Advanced Kalman Filtering and Sensor Fusion

2D Vehicle Unscented Kalman Filter: Update Step

UKF Exercise 2

2D Vehicle UKF: Update Step Exercise

Overview

Implement the ***Unscented Kalman Filter Update Equations*** and the ***Non-Lidar Measurement Model***.

Step 1 (Setup)

- Open your last kalman filter file from the previous exercise which had the prediction step completed.
- Review the code that is already been written for the Update Steps as part of the UKF exercises.
    - ***handleLidarMeasurements()*** is called after the prediction step whenever LIDAR measurements have been made by the sensor. This function then sequentially calls the ***handleLidarMeasurement()*** function which is the main function you will modify to fuse the Lidar measurements ***one*** at a time.

```
void KalmanFilter::handleLidarMeasurements(const std::vector<LidarMeasurement>& dataset, const BeaconMap& map)
{
    // Assume No Correlation between the Measurements and Update Sequentially
    for(const auto& meas : dataset) {handleLidarMeasurement(meas, map);}
}

void KalmanFilter::handleLidarMeasurement(LidarMeasurement meas, const BeaconMap& map)
{
    if (isInitialised())
    {
        VectorXd state = getState();
        MatrixXd cov = getCovariance();

        // Implement The Kalman Filter Update Step for the Lidar Measurements in the
        // section below.
        // HINT: use the wrapAngle() function on angular values to always keep angle
        // values within correct range, otherwise strange angle effects might be seen.
        // HINT: You can use the constants: LIDAR_RANGE_STD, LIDAR_THETA_STD
        // ----------------------------------------------------------------------- //
        // ENTER YOUR CODE HERE

        BeaconData map_beacon = map.getBeaconWithId(meas.id); // Match Beacon with built in Data Association Id
        if (meas.id != -1 && map_beacon.id != -1)
        {

        }

        // ----------------------------------------------------------------------- //

        setState(state);
        setCovariance(cov);
    }
}
```

2D Vehicle UKF:
Update Step Exercise

Step 2 (Implement the Lidar Measurement Model)

- Modify the function ***lidarMeasurementModel()***

```
VectorXd lidarMeasurementModel(VectorXd aug_state, double beaconX, double beaconY)
{
    VectorXd z_hat = VectorXd::Zero(2);

    // ---------------------------------------------------------------------- //
    // ENTER YOUR CODE HERE

    // ---------------------------------------------------------------------- //

    return z_hat;
}
```

$$\hat{x}^a = [p_x \quad p_y \quad \psi \quad V \quad v_r \quad v_\theta]^T$$

$$\begin{bmatrix} \hat{r} \\ \hat{\theta} \end{bmatrix} = \begin{bmatrix} \sqrt{(L_x - \hat{p}_x)^2 + (L_y - \hat{p}_y)^2} + v_r \\ \arctan\left(\frac{L_y - \hat{p}_y}{L_x - \hat{p}_x}\right) - \hat{\psi} + v_\theta \end{bmatrix}$$

2D Vehicle UKF:
Update Step Exercise

Step 3 (Implement the Innovation Calculations)

- Modify the function ***handleLidarMeasurement()***
- Augment the State and Covariance Matrix with measurement noise
- Generate the Sigma Points (using augmented data) and Weights
- Transform Sigma Points with Lidar Measurement Model
- Calculate the mean of the transformed Sigma Points (Estimated Measurement)
- Calculate Innovation (Normalise Bearing Innovation)
- Calculate the covariance of the transformed Sigma Points (Innovation Covariance)

$$\hat{z}_k^{(i)} = h(\hat{x}_k^{(i)}, v_k^{(i)})$$

$$\hat{z}_k = \sum_{i=0}^{2n} W^{(i)} \hat{z}_k^{(i)}$$

$$x_k^a = \begin{bmatrix} \hat{x}_k^- \\ 0 \\ 0 \end{bmatrix}$$

$$P_k^a = \begin{bmatrix} P_k^- & 0 & 0 \\ 0 & \sigma_r^2 & 0 \\ 0 & 0 & \sigma_\theta^2 \end{bmatrix}$$

$$\nu = \begin{bmatrix} r \\ \theta \end{bmatrix}_{\text{meas}} - \begin{bmatrix} \hat{r} \\ \hat{\theta} \end{bmatrix}$$

$$S_k = \sum_{i=0}^{2n} W^{(i)} \left(\hat{z}_k^{(i)} - \hat{z}_k\right) \left(\hat{z}_k^{(i)} - \hat{z}_k\right)^T$$

2D Vehicle UKF:
Update Step Exercise

Step 4 (Calculate the Cross Covariance)

- Use the Original Sigma Points (Normalise Heading Angle Difference)
- Use the Transformed Measurement Sigma Points (Normalise Bearing Innovation)

$$P_{xz} = \sum_{i=0}^{2n} W^{(i)} \left(\hat{x}_k^{(i)} - \hat{x}_k^- \right) \left(\hat{z}_k^{(i)} - \hat{z}_k \right)^T$$

Step 5 (Implement the UKF Update Step Equations)

$$\begin{aligned} \mathbf{K}_k &= \mathbf{P}_{xz} \mathbf{S}_k^{-1} \\ \hat{x}_k^+ &= \hat{x}_k^- + \mathbf{K}_k \nu_k \\ \mathbf{P}_k^+ &= \mathbf{P}_k^- - \mathbf{K}_k \mathbf{S}_k \mathbf{K}_k^T \end{aligned}$$

2D Vehicle UKF:
Update Step Exercise

Step 6 (Run the Simulation in the following configurations)

- Profile 1 (Constant Speed/Heading, Zero Initial Conditions)
- Profile 2 (No-Zero Initial Conditions)
- Profile 3 (Constant Speed, Changing Headings)
- Profile 4 (Changing Speed, Changing Headings)
- Profile 5 (Profile 1 + LIDAR)
- Profile 6 (Profile 2 + LIDAR)
- Profile 7 (Profile 3 + LIDAR)
- Profile 8 (Profile 4 + LIDAR)

Step 7 (Compare the Simulation Results between the LKF, EKF, UKF with/without Lidar)