frequency if the vehicle moves towards us or moves away. This is called radial velocity.

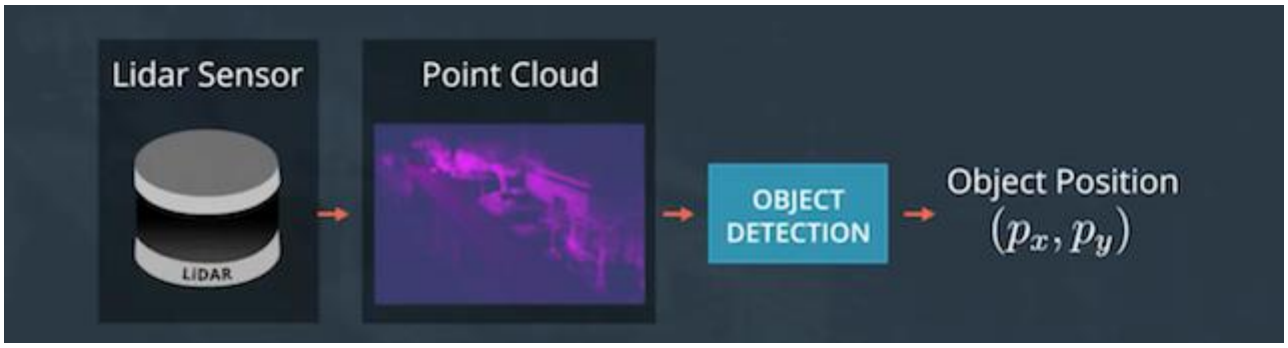
The radar can directly estimate a speed. It has a low resolution, it allows to know the position and speed of a detected object. On the other hand, it can not determine what is the object being sensed.

Lidar (Light Detection and Ranging)

The lidar uses infrared sensors to determine the distance to an object. A rotating system makes it possible to send waves and to measure the time taken for this wave to come back to it. This makes it possible to generate a points cloud of the environment around the sensor. A lidar can generate about 2 million points per second. This point cloud giving different 3D shapes, it is possible to make the classification of objects thanks to a Lidar.


Point cloud of the environment

It allows a great distance (100 to 300 m) to estimate the position of objects around him. Its size is however cumbersome since it exceeds the roof of the vehicles. Not mentioned, its price can go up to 100,000.



Position estimation with Lidar

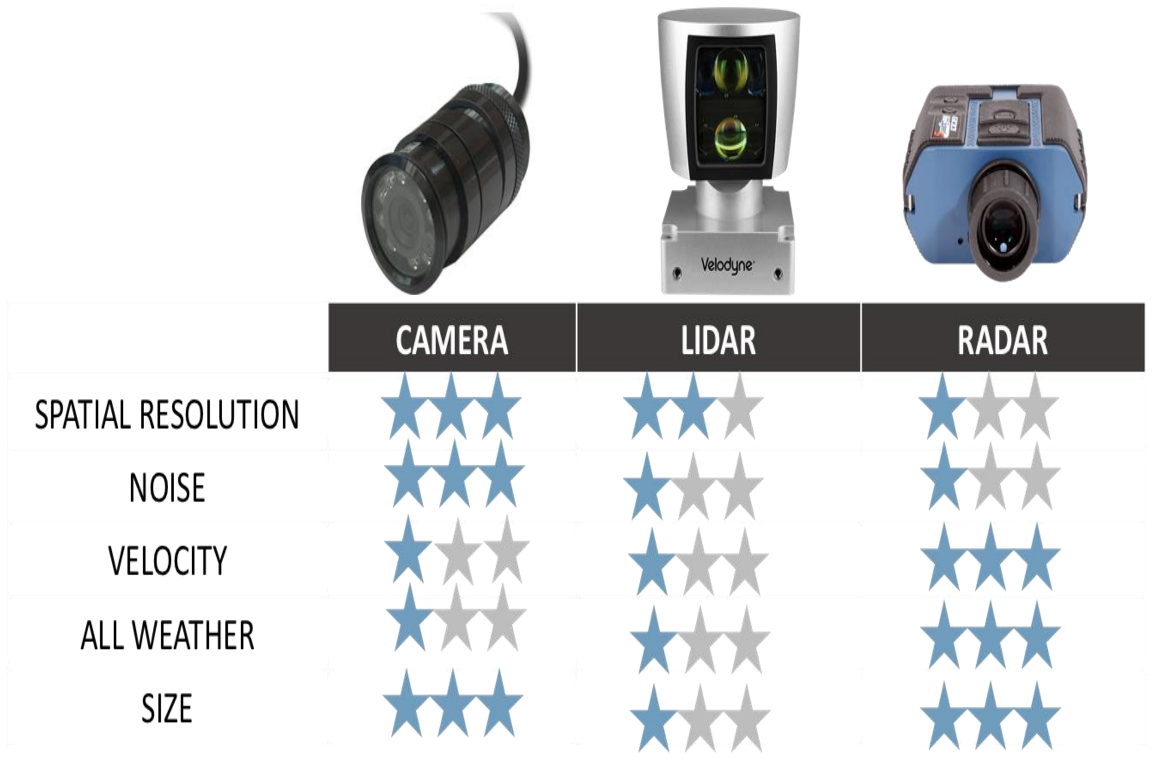
Ultrasonic sensors

Ultrasonic sensors are used to estimate the position of static vehicles, for example, to park. They are much cheaper but have a range of a few meters.

Odometric sensors

They make it possible to estimate the speed of our vehicle by studying the displacement of its wheels.

Benchmark



	CAMERA	LIDAR	RADAR
SPATIAL RESOLUTION	★★★	★★☆	★☆☆
NOISE	★★★	★☆☆	★☆☆
VELOCITY	★☆☆	★☆☆	★★★
ALL WEATHER	★☆☆	★☆☆	★★★
SIZE	★★★	★☆☆	★★★

Sensor Benchmark

Each of these sensors has advantages and disadvantages. The aim of **sensor fusion** is to use the advantages of each to precisely understand its environment. The camera is a very good tool for detecting roads, reading signs or recognizing a vehicle. The Lidar is better at accurately estimating the position of this vehicle while the Radar is better at accurately estimating the speed.

What are Sensor Fusion Algorithms?

Sensor fusion algorithms combine sensory data that, when properly synthesized, help reduce uncertainty in machine perception. They take on the task of combining data from multiple sensors each with unique pros and cons to determine the most accurate positions of objects. As we’ll see shortly, the accuracy of sensor fusion promotes safety by allowing other systems to respond in a timely and situation-appropriate manner.

Why Sensor Fusion Algorithms are Useful

In real-world systems, noise means that using just one sensor to identify the surrounding environment is not sufficiently reliable, which in the world of sensors translates to errors.

Let’s continue with our car example. Many modern cars have parking sensors that help you avoid hitting other cars when parking in tight spaces. If you’re about to hit something, some cars will also activate the brakes, protecting both you and whatever you’re about to hit. However, it can be dangerous to use information from a single parking sensor to stop the car if you’re too close to an object what if the sensor receives noisy information? What if the sensor malfunctions?

Now we see why cars with automatic brake activation, such as Teslas, use information obtained from multiple sensors. Sensor fusion sometimes relies on data from several of the same type of sensor (such as a parking sensor), known as *competitive configuration*. However, combining different types of sensors (such as fusing object proximity data with speedometer data) usually yields a more comprehensive understanding of the object under observation. Let’s take a closer look at this kind of setup, called *complementary configuration*.

In foggy weather, a radar sensor great for sensing the velocity and bearing of an object provides more precision than a LIDAR sensor can. In clear weather, you’ll want to rely more on a LIDAR sensor’s spatial resolution than that of a radar sensor. Sensors have various strengths and weaknesses, and a good algorithm takes multiple types of sensors into consideration. The data from these different sensors can be complementary.

Because different types of sensors have their own pros and cons, a strong algorithm will also give preference to some data points over others. For example, the speed sensor is probably more precise than the parking sensors, so you’ll want to give it more weight. Or perhaps the speed sensor is not very precise, and so you want to rely more on the proximity sensors. The variations are nearly infinite and depend on the specific use case.

How Sensor Fusion Algorithms Work

Sensor fusion algorithms process all inputs and produce output with high accuracy and reliability, even when individual measurements are unreliable.

Let's take a look at the equations that make these algorithms mathematically sound. A sensor fusion algorithm's goal is to produce a probabilistically sound estimate of an object's kinematic state. To calculate this state, an engineer uses two equations and two models: a *predict equation* that employs a *motion model,* and an *update equation* using a *measurement model*.

So, what are these motion and measurement models? The motion model deals with the dynamics of an object in our example, a car across increments of time. It says that the current state of the car draws from a range of values that depend on its state during the last time-step. The measurement model is concerned with the dynamics of the car's sensors. A range of values that depend on the current state of the car define the current measurement of, say, the radar.

To make sense of these models in the context of sensor fusion, let's look at two equations: one that predicts the state of the car, and one that continuously updates that prediction. The predict equation uses the previous prediction of the state (the range of possible state values calculated from the last round of predict-update equations) along with the motion model to predict the current state. This prediction is then updated (via the update equation) by combining the sensory input with the measurement model. We end up with a new range of possible state values, which turns into input for the new predict equation and again we calculate the next measurement to update the prediction.

This process allows us to use sensory input to predict where the car is and where it will be in the next time increment. In turn, this tells us when, and how fast, we need to stop to avoid a collision. For a more thorough explanation, we recommend this article detailing the calculus behind sensor fusion.

Algorithms Based on the Central Limit Theorem

Perhaps a more user-friendly name for the central limit theorem (CLT) is the law of large numbers: It states that as the sample size of whatever we are measuring increases, the average value of those samples will tend towards a normal distribution (a bell curve). A common example is rolling a die — the more rolls we measure, the closer the average value will be to 3.5, or the “true” average.

How does this relate to sensor fusion? Say we have two sensors, an ultrasonic sensor and an infrared sensor. The more samples we take of their readings, the more closely the distribution of the sample averages will resemble a bell curve and thus approach the set’s true average value. The closer we approach an accurate average value, the less noise will factor into sensor fusion algorithms.

Kalman Filter

A Kalman filter is an algorithm that takes data inputs from multiple sources and estimates unknown values, despite a potentially high level of signal noise. Often used in navigation and control technology, Kalman filters have the advantage of being able to predict unknown values more accurately than individual predictions using single methods of measurement.

Sound familiar? This is exactly what we went over in our car example. Because Kalman-filter algorithms are the most widely used application of sensor fusion and provide the foundation for understanding the concept itself, sensor fusion is often synonymous with Kalman filtering. One of the most common uses for Kalman filters is in navigation and positioning technology. Because Kalman filtering is recursive, we only need to know the car's last known position and speed to be able to predict its current and future state.

Algorithms Based on Bayesian Networks

Bayes' rule, which deals with probability, is the basis of the update equation described earlier that combines the motion and measurement models. Bayesian networks, also based on Bayes' rule, predict the likelihood that any one of several hypotheses is the contributing factor in a given event. K2, hill climbing, iterative hill climbing, and simulated annealing are some well-known Bayesian algorithms.

Convolutional Neural Networks

Convolutional neural-network-based methods can simultaneously process many channels of sensor data. From this fusion of such data, they produce classification results based on image recognition. For example, a robot that uses sensory data to tell faces or traffic signs apart relies on convolutional neural-network-based algorithms.

Working of Kalman Filter

```
#include "kalman_filter.h"



using Eigen::MatrixXd;

using Eigen::VectorXd;



KalmanFilter::KalmanFilter() {}



KalmanFilter::~KalmanFilter() {}



void KalmanFilter::Init(VectorXd &x_in, MatrixXd &P_in, MatrixXd
&F_in,
        MatrixXd &H_in, MatrixXd &R_in, MatrixXd &Q_in) {

        x_ = x_in;

        P_ = P_in;

        F_ = F_in;

        H_ = H_in;

        R_ = R_in;

        Q_ = Q_in;

}


void KalmanFilter::Predict() {



        // Predict the state

        x_ = F_ * x_;

        MatrixXd Ft = F_.transpose();

        P_ = F_ * P_ * Ft + Q_;

}
```

```
void KalmanFilter::UpdateRoutine(const VectorXd& y) {


    MatrixXd Ht = H_.transpose();

    MatrixXd S = H_ * P_ * Ht + R_;

    MatrixXd Si = S.inverse();


    // Compute Kalman gain

    MatrixXd K = P_ * Ht * Si;


    // Update estimate

    x_ = x_ + K * y;

    long x_size = x_.size();

    MatrixXd I = MatrixXd::Identity(x_size, x_size);

    P_ = (I - K * H_) * P_;

}


void KalmanFilter::Update(const VectorXd &z) {
    VectorXd z_pred = H_ * x_;

    VectorXd y = z - z_pred;


    UpdateRoutine(y);

}


void KalmanFilter::UpdateEKF(const VectorXd &z) {


    //// Recover explicitly status information

    float px = x_(0);

    float py = x_(1);

    float vx = x_(2);

    float vy = x_(3);


    // Map predicted state into measurement space

    double rho     = sqrt(px * px + py * py);
```

```
    double phi    = atan2(py, px);

    double rho_dot = (px * vx + py * vy) / std::max(rho, 0.0001);



    VectorXd z_pred(3);

    z_pred << rho, phi, rho_dot;



    VectorXd y = z - z_pred;



    // Normalize angle

    while (y(1) > M_PI) y(1) -= 2 * M_PI;

    while (y(1) < -M_PI) y(1) += 2 * M_PI;



    UpdateRoutine(y);



}
```