COMP0130: Robot Vision and Navigation
Coursework 1: Integrated Navigation for a Robotic Lawnmower

Group S
Hongzhan Yu*, Yicheng Gong*, Heshui He*
hongzhan.yu.22@ucl.ac.uk
yicheng.gong.22@ucl.ac.uk
zceehex@ucl.ac.uk

January 31, 2023

1 Introduction

We proposed two different solutions in this coursework, the first is open-loop loosely coupled Kalman filter and the other is closed-loop loosely coupled Kalman filter. In the first section, we introduced the single GNSS Kalman filter solution, which includes outlier detection and how did we initialise the state and covariance by least square method. In the second section, we introduced Dead Reckoning-only solution which uses the result of our Gyro-Magnetometer Kalman filter corrected heading solution. In the third section, we introduced open-loop loosely coupled Kalman filter framework that simply integrated the result of GNSS and Dead Reckoning. In the forth section, we feed the correction back to Dead Reckoning and which formed a closed-loop frame work.

2 GNSS Kalman Filter Solver

GNSS receiver gives measurements of pseudo-ranges and pseudo-range rates corresponding to the 8 satellites. According to the two measurement sets, using GNSS least squares approach to initialize the Kalman filter state estimates. For every time stamp/epoch, the pseudo-ranges and pseudo-range rates are performed outlier detection and remove the outlier measurements. The retained measurements of pseudo-ranges and pseudo-range rates are input to the Kalman Filter to generate filtered state estimates, involving user position and velocity solution.
In this Chapter, we introduce the approach of GNSS least squares, mechanism of outlier detection and application of Kalman filter to GNSS navigation. The Flow chart of the GNSS Kalman Filter Solver is presented in Figure 1.

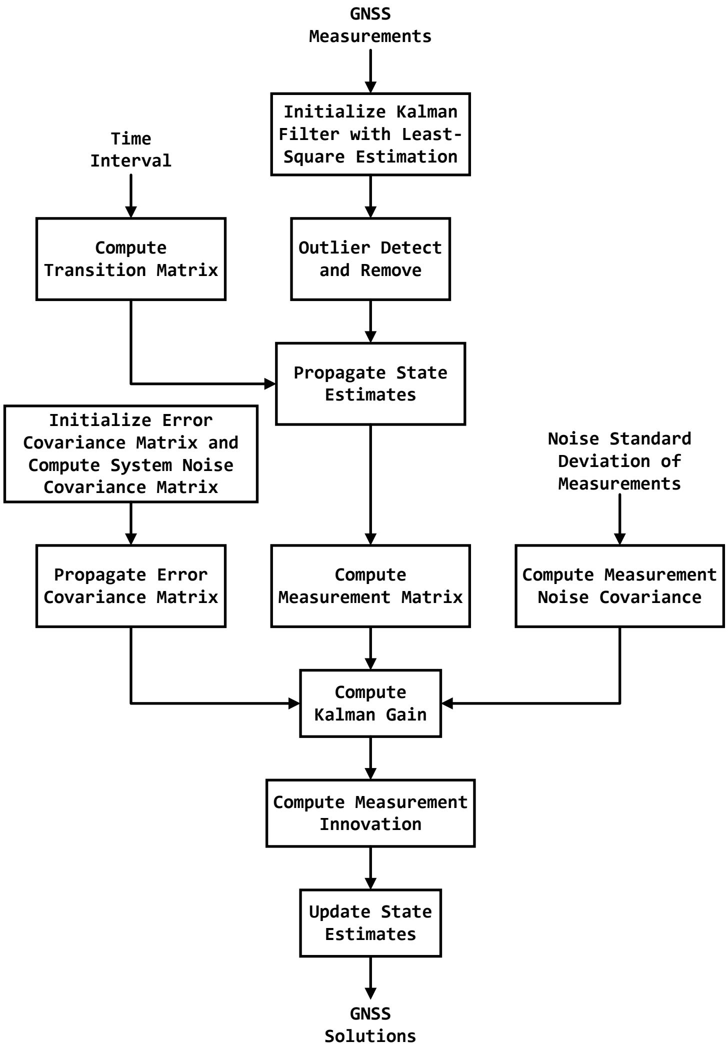


Figure 1: Flow Chart of GNSS Kalman Filter Solver

2.1 GNSS Least Squares Solution Solver

The Least Square Estimation is only used in Initialising the Kalman state and covariance, and it also assists the outlier detection process.
Cartesian ECEF positions and velocities of satellites, $\hat{r}_{ej}$, $\hat{v}_{ej}$, at epoch k are computed by using function ***Satellite_position_and_velocity.m***. Considering the input of user state at epoch k, $x_k^- = \left[\hat{r}_{ea}^-, \hat{v}_{ea}^-, \delta\hat{\rho}^-, \delta\hat{\dot{\rho}}^-\right]^T$, where $\hat{r}_{ea}^-$ is the predicted user position, $\hat{v}_{ea}^-$ is the predicted user velocity, $\delta\hat{\rho}^-$ is the predicted receiver clock offset, $\delta\hat{\dot{\rho}}^-$ is the predicted receiver clock drift, applying the recursion to compute the the predicted ranges and predicted range rates, $\hat{r}_{aj}^-$ and $\hat{\dot{r}}_{aj}^-$, for each satellite. The recursion is resolved by initially computing the range with the C_e^I set to the identity matrix, then using this range to update the C_e^I and then recomputing the range.
In the recursion, predicted ranges are computed by Equation (1) [1].

$$\hat{r}_{aj}^- = \sqrt{[C_e^I\hat{r}_{ej} - \hat{r}_{ea}^-]^T[C_e^I\hat{r}_{ej} - \hat{r}_{ea}^-]} \tag{1}$$

Where C_e^I is the Sagnac effect compensation matrix [2], given by

$$C_e^I(\hat{r}_{aj}^-) \approx \begin{bmatrix} 1 & \omega_{ie}\hat{r}_{aj}^-/c & 0 \\ -\omega_{ie}\hat{r}_{aj}^-/c & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2}$$

Where ω_{ie} is the earth rotation rate, and c is the speed of light.
Then compute the line-of-sight unit vector u_{aj} from the approximate user position to each satellite. The equation is given by Equation (3)[3].

$$u_{aj} = \frac{C_e^I\hat{r}_{ej} - \hat{r}_{ea}^-}{\hat{r}_{aj}^-} \tag{3}$$

The predicted ranges rates are computed by Equation (4)[4].

$$\hat{\dot{r}}_{aj}^- = u_{aj}^T[C_e^I(\hat{v}_{ej} + \Omega_{ie}\hat{r}_{ej}) - (\hat{v}_{ea}^- + \Omega_{ie}\hat{r}_{ea}^-)] \tag{4}$$

Where Ω_{ie} is the skew symmetric matrix of the Earth rotation rate [5], $\Omega_{ie} = \begin{bmatrix} 0 & -\omega_{ie} & 0 \\ \omega_{ie} & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$.
Measurement innovation vector δz^- is formulated by Equation (5)[6], where $\tilde{\rho}_k^j$, $\tilde{\dot{\rho}}_k^j$ are the

pseudo ranges and the pseudo range rates of satellite j at epoch k, N is the number of satellite.

$$\delta z^{-} = \begin{bmatrix} \tilde{\rho}_k^1 - \hat{r}_{a1}^{-} - \delta\hat{\rho}^{-} \\ \vdots \\ \tilde{\rho}_k^j - \hat{r}_{aj}^{-} - \delta\hat{\rho}^{-} \\ \vdots \\ \tilde{\rho}_k^N - \hat{r}_{aN}^{-} - \delta\hat{\rho}^{-} \\ \tilde{\dot{\rho}}_k^1 - \hat{\dot{r}}_{a1}^{-} - \delta\hat{\dot{\rho}}^{-} \\ \vdots \\ \tilde{\dot{\rho}}_k^j - \hat{\dot{r}}_{aj}^{-} - \delta\hat{\dot{\rho}}^{-} \\ \vdots \\ \tilde{\dot{\rho}}_k^N - \hat{\dot{r}}_{aN}^{-} - \delta\hat{\dot{\rho}}^{-} \end{bmatrix}_{2N\times 1} \tag{5}$$

Measurement matrix H_G is formulated by Equation (6)[7].

$$H_G = \begin{bmatrix} -u_{a1,x} & -u_{a1,y} & -u_{a1,z} & 0 & 0 & 0 & 1 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ -u_{aj,x} & -u_{aj,y} & -u_{aj,z} & 0 & 0 & 0 & 1 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ -u_{aN,x} & -u_{aN,y} & -u_{aN,z} & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -u_{a1,x} & -u_{a1,y} & -u_{a1,z} & 0 & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & -u_{aj,x} & -u_{aj,y} & -u_{aj,z} & 0 & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & -u_{aN,x} & -u_{aN,y} & -u_{aN,z} & 0 & 1 \end{bmatrix}_{2N\times 8} \tag{6}$$

Using unweighted least-squares to update the user state [8],

$$\hat{x}_k^{+} = \hat{x}_k^{-} + (H_G^T H_G)^{-1}\delta z^{-} = \begin{bmatrix} \hat{r}_{ea}^{+} & \hat{v}_{ea}^{+} & \delta\hat{\rho}^{+} & \delta\hat{\dot{\rho}}^{+} \end{bmatrix}^T \tag{7}$$

Perform iteration above until it reach maximum iteration number or converge. Hence, the GNSS least squares algorithm can be summarized as Algorithm 1.

2.2 Outlier detection

After obtaining the GNSS Least Squares solution for each epoch, a residual-based outlier detection is performed to reject the outlying measurements.
For epoch a, compute the residuals vector using Equation (8)[9].

$$v = [H_G(H_G^T H_G)^{-1} - I_N]\delta z^{-} \tag{8}$$

Where I_N is the $N \times N$ identity matrix, where N is the number of satellites.
Compute the residuals covariance matrix using Equation (9)[10].

$$C_v = [I_N - H_G(H_G^T H_G)^{-1} H_G^T]\sigma_p^2 \tag{9}$$

Algorithm 1: GNSS Least Squares Solution

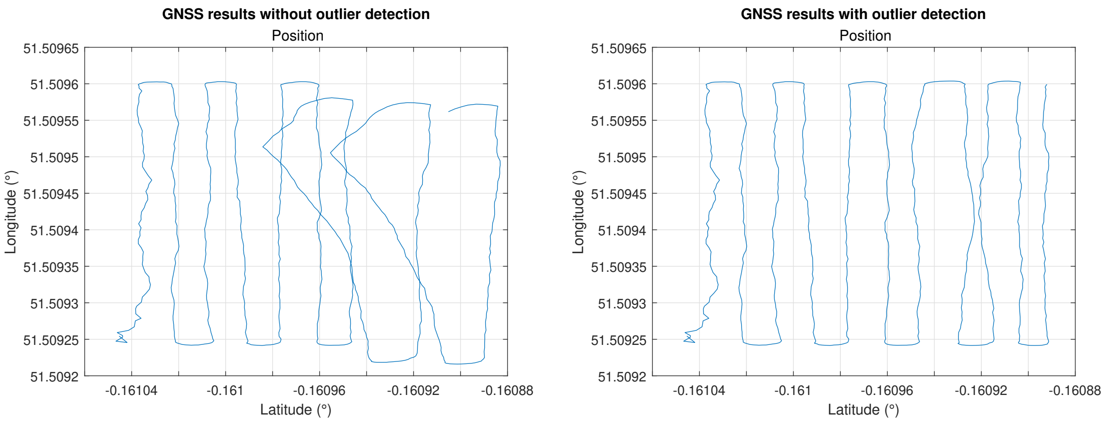
Input: $\tilde{\rho}$, $\dot{\tilde{\rho}}$, x_k^-
Output: x_{k+1}^-, H_G, δz^-

Assign satellite number as N
while $i <$ *maximum iteration* **do**
  for $j < N$ **do**
    Compute the Cartesian ECEF positions and velocities of satellites, $\hat{r}_{ej}$, $\hat{v}_{ej}$.
  Set the Sagnac effect compensation matrix to identity matrix, $C_e^I = I_3$.
  for $j < N$ **do**
    if $i <= 1$ **then**
      Initialize the ranges from the approximate user position to each satellite $\hat{r}_{aj}^-$.
    Update $C_e^I(\hat{r}_{aj}^-)$.
    Compute the ranges from the approximate user position to each satellite, $\hat{r}_{aj}^-$.
    Compute the line-of-sight unit vector from the approximate user position to each satellite u_{aj}.
    Compute the predicted range rates from the approximate user velocity to each satellite $\hat{\dot{r}}_{aj}^-$.
  Formulate measurement innovation vector δz^-.
  Formulate measurement matrix, H_G.
  Update the user state, $\hat{x}_k^+$, and assign it as input of next iteration.
Assign user state as input of epoch $k+1$, $\hat{x}_{k+1}^- = \hat{x}_k^+$.

Where σ_p is the measurement error standard deviation. Then, compute the normalized residuals and compare each with outlier detection threshold T [11].

$$|v_j| > \sqrt{C_{vjj}}T \tag{10}$$

Where C_{vjj} is the j^{th} diagonal element of C_v.



(a) GNSS Without Outlier Detection

(b) GNSS With Outlier Detection

Figure 2: GNSS Result Comparison

CONTINUED

Observe the difference in Figure 2, when the outlier is used in the Kalman loop, it can greatly improve the result . The detailed Kalman loop can be found in next section.
Note down any outliers detected. If the detector detects outliers, recalculate the GNSS least squares solution at that epoch without the measurement that had the largest residual. Repeat this until no outlier be detected, and return the remaining measurements. The outlier detection algorithm can be summarized as Algorithm 2.

Algorithm 2: Outlier Detection

Input: x_k^-, k, $\tilde{\rho}$, $\tilde{\dot{\rho}}$
Output: satellite name list, s, pseudo ranges at epoch k, $\tilde{\rho}_k$, pseudo range rates at epoch k, $\tilde{\dot{\rho}}_k$

while *True* **do**
    Perform GNSS least squares solution algorithm in Algorithm (1), obtaining the measurement matrix H_G and measurement innovation vector δ_z^-.
    Compute the residuals vector, v.
    Compute the residuals covariance matrix, C_v.
    Compute the normalized residuals $|v_j|$ and compare with outlier detection threshold.
    Note down the index of outliers j, and the corresponding residuals v_j.
    Remove the specified measurement of pseudo ranges and pseudo range rates at epoch k which has largest residual.
    if *No outlier detected* **then**
        Break

2.3 Kalman Filter

As an error minimum variance estimation algorithm, Kalman filter can perform optimal unbiased estimation of the state (suitable for linear system with Gaussian noise distribution in our case).
Initialize the Kalman Filter state vector estimate x_0^+ by applying the GNSS Least Squares algorithm with inputs of zero state, pseudo ranges and pseudo range rates at first epoch. After removing the outliers, it returns the state estimates, x_0^+. Then initialize the error covariance matrix as follows

$$P_0^+ = \begin{bmatrix} 100 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 100 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 100 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.01 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.01 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.01 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 10^{10} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 40000 \end{bmatrix} \tag{11}$$

For each epoch, the prior state is generated from the previous state estimates,

$\hat{x}_{k-1}^{+} = \left[\hat{r}_{ea}^{-}, \hat{v}_{ea}^{-}, \delta\hat{\rho}^{-}, \delta\hat{\dot{\rho}}^{-}\right]^{T}$. The prior clock offset and drift solutions are used to predict the current clock offset and clock drift, and the prior user position and velocity can be used to predict the current user position and velocity. The outlier are detected by applying Algorithm 2 and obtaining the corresponding satellite name list, s, pseudo ranges, $\tilde{\rho}_k$, and pseudo range rates, $\tilde{\dot{\rho}}_k$. The current pseudo range and pseudo range rate measurements are used to correct the predicted navigation solutions.
The transition matrix is defined as Equation (12)[12].

$$\Phi_{k-1} = \begin{bmatrix} I_3 & \tau_s I_3 & 0_{3\times1} & 0_{3\times1} \\ 0_{3\times3} & I_3 & 0_{3\times1} & 0_{3\times1} \\ 0_{1\times3} & 0_{1\times3} & 1 & \tau_s \\ 0_{1\times3} & 0_{1\times3} & 0 & 1 \end{bmatrix} \tag{12}$$

Where τ_s is the the propagation interval. The transition matrix describes the relation of states between previous epoch and current epoch.
Compute the system noise covariance matrix using Equation (13)[13].

$$Q_{k-1} = \begin{bmatrix} \frac{1}{3}S_a\tau_s^3 I_3 & \frac{1}{2}S_a\tau_s^2 I_3 & 0_{3\times1} & 0_{3\times1} \\ \frac{1}{2}S_a\tau_s^2 I_3 & S_a\tau_s I_3 & 0_{3\times1} & 0_{3\times1} \\ 0_{1\times3} & 0_{1\times3} & S_{c\phi}^a\tau_s + \frac{1}{3}S_{cf}^a\tau_s^3 & \frac{1}{2}S_{cf}^a\tau_s^2 \\ 0_{1\times3} & 0_{1\times3} & \frac{1}{2}S_{c\phi}^a\tau_s^2 & S_{cf}^a\tau_s \end{bmatrix} \tag{13}$$

Where S_a is the acceleration power spectral density (PSD), $S_{c\phi}^a$ is the clock phase PSD, and S_{cf}^a is the clock frequency PSD. Use the transition matrix to propagate the state estimates[14],

$$\hat{x}_k^- = \Phi_{k-1}\hat{x}_{k-1}^+ \tag{14}$$

According to the computed transition matrix and noise covariance matrix, the error covariance matrix is propagated as Equation (15)[15].

$$P_k^- = \Phi_{k-1}P_{k-1}^+\Phi_{k-1}^T + Q_{k-1} \tag{15}$$

Similar to the Algorithm 1, it applies the recursion that computes the predicted ranges $\hat{r}_{aj}^-$ by using Equation (1), predicted ranges rate $\hat{\dot{r}}_{aj}^-$ by using Equation (4) and line-of-sight unit vector u_{aj} by using Equation (3). Again, the measurement matrix H_k is formulated by using Equation (6). The noise covariance matrix [16] is as follows

$$R_k = \begin{bmatrix} \sigma_\rho^2 & 0 & \dots & 0 & 0 & 0 & \dots & 0 \\ 0 & \sigma_\rho^2 & \dots & 0 & 0 & 0 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 0 & \sigma_r^2 & 0 & \dots & 0 \\ 0 & 0 & \dots & 0 & 0 & \sigma_r^2 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 0 & 0 & 0 & \dots & \sigma_r^2 \end{bmatrix}_{2N\times2N} \tag{16}$$

And that assume all pseudo range measurements have an error standard deviation σ_ρ of 10m and all pseudo range rate measurements have an error standard deviation of σ_r 0.05m/s. Where the N is the number of satellites.

CONTINUED

The Kalman gain matrix is defined as Equation (17)[17].

$$K_k = P_k^- H_k^T (H_k P_k^- H_k^T + R_k)^{-1} \tag{17}$$

According to the Equation (5), the measurement innovation vector δz^- is formulated. Thus, the state estimates is updated as Equation (18)[18]. And the error covariance matrix is updated as Equation (19)[19].

$$\hat{x}_k^+ = \hat{x}_k^- + K_k \delta z^- \tag{18}$$

$$P_k^+ = (I - K_k H_k) P_k^- \tag{19}$$

Hence, the Kalman Filter algorithm can be summarised as Algorithm 3.

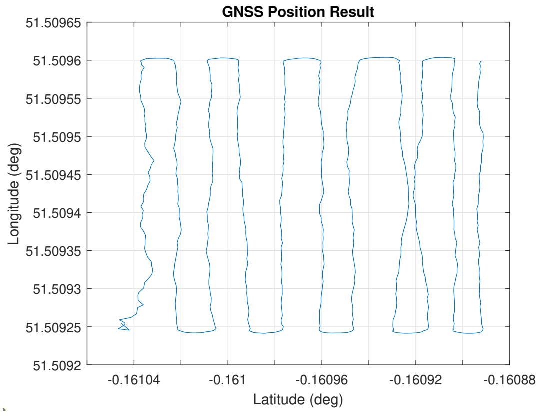
Algorithm 3: Kalman Filter

Input: $\tilde{\rho}$, $\tilde{\dot{\rho}}$

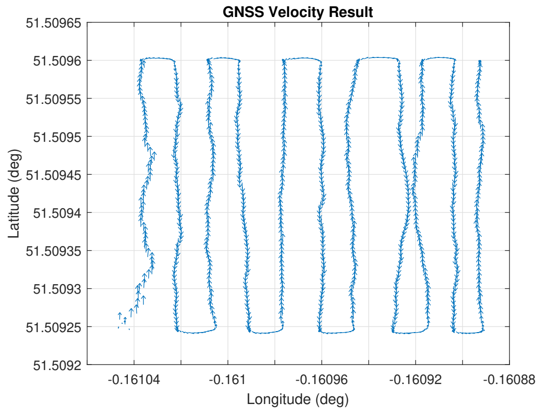
Output: times, t, latitude, L_b, longitude, λ_b, height, h_b, North velocity v_N, and East velocity, v_E

```
Initialize the estimated user state x_0^+ and error covariance matrix P_0^+.
for k < number of epoch do
    Perform the outlier detection algorithm in Algorithm 2 with the input of x_{k-1}^+, ρ̃,
      and ρ̃̇, and obtain the satellite name s, pseudo ranges at epoch k, ρ̃_k, and pseudo
      range rates at epoch k, ρ̃̇_k.
    Compute the transition matrix Φ_{k-1}.
    Compute the noise matrix Q_{k-1}.
    Use the transition matrix to propagate the state estimates.
    Use the Φ_{k-1} and Q_{k-1} to propagate the error covariance matrix.
    Assign number of satellite as N.
    for j < N do
        Compute the Cartesian ECEF positions and velocities of satellites, r̂_ej, v̂_ej.
        Set the Sagnac effect compensation matrix to identity matrix, C_e^I = I_3.
        Compute the ranges from the approximate user position to each satellite, r̂_aj^-.
        Update C_e^I(r̂_aj^-).
        Update the ranges from the approximate user position to each satellite, r̂_aj^-.
        Compute the line-of-sight unit vector from the approximate user position to
          each satellite u_aj.
        Compute the predicted range rates from the approximate user velocity to each
          satellite r̂̇_aj^-.
    Formulate measurement matrix H_k.
    Compute the measurement noise covariance matrix R_k.
    Compute the Kalman gain matrix K_k.
    Formulate measurement innovation vector δz^-.
    Update the state estimates using x_k^+.
    Update the error covariance matrix P_k^+.
    Use function pv_ECEF_to_NED convert Cartesian ECEF user position and
      velocity solution to L_b, λ_b, h_b, v_N and v_E.
```

The filtered states at each epoch possesses user position solution and velocity solution, where position solution is converted to latitude, longitude and height, and convert velocity solution from ECEF resolving axes to north, east and down by using function ***pv_ECEF_to_NED***. The results of GNSS Kalman Filter Solver are plotted in Figure 3.



(a) GNSS Position Diagram



(b) GNSS Velocity Diagram

Figure 3: GNSS Result

CONTINUED

3 Dead Reckoning

This chapter mainly introduces the part of dead reckoning in this coursework, and the algorithm flow chart of the whole part is shown in the Figure 4.

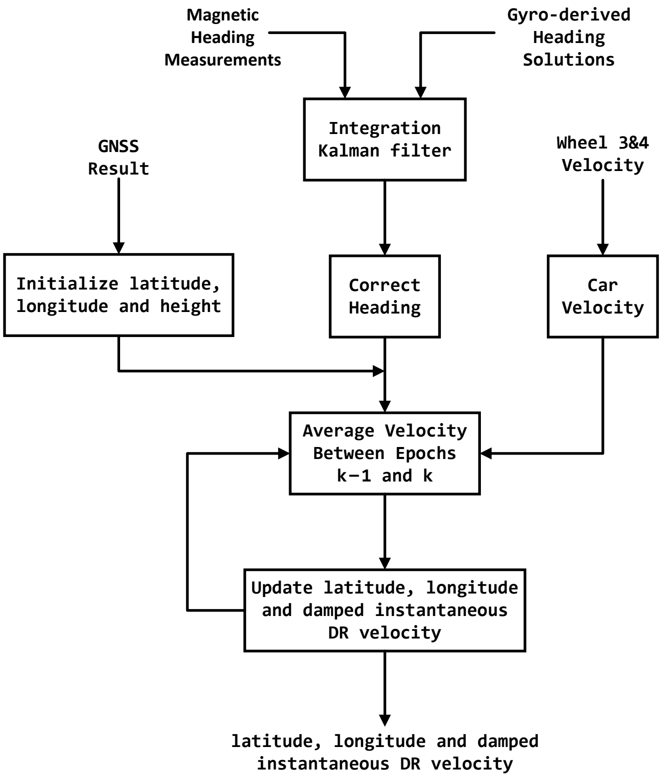


Figure 4: Flow Chart of Dead Reckoning

According to the Figure 4, the GNSS Result will be calculated in Section 2. And this chapter will be divided into two sections, one is the calculation process of gyro-Magnetometer heading integration, and the other is the calculation process of dead reckoning. And after dead reckoning, latitude, longitude and damped instantaneous DR velocity calculated by dead reckoning will return to implement the loosely coupled DR/GNSS integration system.

3.1 Gyro-Magnetometer Heading Integration

The purpose of this section is to make DR more accurate. Consider the known gyroscope angular rate measurements as ω_k^G and the known magnetic heading measurements from the magnetic compass as ψ_k^M, gyro-derived heading solution ψ_k^G can be defined as Equation (20), where $\psi_1^G = \psi_1^M$ and τ_s is the time interval, which value is 0.5 seconds in this coursework.

$$\psi_k^G = \psi_{k-1}^G + \omega_{k-1}^G \times \tau_s \tag{20}$$

According to the gyro-derived heading error $\delta\psi^G$ and gyro bias b^G to determine gyro magnetometer Kalman filter states as $x = \left[\delta\psi^G, b^G\right]^T$. The heading error can be calculated by Equation (22), where transition matrix Φ_{k-1} is defend in Equation (21) [20].

$$\Phi_{k-1} = \begin{bmatrix} 1 & \tau_s \\ 0 & 1 \end{bmatrix} \tag{21}$$

$$\hat{x}_k^- = \begin{bmatrix} \delta\psi_k^G \\ b_k^G \end{bmatrix} = \Phi_{k-1} \begin{bmatrix} \delta\psi_{k-1}^G \\ b_{k-1}^G \end{bmatrix} \tag{22}$$

According to the given gyro random noise with power spectral density (PSD) S_{rg} and the given gyro bias variation with PSD S_{bgd}, the system noise covariance matrix Q can be defined in Equation (23) [21].

$$Q_{k-1} = \begin{bmatrix} S_{rg}\tau_s + \frac{1}{3} S_{bgd}{\tau_s}^3 & \frac{1}{2} S_{bgd}{\tau_s}^2 \\ \frac{1}{2} S_{bgd}{\tau_s}^2 & S_{bgd}\tau_s \end{bmatrix} \tag{23}$$

In the [20], the measurement matrix $H_k = [-1, 0]$. And the measurement noise covariance $R_k = \sigma_M^2$ [20], where σ_M is given as a noise-like error with a standard deviation of $4°$. Then, the Measurement innovation δz_k^- can be calculated by the Equation (24) [20].

$$\delta z_k^- = \left(\psi_k^M - \psi_k^G\right) - H_k \hat{x}_k^- \tag{24}$$

Define the gain K of Kalman filter as Equation (25) [22], where $P = \Phi P_{pos} \Phi^T + Q$ [23] is the prior matrix and P_{pos} is defined as posterior matrix by the scale factor error standard deviation 1% and the cross-coupling error standard deviation 0.1%. And final, correct gyro-heading as ψ_k^C by using Eqaution (26).

$$K = \frac{PH^T}{HPH^T + R} \tag{25}$$

$$\psi_k^C = \psi_k^G - \left(\hat{x}_k^- + K\delta z_k^-\right) \tag{26}$$

Thus, the heading calculation algorithm can be summarized as Algorithm 4.

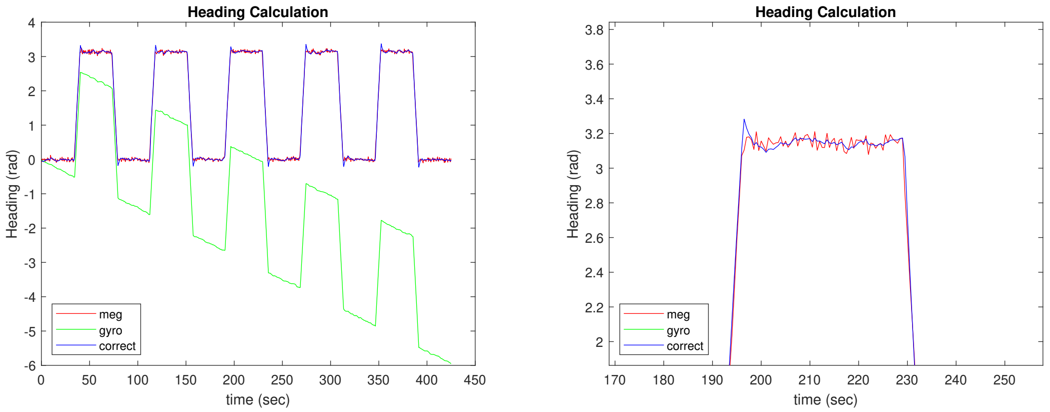
Algorithm 4: Gyro-Magnetometer Heading Integration

Input: ω_k^G, ψ_k^M
Output: ψ_k^M, ψ_k^G, ψ_k^C

Compute gyro-derived heading measurements ψ_k^G.
Determine the posterior matrix P_{pos} and states x.
Determine l as length of ψ_k^G.
for $k < l$ **do**
    Determine PSD of noise S_{rg}, gyro bias variation S_{gbd}, transition matrix Φ.
    Compute the states after transition matrix and prior matrix P.
    Determine the measurement matrix H, the measurement noise covariance R.
    Compute the gain K of Kalman filter.
    Compute the measurement inovation δz_k^-.
    Compute the correct gyro-heading ψ_k^C.
    Compute $k = k + 1$.
Plot the figure of heading.
return ψ_k^C

And Figure 5 shows the heading of magnetic heading measurements, gyro-derived heading measurements and the heading after correcting. Obviously, the calculation of heading got the desired result after being corrected.

CONTINUED



(a) Overall Heading Diagram (b) Partial Heading Diagram

Figure 5: Heading Calculation

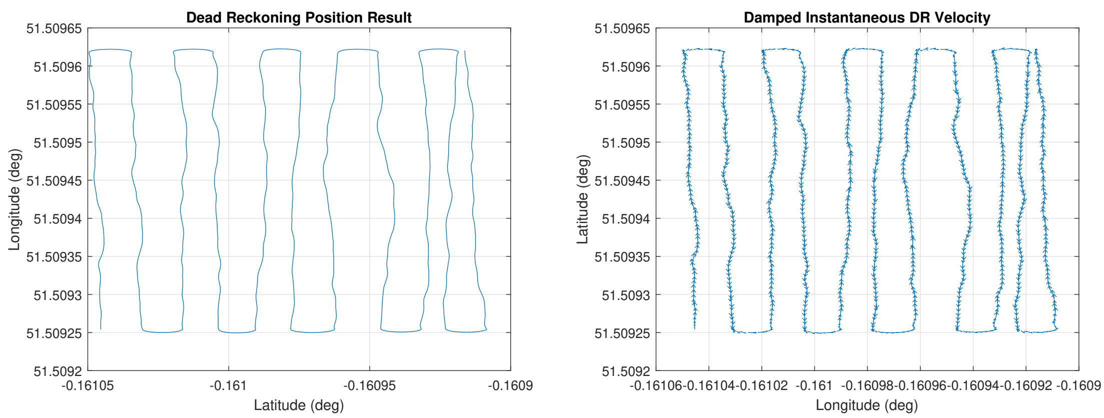
3.2 Dead Reckoning Calculation

Using the result from GNSS to initialize the latitude L_0, longitude λ_0 and height h. Since only the rear wheels with velocity v_k^3 and v_k^4 are the driving wheels, the velocity of car can be defined as $\bar{v}_k = 0.5v_k^3 + 0.5v_k^4$. And then, the average velocity between epochs $k-1$ and k is given in Equation (27) [24].

$$\begin{bmatrix} \bar{v}_{N,k} \\ \bar{v}_{E,k} \end{bmatrix} = \frac{1}{2}\bar{v}_k \begin{bmatrix} \cos\left(\psi_k^C\right) + \cos\left(\psi_{k-1}^C\right) \\ \sin\left(\psi_k^C\right) + \sin\left(\psi_{k-1}^C\right) \end{bmatrix} \tag{27}$$

Then, the meridian radius of curvature R_N and the transverse radius of curvature R_E can be defined by known L_{k-1} and given function 'Radii_of_curvature' in epoch k. Next, compute the L_k and λ_k in epoch k by Equation (28) [25], where t_k means the time in epoch k.

$$L_k = L_{k-1} + \frac{\bar{v}_{N,k}\left(t_k - t_{k-1}\right)}{R_N + h}, \lambda_k = \lambda_{k-1} + \frac{\bar{v}_{E,k}\left(t_k - t_{k-1}\right)}{\left(R_E + h\right)\cos\left(L_k\right)} \tag{28}$$



(a) Dead Reckoning Position Diagram (b) Dead Reckoning Velocity Diagram

Figure 6: Dead Reckoning Result

Consider the damped instantaneous DR velocity in north direction $v_{N,k}$ and the damped instantaneous DR velocity in east direction $v_{E,k}$, initialize them by calculating $v_{N,0} = v_0 \cos\psi_0$ and $v_{E,0} = v_0 \sin\psi_0$. Compute the damped instantaneous DR velocity at each epoch as Equation (29) [26].

$$v_{N,k} = 1.7\bar{v}_{N,k} - 0.7v_{N,k-1}, v_{E,k} = 1.7\bar{v}_{E,k} - 0.7v_{E,k-1} \tag{29}$$

In general, the result of dead reckoning is shown in Figure 6. And the dead reckoning algorithm can be summarized as Algorithm 5.

Algorithm 5: Dead Reckoning Calculation

Input: $\bar{v}_k$, L_0, λ_0, h, ψ_k^C
Output: L_k, λ_k, $v_{N,k}$, $v_{E,k}$

```
Determine l as length of ψ_k^C.
for k < l do
    Compute the average velocity v̄_{N,k} and v̄_{E,k} between epochs k − 1 and k.
    Compute the meridian radius of curvature R_N and the transverse radius of
      curvature R_E.
    Compute the latitude L_k and the longitude λ_k.
    Compute k = k + 1.
Determine the initial damped instantaneous DR velocity v_{N,0} and v_{E,0}.
Determine l as length of ψ_k^C.
for k < l do
    Compute the damped instantaneous DR velocity v_{N,k} and v_{E,k}.
    Compute k = k + 1.
return L_k, λ_k, v_{N,k}, v_{E,k}
```


CONTINUED

4 Loosely Coupled DR/GNSS Integration System

4.1 Open-Loop Loosely Coupled DR/GNSS

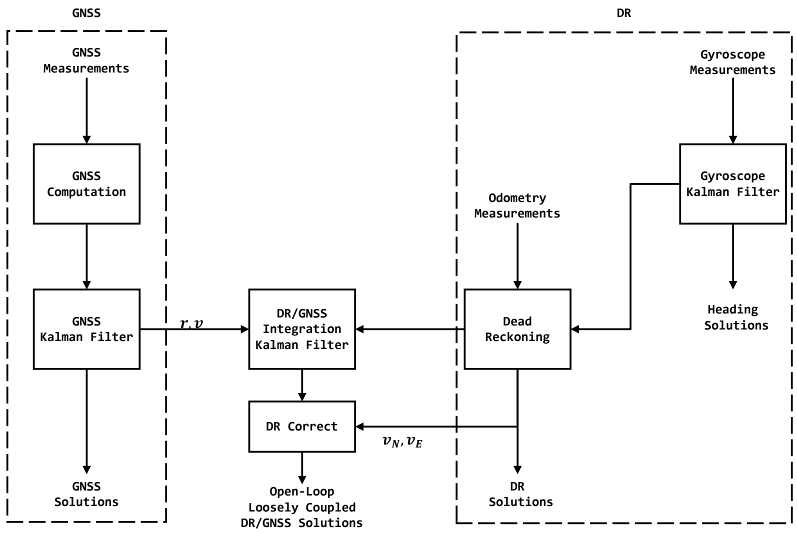


Figure 7: Open-Loop Loosely Coupled DR/GNSS Integration Architecture [27]

The open looped integration Kalman filter has state $x = \{\delta v_N, \delta v_E, \delta L, \delta\lambda\}$, it propagates the state and covariance using dead reckoning kinematics model and correct the result by GNSS data. All the calculation is within NED frame. The equation of transition matrix, system covariance matrix, measurement matrix, measurement covariance matrix refer to [28], [29], [30], [31], respectively. The calculation steps follow the [32].

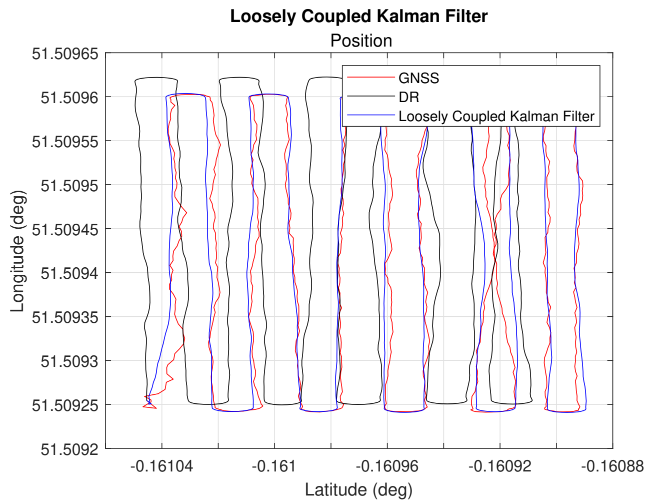


Figure 8: Open-Loop Loosely Coupled DR/GNSS Result

In the figure 8, we can observe that the dead reckoning result will drift as time goes on, but combining the correction of GNSS, the filtered result is quite good, because GNSS can provide accurate measurement while the Dead reckoning provide continuous dynamic model that we can use to predict the state, the Kalman filter fuses the results and takes advantages of both. However, the Dead reckoning open loop result has drifted too much that may affect the final result, in this case, we optimized it by using close loop correction for the Dead Reckoning.

4.2 Closed-Loop Loosely Coupled DR/GNSS

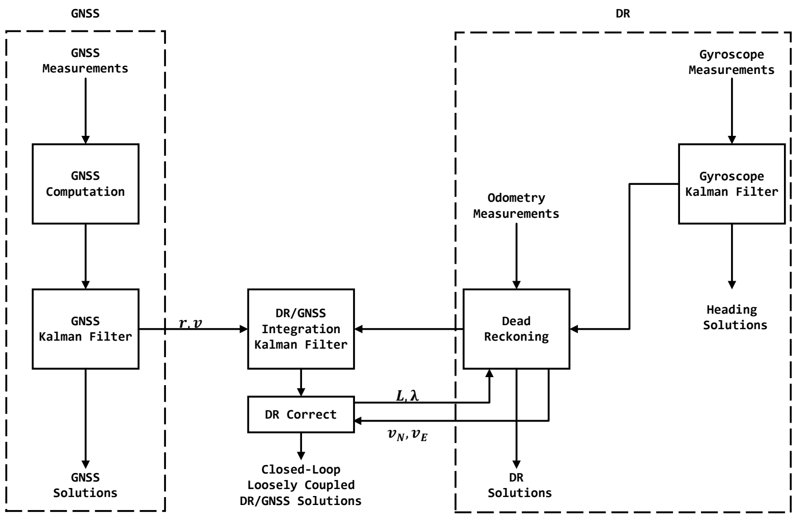
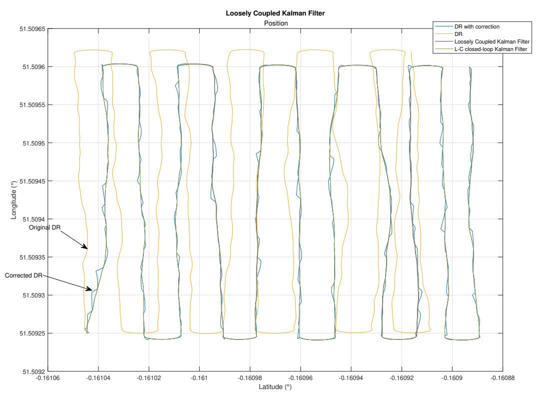
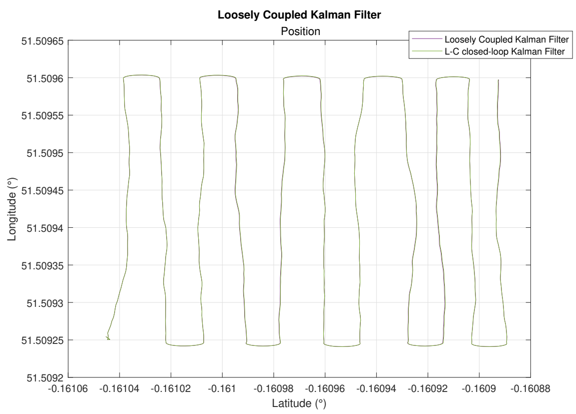


Figure 9: Closed-Loop Loosely Coupled DR/GNSS Integration Architecture [27]

In the closed-loop correction process, we simply provide the corrected result, v_N, v_E, L, λ, at time t, back to the Dead Reckoning model, assuming that our state has zero error (i.e., x = $0, 0, 0, 0$). This allows the Dead Reckoning calculation at the next time step, t + 1, to start with the corrected position and velocity, potentially resulting in a better outcome. However, in this coursework, the corrected result was not continuously fed back. Instead, it was fed back every 5 time steps to make it easier to observe the correction of the Dead Reckoning model and still maintain a good filter result.



(a) Dead Reckoning result Diagram



(b) Dead Reckoning Velocity Diagram

Figure 10: Two Methods Result Comparison

CONTINUED

Observing the original DR and closed-loop DR result in Figure 10, it is clear that the closed-loop DR can fetch a better result, which implies the closd-loop Kalman filter may have better performance than an open-loop one.

References

[1] P. D. Groves, *Workshop 1: Mobile GNSS Positioning using Least-Squares Estimation*. UCL Moodle, 2023, p. 1, eq.(1). [Online]. Available: https://moodle.ucl.ac.uk/mod/resource/view.php?id=4085869

[2] ——, *Workshop 1: Mobile GNSS Positioning using Least-Squares Estimation*. UCL Moodle, 2023, p. 2, eq.(2). [Online]. Available: https://moodle.ucl.ac.uk/mod/resource/view.php?id=4085869

[3] ——, *Workshop 1: Mobile GNSS Positioning using Least-Squares Estimation*. UCL Moodle, 2023, p. 2, eq.(3). [Online]. Available: https://moodle.ucl.ac.uk/mod/resource/view.php?id=4085869

[4] ——, *Workshop 1: Mobile GNSS Positioning using Least-Squares Estimation*. UCL Moodle, 2023, p. 3, eq.(9). [Online]. Available: https://moodle.ucl.ac.uk/mod/resource/view.php?id=4085869

[5] ——, *Workshop 1: Mobile GNSS Positioning using Least-Squares Estimation*. UCL Moodle, 2023, p. 3, eq.(10). [Online]. Available: https://moodle.ucl.ac.uk/mod/resource/view.php?id=4085869

[6] ——, *Workshop 2: Aircraft Navigation using GNSS and Kalman Filter*. UCL Moodle, 2023, p. 7, eq.(26). [Online]. Available: https://moodle.ucl.ac.uk/mod/resource/view.php?id=4085915

[7] ——, *Workshop 2: Aircraft Navigation using GNSS and Kalman Filter*. UCL Moodle, 2023, p. 5, eq.(23). [Online]. Available: https://moodle.ucl.ac.uk/mod/resource/view.php?id=4085947

[8] ——, *Workshop 1: Mobile GNSS Positioning using Least-Squares Estimation*. UCL Moodle, 2023, p. 2, eq.(5). [Online]. Available: https://moodle.ucl.ac.uk/mod/resource/view.php?id=4085869

[9] ——, *Workshop 1: Mobile GNSS Positioning using Least-Squares Estimation*. UCL Moodle, 2023, p. 3, eq.(6). [Online]. Available: https://moodle.ucl.ac.uk/mod/resource/view.php?id=4085915

[10] ——, *Workshop 1: Mobile GNSS Positioning using Least-Squares Estimation*. UCL Moodle, 2023, p. 3, eq.(7). [Online]. Available: https://moodle.ucl.ac.uk/mod/resource/view.php?id=4085869

[11] ——, *Workshop 1: Mobile GNSS Positioning using Least-Squares Estimation*. UCL Moodle, 2023, p. 3, eq.(8). [Online]. Available: https://moodle.ucl.ac.uk/mod/resource/view.php?id=4085869

[12] ——, *Workshop 2: Aircraft Navigation using GNSS and Kalman Filter*. UCL Moodle, 2023, p. 4, eq.(14). [Online]. Available: https://moodle.ucl.ac.uk/mod/resource/view.php?id=4085915

CONTINUED

[13] ——, *Workshop 2: Aircraft Navigation using GNSS and Kalman Filter*. UCL Moodle, 2023, p. 4, eq.(15). [Online]. Available: https://moodle.ucl.ac.uk/mod/resource/view.php?id=4085915

[14] ——, *Workshop 2: Aircraft Navigation using GNSS and Kalman Filter*. UCL Moodle, 2023, p. 4, eq.(16). [Online]. Available: https://moodle.ucl.ac.uk/mod/resource/view.php?id=4085915

[15] ——, *Workshop 2: Aircraft Navigation using GNSS and Kalman Filter*. UCL Moodle, 2023, p. 4, eq.(17). [Online]. Available: https://moodle.ucl.ac.uk/mod/resource/view.php?id=4085915

[16] ——, *Workshop 2: Aircraft Navigation using GNSS and Kalman Filter*. UCL Moodle, 2023, p. 6, eq.(24). [Online]. Available: https://moodle.ucl.ac.uk/mod/resource/view.php?id=4085915

[17] ——, *Workshop 2: Aircraft Navigation using GNSS and Kalman Filter*. UCL Moodle, 2023, p. 6, eq.(25). [Online]. Available: https://moodle.ucl.ac.uk/mod/resource/view.php?id=4085915

[18] ——, *Workshop 2: Aircraft Navigation using GNSS and Kalman Filter*. UCL Moodle, 2023, p. 7, eq.(27). [Online]. Available: https://moodle.ucl.ac.uk/mod/resource/view.php?id=4085915

[19] ——, *Workshop 2: Aircraft Navigation using GNSS and Kalman Filter*. UCL Moodle, 2023, p. 8, eq.(28). [Online]. Available: https://moodle.ucl.ac.uk/mod/resource/view.php?id=4085915

[20] ——, *3B: Multisensor Integrated Navigation*. UCL Moodle, 2023, slides 41. [Online]. Available: https://moodle.ucl.ac.uk/mod/resource/view.php?id=4085936

[21] ——, *3B: Multisensor Integrated Navigation*. UCL Moodle, 2023, slides 40. [Online]. Available: https://moodle.ucl.ac.uk/mod/resource/view.php?id=4085936

[22] ——, *3B: Multisensor Integrated Navigation*. UCL Moodle, 2023, slides 28. [Online]. Available: https://moodle.ucl.ac.uk/mod/resource/view.php?id=4085936

[23] ——, *3B: Multisensor Integrated Navigation*. UCL Moodle, 2023, slides 25. [Online]. Available: https://moodle.ucl.ac.uk/mod/resource/view.php?id=4085936

[24] ——, *Workshop 3: Multisensor Navigation*. UCL Moodle, 2023, p. 1, eq.(1). [Online]. Available: https://moodle.ucl.ac.uk/mod/resource/view.php?id=4085947

[25] ——, *Workshop 3: Multisensor Navigation*. UCL Moodle, 2023, p. 1, eq.(2). [Online]. Available: https://moodle.ucl.ac.uk/mod/resource/view.php?id=4085947

[26] ——, *Workshop 3: Multisensor Navigation*. UCL Moodle, 2023, p. 1, eq.(3). [Online]. Available: https://moodle.ucl.ac.uk/mod/resource/view.php?id=4085947

[27] ——, *Principles of GNSS, Inertial, and Multisensor Integrated Navigation Systems (2nd Edition)*. Artech House, 2013, ch. 14.1.2 Loosely Coupled Integration, p. 566, isbn. 978-1-60807-005-3. [Online]. Available: https://app.knovel.com/hotlink/khtml/id:kt011M4CU3/principles-gnss-inertial/loosely-coupled-integration

[28] ——, *Workshop 3: Multisensor Navigation*. UCL Moodle, 2023, p. 2, eq.(6). [Online]. Available: https://moodle.ucl.ac.uk/mod/resource/view.php?id=4085947

[29] ——, *Workshop 3: Multisensor Navigation*. UCL Moodle, 2023, p. 3, eq.(7). [Online]. Available: https://moodle.ucl.ac.uk/mod/resource/view.php?id=4085947

[30] ——, *Workshop 3: Multisensor Navigation*. UCL Moodle, 2023, p. 3, eq.(10). [Online]. Available: https://moodle.ucl.ac.uk/mod/resource/view.php?id=4085947

[31] ——, *Workshop 3: Multisensor Navigation*. UCL Moodle, 2023, p. 3, eq.(11). [Online]. Available: https://moodle.ucl.ac.uk/mod/resource/view.php?id=4085947

[32] ——, *Workshop 3: Multisensor Navigation*. UCL Moodle, 2023, task. 2. [Online]. Available: https://moodle.ucl.ac.uk/mod/resource/view.php?id=4085947

CONTINUED

Appendix A: Matlab Code for Open-Loop LC Solution

File name: loosely_coupled_solution.m

```
clc
clear
close
% add function and data that i  s given by paul.
addpath("Functions_Given", "Data_Given");
Define_Constants;
% Dataload
Pseudo_ranges_data = readmatrix("Pseudo_ranges.csv");
Pseudo_ranges_rates_data = readmatrix("Pseudo_range_rates.csv");
DR_Data = readmatrix("Dead_reckoning.csv");

% calculate the GNSS Solution,
% return t L_b, lambda_b, h_b, v_N, v_E, Psi
GNSS_Solution = GNSS_Solver(Pseudo_ranges_data, ...
    Pseudo_ranges_rates_data);

% calculate the heading Solution, use gyro and meg ,return psi
psi_C = Gyro_Mag_KF_Solver(DR_Data);

% calculate the Dead reconing Solution,
% return t L_b, lambda_b, h_b, v_N, v_E, Psi
DR_Solution = DR_Solver(DR_Data, GNSS_Solution, psi_C);

% couple the two solutions loosely
LCKF_Solution = LC_KF_Inte_Solver(DR_Solution, GNSS_Solution);

% couple the two solutions loosely with close loop
LCKF_Close_Solution = LC_KF_Inte_Close_Solver(DR_Data, ...
    GNSS_Solution, psi_C);

% plot figure
color = ['r', 'k', 'b', 'g'];
Result_Plot('GNSS', GNSS_Solution.L_b*rad_to_deg, ...
                GNSS_Solution.lambda_b*rad_to_deg, color(1));
Result_Plot('DR', DR_Solution.L_b*rad_to_deg, ...
                DR_Solution.lambda_b*rad_to_deg, color(2));
Result_Plot('Loosely Coupled Kalman Filter', ...
                LCKF_Solution.L_b*rad_to_deg, ...
                LCKF_Solution.lambda_b*rad_to_deg, color(3));
Result_Plot('Loosely Coupled closed loop Kalman Filter', ...
                LCKF_Close_Solution.L_b*rad_to_deg, ...
                LCKF_Close_Solution.lambda_b*rad_to_deg, color(4));
legend("DR with correction",'GNSS', 'DR', 'Loosely Coupled Kalman ...
    Filter', ...
        "L-C closed-loop Kalman Filter")

% store file
opres = [LCKF_Solution.t,LCKF_Solution.L_b'*rad_to_deg, ...
         LCKF_Solution.lambda_b'*rad_to_deg,LCKF_Solution.v_N', ...
         LCKF_Solution.v_E',psi_C'*rad_to_deg];
```

```
writematrix(opres,"Openloop_Output_Profile.csv");
cpres = [LCKF_Close_Solution.t,LCKF_Close_Solution.L_b'*rad_to_deg, ...
         LCKF_Close_Solution.lambda_b'*rad_to_deg,LCKF_Close_Solution.v_N', ... ...
         LCKF_Close_Solution.v_E',psi_C'*rad_to_deg];
writematrix(cpres,"Closedloop_Output_Profile.csv");
```


CONTINUED

File name: GNSS_Solver.m

```
function GNSS_Solution = GNSS_Solver(Pseudo_ranges_data, ...
    Pseudo_ranges_rates_data)
     % Pseudo_ranges_data:
     % col 2-9, row 2-end : satellete data
     % col 1, row 2-end : time

     % Pseudo_ranges_rate_data:
     % col 2-9, row 2-end : satellete data
     % col 1, row 2-end : time
     Define_Constants;

     num_epoch = size(Pseudo_ranges_data,1);
     num_epoch = num_epoch -1;
     % Task2A (a) Initialise the Kalman filter
     [x_pos,P_pos] = Initialise_GNSS_KF(Pseudo_ranges_data, ...
                                        Pseudo_ranges_rates_data);

     for epoch =  2:num_epoch+1
         % Outlier detect and remove
         [satellite_name_list, pseudo_ranges, pseudo_range_rates ] = ...
                    Detect_Outlier_And_Remove(x_pos,epoch, ...
                               Pseudo_ranges_data, ...
                                   Pseudo_ranges_rates_data);
         num_satellite = length(satellite_name_list);
         % Task2A (b) Compute the transition matrix
         tau_s = 0.5;
         Phi = [eye(3), tau_s*eye(3), zeros(3,2);
                zeros(3), eye(3), zeros(3,2);
                zeros(1,6), 1, tau_s;
                zeros(1,7), 1];

         % Task2A (c) Compute the system noise covariance matrix
         S_a = 2;
         S_c_phi = 0.01;
         S_c_f = 0.04;
         Q = [1/3*S_a*tau_s^3*eye(3), 0.5*S_a*tau_s^2*eye(3), zeros(3,2);
              0.5*S_a*tau_s^2*eye(3), S_a*tau_s*eye(3), zeros(3,2);
              zeros(1,6), S_c_phi*tau_s + 1/3*S_c_f*tau_s^3, ...
                  0.5*S_c_f*tau_s^2;
              zeros(1,6), 0.5*S_c_f*tau_s^2, S_c_f*tau_s];

         % Task2A (d) Use the transition matrix to propagate the ...
             state estimates
         x_pri = Phi * x_pos;

         % Task2A (e) use this to propagate the error covariance matrix
         x_conv_pri = Phi*P_pos*Phi' + Q;

         % Task2A (f)(g) Predict the ranges from the approximate user ...
             position to each satellite
         % calculate satellite position
         r_ej = zeros(3,num_satellite);
         v_ej = zeros(3,num_satellite);
```

```matlab
        for i =  1:num_satellite
            [position_satellite, velocity_satellite] = ...
                                 Satellite_position_and_velocity ...
                                     (Pseudo_ranges_data(epoch,1), ...
                                      satellite_name_list(i));
            r_ej(:,i) = position_satellite';
            v_ej(:,i) = velocity_satellite';
        end

        r_aj_pri = zeros(1,num_satellite);
        u_aj = zeros(3,num_satellite);
        r_aj_dot_pri = zeros(1,num_satellite);
        for i = 1:num_satellite
            % calculate C

            C = eye(3);
            r_aj_pri(i) = sqrt((C* r_ej(:,i) - x_pri(1:3))' * ...
                               (C* r_ej(:,i) - x_pri(1:3)));

            C(1,2) = omega_ie * r_aj_pri(i) /c;
            C(2,1) = -omega_ie * r_aj_pri(i) /c;
            r_aj_pri(i) = sqrt((C* r_ej(:,i) - x_pri(1:3))' * ...
                               (C* r_ej(:,i) - x_pri(1:3)));
            u_aj(:,i) = (C * r_ej(:,i) - x_pri(1:3)) / r_aj_pri(i);
            % Task2A (h) Predict the range rates
            r_aj_dot_pri(i) = u_aj(:,i)' * ...
                             (C * (v_ej(:,i) + Omega_ie*r_ej(:,i)) - ...
                              (x_pri(4:6) + Omega_ie* x_pri(1:3))) ;
        end

        % Task2A (i) Compute the measurement matrix
        H = [-u_aj', zeros(num_satellite,3), ones(num_satellite,1), ...
            zeros(num_satellite,1);
              zeros(num_satellite,3), -u_aj', ...
                  zeros(num_satellite,1),ones(num_satellite,1)];

        % Task2A (j) Compute the measurement noise covariance
        sigma_rho = 10;
        sigma_r = 0.05;
        R_k = [eye(num_satellite)*sigma_rho^2 , zeros(num_satellite);
               zeros(num_satellite), eye(num_satellite)*sigma_r^2];

        % Task2A (k) Compute the Kalman gain
        K_k = x_conv_pri * H' / (H*x_conv_pri*H' + R_k);

        % Task2A (I) Formulate the measurement innovation
        Δ_z = [(pseudo_ranges - r_aj_pri - x_pri(7))';
                  (pseudo_range_rates - r_aj_dot_pri - x_pri(8))'];

        % Task2A (m) Update the state estimates
        x_pos = x_pri + K_k * Δ_z;

        % Task2A (n) Update the error covariance matrix
        P_pos = (eye(8) - K_k*H)*x_conv_pri;
```


CONTINUED

```
            % Task2A (o) check solution
            [latitude(epoch-1), longitude(epoch-1) , ...
               height(epoch-1), v_ebn(:,epoch-1)] = ...
                  pv_ECEF_to_NED(x_pos(1:3), x_pos(4:6));

        end
        GNSS_Solution.t = Pseudo_ranges_data(2:end,1);
        GNSS_Solution.L_b = latitude;
        GNSS_Solution.lambda_b = longitude;
        GNSS_Solution.h_b = height;
        GNSS_Solution.v_N = v_ebn(1,:);
        GNSS_Solution.v_E = v_ebn(2,:);

        figure
        plot(longitude*rad_to_deg, latitude*rad_to_deg, '-');
        title('GNSS Position Result');
        xlabel('Latitude (deg)');
        ylabel('Longitude (deg)');
        grid on;

        figure
        quiver(longitude*rad_to_deg,latitude*rad_to_deg, ...
                         GNSS_Solution.v_E,GNSS_Solution.v_N)
        grid on;
        xlabel('Longitude (deg)');
        ylabel('Latitude (deg)');
        title('GNSS Velocity Result')

    end
```

File name: GNSS_LS_Solver.m

```
function [x_pos,latitude,longitude,height,velocity,H_G,Δ_z] = ...
                GNSS_LS_Solver(sat_names,pseudo_ranges_epoch, ...
                               pseudo_range_rates_epoch,x_pri)
    % Constant
    r_eb_e = x_pri(1:3);
    v_eb_e = x_pri(4:6);
    Δ_rho = x_pri(7);
    dot_Δ_rho = x_pri(8);
    time = pseudo_ranges_epoch(1);
    num_sat = length(pseudo_ranges_epoch)-1;
    omega_ie = 7.292115e-5;
    Omega_ie = Skew_symmetric([0,0,omega_ie]);
    c = 299792458;
    max_it = 10;
    R_a = zeros(1,8);
    for i = 1:max_it
        % b) compute Cartesian ECEF satellite position and velocity
        Sat_r_es_e = zeros(3,num_sat);
        Sat_v_es_e = zeros(3,num_sat);
        for j = 2:num_sat+1
            [sat_r_es_e,sat_v_es_e] = ...
                Satellite_position_and_velocity(time,sat_names(j));
            Sat_r_es_e(:,j-1) = sat_r_es_e';
            Sat_v_es_e(:,j-1) = sat_v_es_e';
        end

        % c) & d) compute predicted ranges and range rates
        C_Ie = eye(3);
        U_a = zeros(3,num_sat);
        dot_R_a = zeros(1,num_sat);
        for j = 1:length(Sat_r_es_e)
            if i == 1
                R_a(j) = sqrt((C_Ie*Sat_r_es_e(:,j)-r_eb_e)'* ...
                             (C_Ie*Sat_r_es_e(:,j)-r_eb_e));
                C_Ie = [1, omega_ie*R_a(j)/c, 0;
                    -omega_ie*R_a(j)/c, 1, 0;
                    0, 0, 1];
            else
                C_Ie = [1, omega_ie*R_a(j)/c, 0;
                    -omega_ie*R_a(j)/c, 1, 0;
                    0, 0, 1];
            end
            R_a(j) = sqrt((C_Ie*Sat_r_es_e(:,j)-r_eb_e)'* ...
                         (C_Ie*Sat_r_es_e(:,j)-r_eb_e));
            U_a(:,j) = (C_Ie*Sat_r_es_e(:,j)-r_eb_e)/R_a(j);
            dot_R_a(j) = U_a(:,j)'*(C_Ie*(Sat_v_es_e(:,j)+ ...
                         Omega_ie*Sat_r_es_e(:,j))- ...
                         (v_eb_e+Omega_ie*r_eb_e));
        end


        % e) formulate measurement innovation vector and measurement ...
            matrix
        x_hat = [r_eb_e; Δ_rho];
```

```
        dot_x_hat = [v_eb_e; dot_Δ_rho];
        Δ_z = zeros(num_sat,1);
        dot_Δ_z = zeros(num_sat,1);
        H_G = zeros(num_sat,4);
        for j = 2:num_sat+1
            Δ_z(j-1) =  pseudo_ranges_epoch(j)-R_a(j-1)-Δ_rho;
            dot_Δ_z(j-1) =  ...
                pseudo_range_rates_epoch(j)-dot_R_a(j-1)-dot_Δ_rho;
            H_G(j-1,:) = [-U_a(:,j-1)',1];
        end

        % f) unweighted least squares
        x_hat = x_hat + (H_G'*H_G)\H_G'*Δ_z;
        dot_x_hat = dot_x_hat + (H_G'*H_G)\H_G'*dot_Δ_z;

        % g) convert Cartesian ECEF solution to latitude, longitude, ...
           height and velocity
        [latitude,longitude,height,velocity] = ...
           pv_ECEF_to_NED(x_hat(1:3),dot_x_hat(1:3));

        % update user state
        x_pos = [x_hat(1:3);dot_x_hat(1:3);x_hat(4);dot_x_hat(4)];
        if norm(r_eb_e-x_hat(1:3)) < 0.1
            break;
        end
        r_eb_e = x_hat(1:3);
        v_eb_e = dot_x_hat(1:3);
        Δ_rho = x_hat(4);
        dot_Δ_rho = dot_x_hat(4);
    end
end
```

File name: Initialise_GNSS_KF.m

```
function [x_est,P_matrix] = ...
    Initialise_GNSS_KF(pseudo_ranges_data,pseudo_range_rates_data )

    % compute least-square solution
    % and remove outlier
    while true
        [x_est,¬,¬,¬,¬,H_G,∆_z] = ...
            GNSS_LS_Solver(pseudo_ranges_data(1,:), ...
                           pseudo_ranges_data(2,:), ...
                           pseudo_range_rates_data(2,:), ...
                           zeros(8,1));
        outlier_labels = ...
           outlier_detection(pseudo_ranges_data(2,:),H_G,∆_z);
        if(¬isempty(outlier_labels))
            [¬,index] = max(abs(outlier_labels(:,2)));
            removel_index = outlier_labels(index,1);
            pseudo_ranges_data(:,removel_index+1) = [];
            pseudo_range_rates_data(:,removel_index+1) = [];
        else
            break
        end
    end

% Initialise error covariance matrix
P_matrix =  zeros(8);
P_matrix(1,1) = 100;
P_matrix(2,2) = 100;
P_matrix(3,3) = 100;
P_matrix(4,4) = 0.01;
P_matrix(5,5) = 0.01;
P_matrix(6,6) = 0.01;
P_matrix(7,7) = 100000^2;
P_matrix(8,8) = 200^2;

end
```


CONTINUED

File name: Detect_Outlier_And_Remove.m

```
function [satellite_name_list, pseudo_ranges, pseudo_range_rates ] = ...
    Detect_Outlier_And_Remove(x_pri,epoch, pseudo_ranges_data, ...
    pseudo_range_rates_data)
     while true
         % compute H matrix and z
         [¬,¬,¬,¬,¬,H_G,∆_z] = ...
              GNSS_LS_Solver(pseudo_ranges_data(1,:), ...
                             pseudo_ranges_data(epoch,:), ...
                             pseudo_range_rates_data(epoch,:), ...
                             x_pri);

         % compute outlier index
         outlier_labels = ...
             outlier_detection(pseudo_ranges_data(epoch,:),H_G,∆_z);

         % check outlier: none, then stop
         if(isempty(outlier_labels))
             break
         end

         % check outlier: yes, remove the maximum
         [¬,index] = max(abs(outlier_labels(:,2)));
         removel_index = outlier_labels(index,1);
         pseudo_ranges_data(:,removel_index+1) = [];
         pseudo_range_rates_data(:,removel_index+1) = [];
     end

     % ouput the measurement of pseudo_ranges, pseudo_range_rates and ...
         satellite name retained
     satellite_name_list = pseudo_ranges_data(1,2:end);
     pseudo_ranges = pseudo_ranges_data(epoch,2:end);
     pseudo_range_rates = pseudo_range_rates_data(epoch,2:end);
end
```

File name: outlier_detection.m

```
function outlier_labels = outlier_detection(detecting_epoch,H_G,Δ_z)
    %%%%%%%%%%%%%%%%%%%%%
    % Input:  detecting_epoch, 1xN, current epoch of pseudo ranges
    %         H_G, Nx4, measurement matrix
    %         Δ_z, Nx1, measurement innovation vector
    % Output: outlier_labels, nx2, index of outlier and ...
        corresponding residual
    %%%%%%%%%%%%%%%%%%%%%

    outlier_labels = [];
    sigma = 5; % measurement error standard deviation
    T = 6; % outlier detection threshold
    %% a) Compute the residuals vector
    residuals = ...
        (H_G/(H_G'*H_G)*H_G'-eye(size(detecting_epoch,2)-1))*Δ_z;

    %% b) Compute the residuals convariance matrix
    C_v = (eye(size(detecting_epoch,2)-1)-H_G/(H_G'*H_G)*H_G')*sigma^2;

    %% c) Compute the normalised residuals and compare each with ...
        threshold
    for j = 1: length(residuals)
        if(abs(residuals(j)) > sqrt(C_v(j,j))*T)

            outlier_labels = [outlier_labels;[j,residuals(j)]];
        end
    end
end
```


CONTINUED

File name: Gyro_Mag_KF_Solver.m

```
function psi_C = Gyro_Mag_KF_Solver(DR_Data)
    Define_Constants;
    omega_G = DR_Data(:,6);
    psi_M = DR_Data(:,7)*deg_to_rad;
    psi_G = zeros(length(psi_M), 1);
    psi_G(1) = psi_M(1);
    Tau_s = 0.5;
    for i = 2:length(psi_M)
        psi_G(i) = psi_G(i-1) + omega_G(i-1) * Tau_s;
    end

    % assume state x = [Δ_psi, b]  b = 1deg/s
    x_pos = [0; deg_to_rad];
    P_pos = [0.01^2,  0;
              0, 0.001^2];
    % start kalman
    for k = 1:length(psi_G)
        S_rg = 1e-4;
        S_bgd = deg_to_rad;
        Phi = [1, Tau_s;
               0, 1];
        Q = [S_rg*Tau_s + 1/3*S_bgd*Tau_s^3, 0.5*S_bgd*Tau_s^2;
            0.5*S_bgd*Tau_s^2, S_bgd *Tau_s];

        % transite the model
        x_pri = Phi*x_pos;
        P_pri = Phi*P_pos*Phi' + Q;

        % define H and R
        H = [-1 0];
        R = (4 * deg_to_rad)^2;
        K = P_pri*H'/(H*P_pri*H' + R);

        % calculate measurement inovation
        Δ_z = (psi_M(k) - psi_G(k)) - H * x_pri;

        % calculate posterir
        x_pos = x_pri + K*Δ_z;
        P_pos = (eye(2) - K*H)*P_pri;

        % get corrected value
        psi_C(k) = psi_G(k) - x_pos(1);
    end
    figure
    plot(DR_Data(:,1),psi_M, Color='r');
    hold on
    plot(DR_Data(:,1),psi_G, Color='k');
    plot(DR_Data(:,1),psi_C, Color='b');
    xlabel('time (sec)')
    ylabel('Heading (rad)')
    title('Heading Calculation')
    legend("meg","gyro","correct", Location="southwest")
end
```

File name: DR_Solver.m

```
function DR_Solution = DR_Solver(DR_Data, GNSS_Solution, psi_C)
    % DR_Data:
    % columns 1 time
    % columns 2-5: wheel-speed measurements
    % column 6: the gyroscope angular rate radians per second
    % column 7: contains the heading measurements in degrees from ...
        the magnetic compass
    Define_Constants;

    lambda_0 = GNSS_Solution.lambda_b(1);
    L_0 = GNSS_Solution.L_b(1);
    h = mean(GNSS_Solution.h_b); % check here for every GNSS data
    v_bar = (DR_Data(:,4)+DR_Data(:,5))/2 ;

    t = DR_Data(:,1);
    psi = psi_C; %DR_Data(:,7) * deg_to_rad;

    L_k(1) = L_0;
    lambda_k(1) = lambda_0;
    for k = 2:length(psi)
        v_N_bar(k) = 0.5*(cos(psi(k)) + cos(psi(k-1)))*v_bar(k);
        v_E_bar(k) = 0.5*(sin(psi(k)) + sin(psi(k-1)))*v_bar(k);

        [R_N,R_E] = Radii_of_curvature(L_k(k-1));
        L_k(k) = L_k(k-1) + (v_N_bar(k)*(t(k) - t(k-1))) / (R_N + h);
        lambda_k(k) = lambda_k(k-1) + (v_E_bar(k)*(t(k) - t(k-1)) / ...
            ((R_E + h) * cos(L_k(k))));
    end
    v_N(1) = v_bar(1) * cos(psi(1));
    v_E(1) = v_bar(1) * sin(psi(1));
    for k = 2:length(psi)
       v_N(k) = 1.7 * v_N_bar(k) - 0.7 * v_N(k-1);
       v_E(k) = 1.7 * v_E_bar(k) - 0.7 * v_E(k-1);
    end
    DR_Solution.L_b = L_k;
    DR_Solution.lambda_b = lambda_k;
    DR_Solution.Psi = psi;
    DR_Solution.v_E = v_E;
    DR_Solution.v_N = v_N;
    DR_Solution.t = DR_Data(:,1);
    figure
    plot(lambda_k*rad_to_deg, L_k*rad_to_deg, '-');
    title('Dead Reckoning Position Result');
    xlabel('Latitude (deg)');
    ylabel('Longitude (deg)');
    grid on;
    figure
    quiver(lambda_k*rad_to_deg ,L_k*rad_to_deg,v_E,v_N)
    grid on;
    xlabel('Longitude (deg)');
    ylabel('Latitude (deg)');
    title('Damped Instantaneous DR Velocity')
end
```


CONTINUED

File name: LC_KF_Inte_Solver.m

```
function LCKF_Solution = LC_KF_Inte_Solver(DR_Solution, GNSS_Solution)

    L_k_D = DR_Solution.L_b;
    lambda_k_D = DR_Solution.lambda_b;
    v_N_D = DR_Solution.v_N;
    v_E_D = DR_Solution.v_E;

    t = DR_Solution.t;
    L_k_G = GNSS_Solution.L_b;
    lambda_k_G = GNSS_Solution.lambda_b;
    h_k_G = GNSS_Solution.h_b;
    v_N_G= GNSS_Solution.v_N;
    v_E_G= GNSS_Solution.v_E;

    [R_N, R_E]= Radii_of_curvature(L_k_G(1));

    x_pos = zeros(4,1); % v_N, v_E, L, lambda
    P_pos = [eye(2)*0.1^2, zeros(2);
             zeros(2), [(10/(R_N+h_k_G(1)))^2, 0;
                         0, (10/((R_E+h_k_G(1)) * cos(L_k_G(1))))^2]];
    tau_s = 0.5;
    S_DR = 0.2;
    sigma_Gr = 5;
    sigma_Gv = 0.02;
    %%%%%%% prepare kalman filter parameters
    [R_N, R_E]= Radii_of_curvature(L_k_G(1));
    Phi = eye(4);
    Phi(3,1) = 0.5/(R_N + h_k_G(1));
    Phi(4,2) = 0.5/((R_E + h_k_G(1)) * cos(L_k_D(1)));
    % define system noise covariance %
    Q = [S_DR*tau_s, 0 , 0.5*S_DR*tau_s^2/(R_N+h_k_G(1)), 0;
         0, S_DR*tau_s, 0, ...
            0.5*S_DR*tau_s^2/((R_E+h_k_G(1))*cos(L_k_D(1)));
         0.5*S_DR*tau_s^2/(R_N+h_k_G(1)), 0, ...
            1/3*S_DR*tau_s^3/(R_N+h_k_G(1))^2, 0;
         0, 0.5*S_DR*tau_s^2/((R_E+h_k_G(1))*cos(L_k_D(1))), ...
         0, 1/3*S_DR*tau_s^3/((R_E+h_k_G(1))^2*cos(L_k_D(1))^2)];

    % Propagate the state estimates and the error covariance matrix
    x_pri = Phi*x_pos;
    P_pri = Phi*P_pos*Phi' + Q;
    % Compute the measurement matrix
    H = [0 0 -1 0;
         0 0 0 -1;
         -1 0 0 0;
         0 -1 0 0];
    % Compute the measurement noise covariance
    R = [sigma_Gr^2/(R_N+ h_k_G(1))^2, 0, 0, 0;
        0, sigma_Gr^2 / (R_E+h_k_G(1))^2 / cos(L_k_G(1))^2, 0, 0;
        0, 0,  sigma_Gv^2 ,0;
        0, 0, 0, sigma_Gv^2];
    K = P_pri*H'*inv(H*P_pri*H' + R);
```


TURN OVER

```
% Formulate the measurement innovation
∆_z = [L_k_G(1) - L_k_D(1);
           lambda_k_G(1) - lambda_k_D(1);
           v_N_G(1) - v_N_D(1);
           v_E_G(1) - v_E_D(1);
           ] - H*x_pri;
% Update the state estimates, the error covariance matrix
x_pos = x_pri + K*∆_z;
P_pos = (eye(4) - K*H)*P_pri;
v_N_C(1) = v_N_D(1) - x_pos(1);
v_E_C(1) = v_E_D(1) - x_pos(2);
L_k_C(1) = L_k_D(1)- x_pos(3);
lambda_k_C (1) = lambda_k_D(1) - x_pos(4);

%%%%%%% start kalman filter
for k = 2:length(t)
    [R_N, R_E]= Radii_of_curvature(L_k_G(k));
    Phi = eye(4);
    Phi(3,1) = 0.5/(R_N + h_k_G(k-1));
    Phi(4,2) = 0.5/((R_E + h_k_G(k-1)) * cos(L_k_D(k-1)));
    % define system noise covariance %
    Q = [S_DR*tau_s, 0 , 0.5*S_DR*tau_s^2/(R_N+h_k_G(k-1)), 0;
         0, S_DR*tau_s, 0, ...
            0.5*S_DR*tau_s^2/((R_E+h_k_G(k-1))*cos(L_k_D(k-1)));
         0.5*S_DR*tau_s^2/(R_N+h_k_G(k-1)), 0, ...
            1/3*S_DR*tau_s^3/(R_N+h_k_G(k-1))^2, 0;
         0, 0.5*S_DR*tau_s^2/((R_E+h_k_G(k-1))*cos(L_k_D(k-1))), ...
         0, ...
            1/3*S_DR*tau_s^3/((R_E+h_k_G(k-1))^2*cos(L_k_D(k-1))^2)];

    % Propagate the state estimates and the error covariance matrix
    x_pri = Phi*x_pos;
    P_pri = Phi*P_pos*Phi' + Q;


    % Compute the measurement matrix
    H = [0 0 -1 0;
         0 0 0 -1;
         -1 0 0 0;
         0 -1 0 0];
    % Compute the measurement noise covariance
    R = [sigma_Gr^2/(R_N+ h_k_G(k))^2, 0, 0, 0;
        0, sigma_Gr^2 / (R_E+h_k_G(k))^2 / cos(L_k_G(k))^2, 0, 0;
        0, 0,  sigma_Gv^2 ,0;
        0, 0, 0, sigma_Gv^2];

    % Compute the Kalman gain matrix
    K = P_pri*H'*inv(H*P_pri*H' + R);

    % Formulate the measurement innovation
    ∆_z = [L_k_G(k) - L_k_D(k);
               lambda_k_G(k) - lambda_k_D(k);
               v_N_G(k) - v_N_D(k);
               v_E_G(k) - v_E_D(k);
```

```
                    ] - H*x_pri;

        % Update the state estimates, the error covariance matrix
        x_pos = x_pri + K*Δ_z;
        P_pos = (eye(4) - K*H)*P_pri;
        v_N_C(k) = v_N_D(k) - x_pos(1);
        v_E_C(k) = v_E_D(k) - x_pos(2);
        L_k_C(k) = L_k_D(k)- x_pos(3);
        lambda_k_C (k) = lambda_k_D(k) - x_pos(4);

    end
    LCKF_Solution.v_N = v_N_C;
    LCKF_Solution.v_E = v_E_C;
    LCKF_Solution.L_b = L_k_C;
    LCKF_Solution.lambda_b = lambda_k_C;
    LCKF_Solution.t = t;
end
```

File name: LC_KF_Inte_Close_Solver.m

```
function LCKF_Solution = LC_KF_Inte_Close_Solver(DR_Data, ...
    GNSS_Solution, Psi_C)
    Define_Constants;
    % init DR data param to loop
    h_0 = GNSS_Solution.h_b(1); % check here for every GNSS data
    v_bar(1) = (DR_Data(1,4) + DR_Data(1,5))/2 ;
    t = DR_Data(:,1);
    L_k_D(1) = GNSS_Solution.L_b(1);
    lambda_k_D(1) = GNSS_Solution.lambda_b(1);
    v_N_D(1) = v_bar(1) * cos(psi(1));
    v_E_D(1) = v_bar(1) * sin(psi(1));
    % load GNSS data
    L_k_G = GNSS_Solution.L_b;
    lambda_k_G = GNSS_Solution.lambda_b;
    h_k_G = GNSS_Solution.h_b;
    v_N_G= GNSS_Solution.v_N;
    v_E_G= GNSS_Solution.v_E;

    % init state and covariant
    [R_N, R_E]= Radii_of_curvature(L_k_G(1));
    x_pos = zeros(4,1); % v_N, v_E, L, lambda
    P_pos = [eye(2)*0.1^2, zeros(2);
             zeros(2), [(10/(R_N+h_k_G(1)))^2, 0;
                        0, (10/((R_E+h_k_G(1)) * cos(L_k_G(1))))^2]];
    tau_s = 0.5;
    S_DR = 0.2;
    sigma_Gr = 5;
    sigma_Gv = 0.02;

    %%%%%%% prepare kalman filter parameters
    Phi = eye(4);
    Phi(3,1) = 0.5/(R_N + h_k_G(1));
    Phi(4,2) = 0.5/((R_E + h_k_G(1)) * cos(L_k_D(1)));

    % define system noise covariance %
    Q = [S_DR*tau_s, 0 , 0.5*S_DR*tau_s^2/(R_N+h_k_G(1)), 0;
         0, S_DR*tau_s, 0, ...
            0.5*S_DR*tau_s^2/((R_E+h_k_G(1))*cos(L_k_D(1)));
         0.5*S_DR*tau_s^2/(R_N+h_k_G(1)), 0, ...
            1/3*S_DR*tau_s^3/(R_N+h_k_G(1))^2, 0;
         0, 0.5*S_DR*tau_s^2/((R_E+h_k_G(1))*cos(L_k_D(1))), 0, ...
         1/3*S_DR*tau_s^3/((R_E+h_k_G(1))^2*cos(L_k_D(1))^2)];

    % Propagate the state estimates and the error covariance matrix
    x_pri = Phi*x_pos;
    P_pri = Phi*P_pos*Phi' + Q;

    % Compute the measurement matrix
    H = [0 0 -1 0;
         0 0 0 -1;
         -1 0 0 0;
         0 -1 0 0];
```

```matlab
    % Compute the measurement noise covariance
    R = [sigma_Gr^2/(R_N+ h_k_G(1))^2, 0, 0, 0;
        0, sigma_Gr^2 / (R_E+h_k_G(1))^2 / cos(L_k_G(1))^2, 0, 0;
        0, 0,  sigma_Gv^2 ,0;
        0, 0, 0, sigma_Gv^2];
    K = P_pri*H'*inv(H*P_pri*H' + R);

    % Formulate the measurement innovation
    Δ_z = [L_k_G(1) - L_k_D(1);
               lambda_k_G(1) - lambda_k_D(1);
               v_N_G(1) - v_N_D(1);
               v_E_G(1) - v_E_D(1);
               ] - H*x_pri;

    % Update the state estimates, the error covariance matrix
    x_pos = x_pri + K*Δ_z;
    P_pos = (eye(4) - K*H)*P_pri;
    v_N_C(1) = v_N_D(1) - x_pos(1);
    v_E_C(1) = v_E_D(1) - x_pos(2);
    L_k_C(1) = L_k_D(1)- x_pos(3);
    lambda_k_C (1) = lambda_k_D(1) - x_pos(4);

    %%%%%%% start kalman filter
    for k = 2:length(t)
        % calculate DR solution for this epoch
        v_bar(k) = (DR_Data(k,4) + DR_Data(k,5))/2 ;
        v_N_bar = 0.5*(cos(Psi_C(k)) + cos(Psi_C(k-1)))*v_bar(k);
        v_E_bar = 0.5*(sin(Psi_C(k)) + sin(Psi_C(k-1)))*v_bar(k);
        [R_N,R_E] = Radii_of_curvature(L_k_C(k-1));
        L_k_D(k) = L_k_D(k-1) + (v_N_bar*(t(k) - t(k-1))) / (R_N + h_0);
        lambda_k_D(k) = lambda_k_D(k-1) + (v_E_bar*(t(k) - t(k-1)) / ...
                        ((R_E + h_0) * cos(L_k_D(k))));
        v_N_D(k) = 1.7 * v_N_bar - 0.7 * v_N_D(k-1);
        v_E_D(k) = 1.7 * v_E_bar - 0.7 * v_E_D(k-1);

        % calculate transition matrix
        [R_N, R_E]= Radii_of_curvature(L_k_G(k));
        Phi = eye(4);
        Phi(3,1) = 0.5/(R_N + h_k_G(k-1));
        Phi(4,2) = 0.5/((R_E + h_k_G(k-1)) * cos(L_k_D(k-1)));

        % define system noise covariance
        Q = [S_DR*tau_s, 0 , 0.5*S_DR*tau_s^2/(R_N+h_k_G(k-1)), 0;
             0, S_DR*tau_s, 0, ...
                0.5*S_DR*tau_s^2/((R_E+h_k_G(k-1))*cos(L_k_D(k-1)));
             0.5*S_DR*tau_s^2/(R_N+h_k_G(k-1)), 0, ...
                1/3*S_DR*tau_s^3/(R_N+h_k_G(k-1))^2, 0;
             0, 0.5*S_DR*tau_s^2/((R_E+h_k_G(k-1))*cos(L_k_D(k-1))), ...
                0, ...
             1/3*S_DR*tau_s^3/((R_E+h_k_G(k-1))^2*cos(L_k_D(k-1))^2)];

        % Propagate the state estimates and the error covariance matrix
        x_pri = Phi*x_pos;
        P_pri = Phi*P_pos*Phi' + Q;
```

```
        % Compute the measurement matrix
        H = [0 0 -1 0;
             0 0 0 -1;
             -1 0 0 0;
             0 -1 0 0];
        % Compute the measurement noise covariance
        R = [sigma_Gr^2/(R_N+ h_k_G(k))^2, 0, 0, 0;
            0, sigma_Gr^2 / (R_E+h_k_G(k))^2 / cos(L_k_G(k))^2, 0, 0;
            0, 0,  sigma_Gv^2 ,0;
            0, 0, 0, sigma_Gv^2];

        % Compute the Kalman gain matrix
        K = P_pri*H'*inv(H*P_pri*H' + R);

        % Formulate the measurement innovation
        ∆_z = [L_k_G(k) - L_k_D(k);
                   lambda_k_G(k) - lambda_k_D(k);
                   v_N_G(k) - v_N_D(k);
                   v_E_G(k) - v_E_D(k);
                   ] - H*x_pri;

        % Update the state estimates, the error covariance matrix
        x_pos = x_pri + K*∆_z;
        P_pos = (eye(4) - K*H)*P_pri;
        v_N_C(k) = v_N_D(k) - x_pos(1);
        v_E_C(k) = v_E_D(k) - x_pos(2);
        L_k_C(k) = L_k_D(k)- x_pos(3);
        lambda_k_C(k) = lambda_k_D(k) - x_pos(4);

        if mod(k,5) == 0
            x_pos = [0;0;0;0];
            v_N_D(k) = v_N_C(k);
            v_E_D(k) = v_E_C(k);
            L_k_D(k) = L_k_C(k);
            lambda_k_D(k) = lambda_k_C(k);
        end
    end
    figure
    plot(lambda_k_D*rad_to_deg, L_k_D*rad_to_deg, '-');
    title("DR Result after Closed-loop correction", ' Position');
    xlabel('Latitude (deg)');
    ylabel('Longitude (deg)');
    grid on;
    hold on
    LCKF_Solution.v_N = v_N_C;
    LCKF_Solution.v_E = v_E_C;
    LCKF_Solution.L_b = L_k_C;
    LCKF_Solution.lambda_b = lambda_k_C;
    LCKF_Solution.t = t;
end
```

File name: Result_Plot.m

```
function Result_Plot(Method, L_b, lambda_b, color)
%% Positioning plot
% figure
plot(lambda_b, L_b, '-', Color=color);
title(Method, ' Position');
xlabel('Latitude (deg)');
ylabel('Longitude (deg)');
grid on;
end
```

END OF COURSEWORK