Technitute_

Advanced Kalman Filtering and Sensor Fusion

2D Vehicle Extended Kalman Filter: Prediction Step

EKF Exercise 1

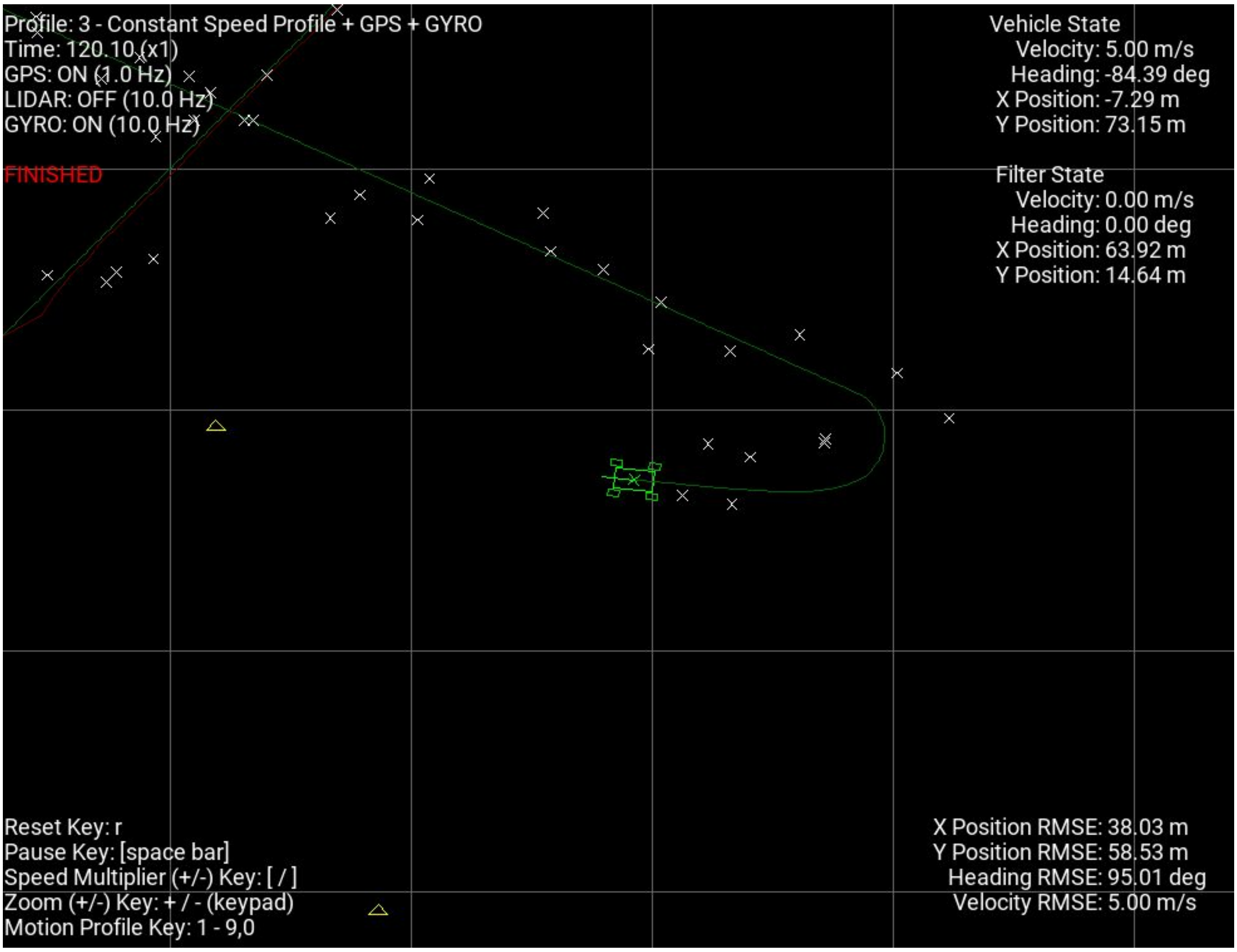
2D Vehicle EKF:
Prediction Step Exercise

Overview

Implement the ***Extended Kalman Filter Prediction Equations*** and the ***2D Non-Linear Vehicle Process Model***.

Step 1 (Setup)

- Open the c++ file "**kalmanfilter.cpp**" which will be the file used in this exercise (it should be a new copy of the file "**kalmanfilter_ekf_student.cpp**" file).
- Compile the run the simulation as is, using profile 3. See that the car starts at the origin (0,0) and moves at 5 m/s while performing a series of turns.
- Note: The GPS measurement model and state initialisation is copied from the previous.Linear Kalman Filter exercises as the filter equations and logic are the same, so the filter will start to run (but using GPS only and without an process model)



2D Vehicle EKF:
Prediction Step Exercise

```cpp
// ------------------------------------------------------------------------------- //
// Advanced Kalman Filtering and Sensor Fusion Course - Extended Kalman Filter
//
// ####### STUDENT FILE #######
//
// Usage:
// -Rename this file to "kalmanfilter.cpp" if you want to use this code.

#include "kalmanfilter.h"
#include "utils.h"

// ------------------------------------------------- //
// YOU CAN USE AND MODIFY THESE CONSTANTS HERE
constexpr double ACCEL_STD = 0.0;
constexpr double GYRO_STD = 0.01/180.0 * M_PI;
constexpr double INIT_VEL_STD = 10.0;
constexpr double INIT_PSI_STD = 45.0/180.0 * M_PI;
constexpr double GPS_POS_STD = 3.0;
constexpr double LIDAR_RANGE_STD = 3.0;
constexpr double LIDAR_THETA_STD = 0.02;
// ------------------------------------------------- //
```

Constants that you can modify and use

```cpp
void KalmanFilter::handleGPSMeasurement(GPSMeasurement meas)
{
    // All this code is the same as the LKF as the measurement model is linear
    // so the EKF update state would just produce the same result.
    if(isInitialised())
    {
        VectorXd state = getState();
        MatrixXd cov = getCovariance();

        VectorXd z = Vector2d::Zero();
        MatrixXd H = MatrixXd(2,4);
        MatrixXd R = Matrix2d::Zero();

        z << meas.x,meas.y;
        H << 1,0,0,0,0,1,0,0;
        R(0,0) = GPS_POS_STD*GPS_POS_STD;
        R(1,1) = GPS_POS_STD*GPS_POS_STD;

        VectorXd z_hat = H * state;
        VectorXd y = z - z_hat;
        MatrixXd S = H * cov * H.transpose() + R;
        MatrixXd K = cov*H.transpose()*S.inverse();

        state = state + K*y;
        cov = (Matrix4d::Identity() - K*H) * cov;

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

GPS Measurement is already done (Same as LKF)

2D Vehicle EKF:
Prediction Step Exercise

```
else
{
    VectorXd state = Vector4d::Zero();
    MatrixXd cov = Matrix4d::Zero();

    state(0) = meas.x;
    state(1) = meas.y;
    cov(0,0) = GPS_POS_STD*GPS_POS_STD;
    cov(1,1) = GPS_POS_STD*GPS_POS_STD;
    cov(2,2) = INIT_PSI_STD*INIT_PSI_STD;
    cov(3,3) = INIT_VEL_STD*INIT_VEL_STD;

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

State Initialization is automatically done on ***FIRST*** GPS measurement

$$\hat{x}_0 = \begin{bmatrix} p_{x\,gps} \\ p_{y\,gps} \\ 0 \\ 0 \end{bmatrix} \qquad \mathbf{P}_0 = \begin{bmatrix} \sigma^2_{gps} & 0 & 0 & 0 \\ 0 & \sigma^2_{gps} & 0 & 0 \\ 0 & 0 & \sigma^2_{\psi_0} & 0 \\ 0 & 0 & 0 & \sigma^2_{V_0} \end{bmatrix}$$

NOTE:
The code assumes that the state vector has the following form!!

$$\hat{x} = \begin{bmatrix} p_x \\ p_y \\ \psi \\ V \end{bmatrix}$$

2D Vehicle EKF:
Prediction Step Exercise

Step 2 (Implement the Prediction Step (Process Model))

- Modify the ***predictionStep()*** function
- Use the time step and gyroscope input. Assume zero acceleration.
- Normalise the heading angle to +/- PI after prediction!

```
void KalmanFilter::predictionStep(GyroMeasurement gyro, double dt)
{
    if (isInitialised())
    {
        VectorXd state = getState();
        MatrixXd cov = getCovariance();

        // Implement The Kalman Filter Prediction Step for the system in the
        // section below.
        // HINT: Assume the state vector has the form [PX, PY, PSI, V].
        // HINT: Use the Gyroscope measurement as an input into the prediction step.
        // HINT: You can use the constants: ACCEL_STD, GYRO_STD
        // HINT: use the wrapAngle() function on angular values to always keep angle
        // values within correct range, otherwise strange angle effects might be seen.
        // ----------------------------------------------------------------------- //
        // ENTER YOUR CODE HERE

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

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

$$\begin{bmatrix} p_x \\ p_y \\ \psi \\ V \end{bmatrix}_k = \begin{bmatrix} p_x \\ p_y \\ \psi \\ V \end{bmatrix}_{k-1} + \Delta t \begin{bmatrix} V_{k-1}\cos(\psi_{k-1}) \\ V_{k-1}\sin(\psi_{k-1}) \\ \dot{\psi}_k \\ a_k \end{bmatrix}$$

0

gyro.psi_dot

$$a_k \sim N(0, \sigma^2_{accel})$$

```
double psi_new = wrapAngle(psi + dt * gyro.psi_dot);
```

2D Vehicle EKF: Prediction Step Exercise

Step 3 (Implement the Prediction Step (Covariance))

- Modify the ***predictionStep()*** function
- Calculate the Jacobian State matrix (F matrix)
- Define the Q matrix for gyroscope noise and acceleration noise (Assume zero positional noise)
- Implement the covariance prediction step

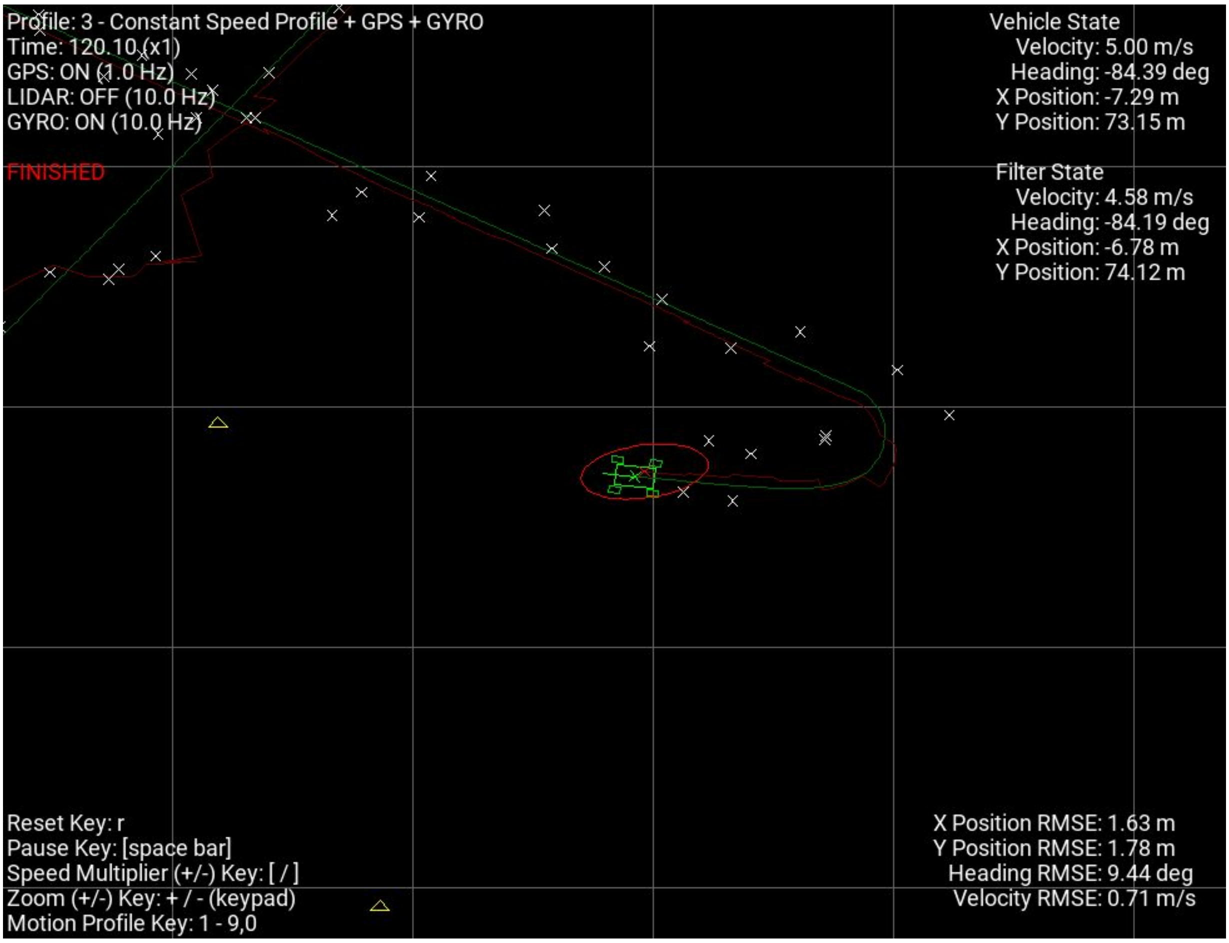
$$\mathbf{F}_k = \nabla \mathbf{f}_x = \begin{bmatrix} 1 & 0 & -\Delta t V_{k-1} \sin(\psi_{k-1}) & \Delta t \cos(\psi_{k-1}) \\ 0 & 1 & \Delta t V_{k-1} \cos(\psi_{k-1}) & \Delta t \sin(\psi_{k-1}) \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad \mathbf{Q} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & \Delta t^2 \sigma_{gyro}^2 & 0 \\ 0 & 0 & 0 & \Delta t^2 \sigma_{accel}^2 \end{bmatrix}$$

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

2D Vehicle EKF: Prediction Step Exercise

Step 4 (Run the Simulation)

- Check out how the simulation runs with profiles 1 - 4.



2D Vehicle EKF:
Prediction Step Exercise

Step 5 (You can quickly compare to the LKF Solution for the Error Statistics)

Note: There will be slight differences in the results due to tuning, etc.

2D Vehicle EKF:
Prediction Step Exercise

Step 6 (Play around with your code and make sure it is working as expected)

- Is this filter performing better than the LKF? Is that to be expected? Is it more robust?
- Can you think of a better way to initialise the state? (i.e how to initialize velocity and heading without assume them to be zero or a known constant)