COMP0130: Coursework 1

Integrated Navigation for a Robotic Lawnmower

Weizheng Zhang, 16081188, weizheng.zhang.17@ucl.ac.uk
Naiyuan Cui, 20172095, naiyuan.cui.20@ucl.ac.uk
Yuhang Zhang, 20063072, yuhang.zhang.20@ucl.ac.uk

Introduction

To eventually get the desired output result, we implemented several steps to compute the position, velocity and heading degrees of the lawnmower from the given sensor measurements. The overview of our designed method is showing as following:

1. Compute the single-epoch positioning without initialisation.
   This step we simply calculate the position of the lawnmower using least-square estimation given the pseudo-ranges measurements from the satellites. Because there is no initialisation, we keep iterating the least-square computation until no changes observed. This step prepares to compute the position of the lawnmower for every epoch in the next step.

2. Compute the multi-epoch positioning with initialisation from step 1.
   After we get the position of the lawnmower at time 0, we can extend the same algorithm to Multi-epoch Positioning. Additionally, we use the provided pseudo-range rate measurements to get least-square estimation of the velocity at all the epochs. Although the result is not accurate, this step helps to predict position and velocity from satellite data approximately, which also helps to detect the outliers. We can remove the measurements that has the largest error residual to get filtered data ready for the further computations.

3. Compute the position and velocity from GNSS Kalman Filter with initialisation from step 2.
   After Detecting the outliers and remove the noise measurement data, we can apply Kalman Filter to compute more accurate position and velocity directly from pseudo-ranges and pseudo-range rates. This step helps to reduce the noise/uncertainties and produce the estimates of sensor data over time.

4. Compute smoothed heading.
   The original magnetic compass heading method is accurate, but it is also vulnerable from magnetic anomalies environment. A compounded heading method can provide to reduce both magnetic anomalies and gyro bias effect for stable heading performance.

5. Compute Dead Reckoning (DR).
   This step helps to produce another solution of the position and velocity from the local sensor measurements, such as wheel speed sensor, and magnetic

compass. Through DR estimation, we can construct an additional internal navigation to allow further DR/GNSS integration.

6. Compute corrected position and velocity from applying DR/GNSS Integration. By combining and comparing two solution from DR and GNSS respectively, we can merge to a single optimal solution for both position and velocity. Eventually, it ends up with a corrected result for the latitude, longitude, north velocity, and east velocity.

- **Single-epoch positioning without initialisation**

  The implemented function 'cw1_single_epoch_positioning' takes the pseudo-ranges in table format which is read from file, and it returns the computed position of the lawnmower at time 0.

  The algorithm starts from an initial position of the lawnmower at the centre of the Earth:

$$r_{ea}^{e} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

  Iteratively, it updates the position using least-square computation until the position does not change. We set a break condition for the iteration, which is the distance (error) between the last position and current position.

  Additionally, we need to compute the Cartesian ECEF position of the satellites at time 0 using the provided function 'Satellite_position_and_velocity.m', and we have:

$$r_{e0}^{e}, \_ = \text{Satellite\_position\_and\_velocity()}$$

  At each iteration, we first need to predict the ranges from the estimated position of the lawnmower to each of the satellite from the following:

$$r_{aj}^{-} = \sqrt{\left(C_e^I\left(r_{aj}^{-}\right)r_{e0}^{e} - r_{ea}^{e}\right)^T\left(C_e^I\left(r_{aj}^{-}\right)r_{e0}^{e} - r_{ea}^{e}\right)}$$

  where C_e^I is the Sagnaac effect compensation matrix, and we get compute it from an estimated range r_{aj}^{-}.

$$C_e^I\left(r_{aj}^{-}\right) = \begin{bmatrix} 1 & \frac{\omega_{ie} r_{aj}^{-}}{c} & 0 \\ -\frac{\omega_{ie} r_{aj}^{-}}{c} & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

  To get more accurate prediction of r_{aj}^{-}, we recursively compute it 5 times beginning with an Identity Matrix for C_e^I. ω_{ie} is the Earth rotation rate, with a constant value $7.292115 \times 10^5\ rad\ s^{-1}$. c is the speed of light, with a constant value $299792458\ m\ s^{-1}$.

Then we compute the line-of-sight unit vector from the estimated position of lawnmower to each of the satellite at time 0 as following:

$$u_{aj}^{e} = \frac{C_e^I\left(r_{aj}^{-}\right) - r_{ea}^{e}}{r_{aj}^{-}}$$

Then, we now can formulate the predicted state vector $x^{-} = \begin{bmatrix} r_{ea}^{e} \\ \delta\rho_c^{a-} \end{bmatrix}$, as well as the measurement innovation vector δz^{-} and measurement matrix H_G^e:

$$\delta z^{-} = \tilde{\rho}_a^{j} - r_{aj}^{-} - \delta\rho_c^{a-},$$

$$H_G^e = \begin{bmatrix} -u_{a5}^{e}[x] & -u_{a5}^{e}[y] & -u_{a5}^{e}[z] & 1 \\ -u_{a6}^{e}[x] & -u_{a6}^{e}[y] & -u_{a6}^{e}[z] & 1 \\ \vdots & \ddots & \vdots & \vdots \\ -u_{a30}^{e}[x] & \cdots & -u_{a30}^{e}[z] & 1 \end{bmatrix}$$

where $\tilde{\rho}_a^{j}$ is the sensor data pseudo-range for each satellite at the current epoch, and $\delta\rho_c^{a-}$ is the predicted clock offset so far.

Now, we have everything needed to update the state estimation using unweighted least-squares:

$$X^{+} = \begin{bmatrix} r_{ea}^{e+} \\ \delta\rho_c^{a+} \end{bmatrix} = x^{-} + \left(H_G^{e\,T} H_G^{e}\right)^{-1} H_G^{e\,T}\, \delta z^{-}$$

The result of estimation for the position of the lawnmower at time 0:

- Latitude: $51.509254°$
- Longitude: $-0.161045°$
- Height: 38.825815m

- **Multi-epoch positioning**

  In this step, we implemented function 'cw1_multi_epoch_positioning' takes 3 inputs 1) the pseudo-ranges, 2) pseudo-range rates, and 3) initial position (NED in rad) at time 0. And it returns the computed positions and velocities for every epochs (ECEF) and the outliers detected. The first output of this function 'Solutions' has 8 rows and numbers of columns, in which each column contains the position, velocity, clock offset, and clock drift for each epoch.

  1. Rows $(1{:}3)$ represents the position x, y, and z;
  2. Rows $(4{:}6)$ represents the velocities x, y, and z;
  3. Row 7 and Row 8 represent the clock offset and clock drift respectively.

  This function uses the same method we did for step 1, but repeatedly extends it to compute for every epochs. The changes we made:

1. Instead of only predicting the position at each epoch, we also predict the velocity and update it iteratively.
2. After we optimise the prediction for each epoch, we use the position, velocity, clock offset, and clock drift as the initialisation for next epoch.

The algorithm starts updating four initial states, with an initial position of the lawnmower derived from step 1 r_{ea}^{e}, an initial velocity v_{ea}^{e} with values all zeros, a clock offset $\delta\rho_{c}^{a}$ with value zeros, and a clock drift $\delta\dot{\rho}_{c}^{a}$ with value zero.

At each epoch, we need to compute the Cartesian ECEF position and position of the each of the satellites using the provided function 'Satellite_position_and_velocity.m', and we have:

$$[r_{ej}^{e}, v_{ej}^{e}] = \text{Satellite\_position\_and\_velocity()}$$

The approximated range rates from the lawnmower to each of the satellite is given by:

$$\dot{r}_{aj}^{-} = {u_{aj}^{e}}^{T}\left(C_{e}^{I}\left(r_{aj}^{-}\right)\left(v_{ej}^{e} + \Omega_{ie}^{e} r_{ej}^{e}\right) - \left(v_{ea}^{e} + \Omega_{ie}^{e} r_{ea}^{e}\right)\right)$$

where Ω_{ie}^{e} is the skew symmetric matrix of the earth rotation rate:

$$\Omega_{ie}^{e} = \begin{bmatrix} 0 & -\omega_{ie} & 0 \\ \omega_{ie} & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

Similarly, the predicted state vector for the velocity $x_{v}^{-} = \begin{bmatrix} v_{ea}^{e-} \\ \delta\dot{\rho}_{c}^{a-} \end{bmatrix}$, and the measurement innovation vector δz_{v}^{-}:

$$\delta z^{-} = \tilde{\dot{\rho}}_{a}^{j} - \dot{r}_{aj}^{-} - \delta\dot{\rho}_{c}^{a-},$$

Where $\tilde{\dot{\rho}}_{a}^{j}$ the sensor data pseudo-range rate for each satellite at the current epoch, and $\delta\dot{\rho}_{c}^{a-}$ is the predicted clock drift so far.

Outlier Detection:
Before going to update the velocity, we designed to detect outliers and remove the measurements that has large residuals (outliers).

To check if outliers exist, we compute and compare in 3 steps:
1. Compute the residuals vector:

$$v = \left(H_{G}^{e}\left({H_{G}^{e}}^{T} H_{G}^{e}\right)^{-1} {H_{G}^{e}}^{T} - I_{m}\right)\delta z^{-}$$

where I_m is an Identity Matrix, and m is the number of measurements (number of satellites).

2. Compute the residuals covariance matrix:

$$C_v = \left(I_m - H_G^e\left({H_G^e}^T H_G^e\right)^{-1} {H_G^e}^T\right)\sigma_\rho^2$$

where σ_ρ is the measurement error standard deviation, and we use a constant value 5m.

3. Compute the normalised residuals and compare each with a threshold:

$$Condition(|v_j| > \sqrt{C_{vij}}T)$$

where j represents the j-th measurement, and T is the outlier detection threshold with a value 6.

When any outlier is detected, we keep the apply the outlier which has maximum residuals and remove the j-th measurement from δz^-, δz_v^-, and H_G^e.

Then, we can use unweighted least-square computation to update the velocity as following:

$$x_v^+ = \begin{bmatrix} v_{ea}^{e+} \\ \delta\dot{\rho}_c^{a+} \end{bmatrix} = x_v^- + \left({H_G^e}^T H_G^e\right)^{-1} {H_G^e}^T \delta z_v^-$$

The below figure shows the computed positions for all epochs, and we can see that the result is pretty bad from the multi-epoch positioning so far.

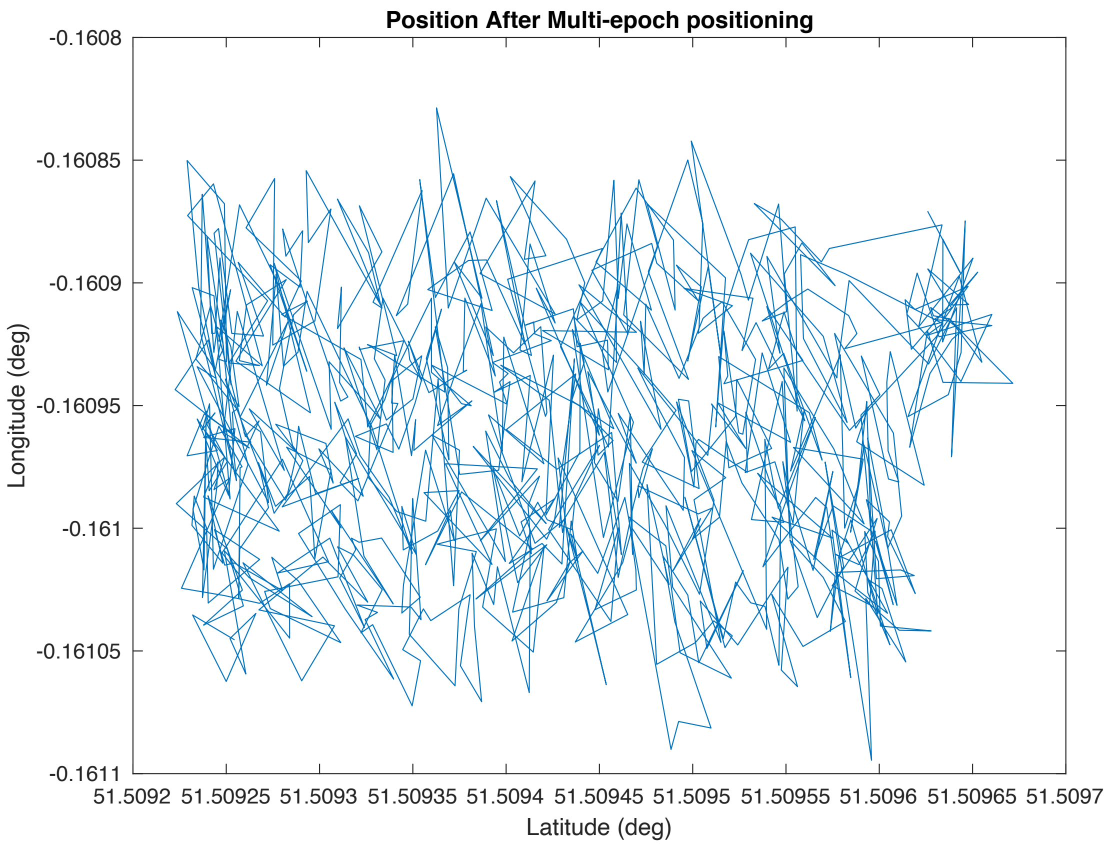

Figure 1: Latitude and longitude after Multi-epoch Positioning

And we also detected outliers in this step, the result shows that the outlier exists at satellite number 7, and the index of that measurement is 3.

- **GNSS Kalman Filter**
  The function ‘cw1_GNSS_kalman_filter_multi.m’ is implemented to
  Now we process GNSS Kalman Filter with an initialised state x_0^+ at time 0 which is computed from Multi-epoch positioning. We also initialise an error covariance matrix P_0^+ from provided noise standard deviation 10 on pseudo-ranges and 0.05 on all pseudo-range rates. It has initial receiver clock offset standard deviation 100000 m and receiver clock drift standard deviation 200 m/s.

$$x_0^+ = \begin{bmatrix} r_{ea}^e \\ v_{ea}^e \\ \delta\rho_c^a \\ \delta\dot{\rho}_c^a \end{bmatrix} = \begin{bmatrix} 3977851 \\ -11181 \\ 4969034 \\ 0 \\ 0 \\ 0 \\ 10009 \\ 100 \end{bmatrix}$$

$$P_0^+ = \begin{bmatrix} 10^2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 10^2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 10^2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.05^2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.05^2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.05^2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 100000 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 200 \end{bmatrix}$$

The transition matrix Φ_{k-1} helps to propagate the error covariance matrix:

$$\Phi_{k-1} = \begin{bmatrix} I_3 & \tau_s I_3 & 0_{3,1} & 0_{3,1} \\ 0_3 & I_3 & 0_{3,1} & 0_{3,1} \\ 0_{1,3} & 0_{1,3} & 1 & \tau_s \\ 0_{1,3} & 0_{1,3} & 0 & 1 \end{bmatrix}$$

Where τ_s is the propagation interval, with 0.5 in this case.

Next, we compute the system noise covariance matrix Q_{k-1}:

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

Where S_a is the acceleration power spectral density, with 5 m^2s^{-3}. $S_{c\phi}^a$ is the clock phase PSD, with 0.01 m^2s^{-3}. S_{cf}^a is the clock frequency PSD, with 0.04 m^2s^{-1}.

At each iteration, the state estimates can be propagated from transition matrix:

$$x_k^- = \Phi_{k-1}x_0^+$$

And the error covariance matrix P_k^- is given by:

$$P_k^- = \Phi_{k-1}P_0^+\Phi_{k-1}^T + Q_{k-1}$$

We can again predict the ranges from the estimated position of the lawnmower to each of the satellite from the following:

$$r_{aj}^- = \sqrt{\left(C_e^I(r_{aj}^-)r_{e0}^e - r_{ea}^e\right)^T\left(C_e^I(r_{aj}^-)r_{e0}^e - r_{ea}^e\right)}$$

The measurement matrix H_k and measurement noise covariance matrix R_k can be computed using the same way as we did in single-epoch positioning.

Then, we compute the Kalman gain matrix as following:

$$H_k = P_k^- H_k^T (H_k P_k^- H_k^T + R_k)^{-1}$$

The measurement innovation vector δz_k^- can be formulated:

$$\delta z_k^- = \begin{bmatrix} \tilde{\rho}_a^j - r_{aj}^- - \delta\rho_c^a \\ \tilde{\dot{\rho}}_a^j - \dot{r}_{aj}^- - \delta\dot{\rho}_c^a \end{bmatrix}$$

Where $\tilde{\rho}_a^j$ is the measured pseudo-range from the j-th satellite to the lawnmower. And $\tilde{\dot{\rho}}_a^j$ is the measured pseudo-range rate from the j-th satellite to the lawnmower.

To update the state estimates, we use the following equation:

$$x_k^+ = x_k^- + K_k \delta z_k^-$$

And the error covariance matrix is updated from:

$$P_k^+ = (I - K_k H_k) P_k^-$$

At the end of each iteration, we set the updated state estimates and error covariance matrix to x_0^+ and P_0^+ for the computation in next iteration.

When we apply Kalman Filter from the all the measurements without outlier remove we get the position trajectory as following:

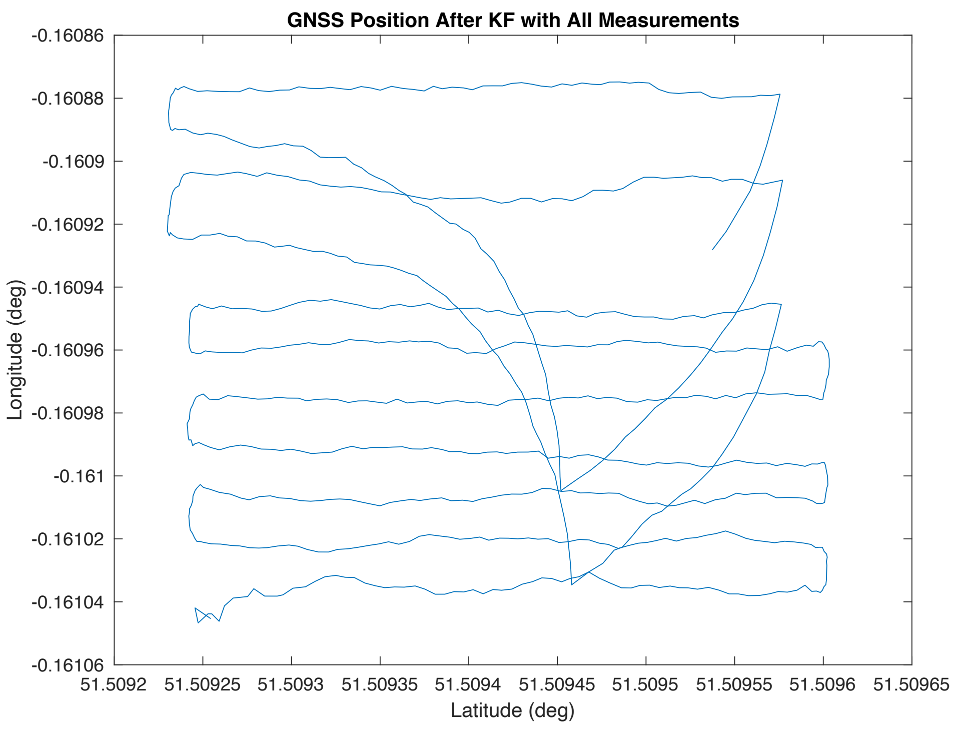

Figure 2: GNSS position solution after Kalman filter

When we apply Kalman Filter from the measurements that has the outliers removed, we get the position trajectory as following:

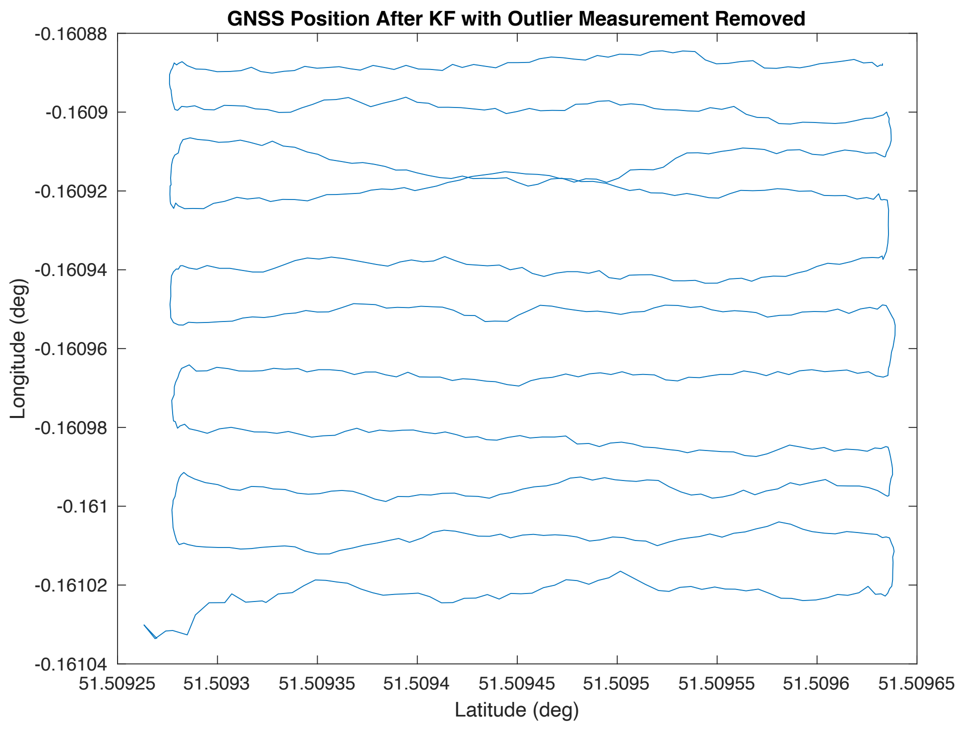

Figure 3: GNSS position solution after Kalman filter with outlier measurement removed

- **Heading**

  There are varies of heading determination methods in the dead reckoning field. As the most traditional method, the magnetic compass is still useful at state of the art. Although its accuracy normally would not dull over a period, it may exhibit error cause from the magnetic environment because the lack of anti-interference ability.

  The heading for dead reckoning plus GNSS is easy to calculate. In the dead-reckoning file, column 7 contains the heading measurements in degrees from the magnetic compass, however, to obtain a better heading called Gyro-Smoothed Magnetic Heading, referencing the lecture 3a, by combining magnetic and the gyroscope velocity. Gyroscopic heading has great short-term stability, but for the long period it may not very reliable and it requires occasional initialization. The combination may not affect the original performance much, but it has good resistance to severe natural interference. The equation below shows the calculation:

$$\tilde{\psi}_{nb,S}(t) = W_M \tilde{\psi}_{nb,M}(t) + (1 - W_M)\left[\tilde{\psi}_{nb,S}(t-\tau) + \int_{t-\tau}^{t} \widetilde{\omega_{ib}}\, dt\right]$$

W_M represents the weighting constant, represent the ratio between magnetic heading and rotation from gyroscope. The function $\int_{t-\tau}^{t} \widetilde{\omega_{ib}}\, dt$ represent Gyro yaw rotation measurement.

- **Dead Reckoning**

  Dead reckoning is a technique used for estimating the current position by measuring and integrating the change in position or velocity. The supplied file Dead_reckoning.csv contains the measurement of heading and average wheels' speed of the lawnmower, which can be used to calculate DR velocity, latitude and longitude over time.

Since the four-wheel speed sensors nearly symmetrically locate about the GNSS antenna. The centre average speed of the lawnmower is calculated as:

$$\bar{v} = \frac{v_1 + v_2 + v_3 + v_4}{4}$$

The average north velocity and east velocity can be calculated as:

$$\begin{pmatrix} \bar{v}_{N,k} \\ \bar{v}_{E,k} \end{pmatrix} = \frac{1}{2}\begin{pmatrix} cos\psi_k + cos\psi_{k-1} \\ sin\psi_k + sin\psi_{k-1} \end{pmatrix} \bar{v}_k$$

where the ψ is the smooth heading, measured by the magnetic compass.

The initial geodetic latitude and longitude are the computed GNSS solution at time 0 in NED state. The geodetic height, h, is also the height from the GNSS solution. Therefore, the rest latitude, L, and longitude, λ, at epoch k can then be computed:

$$L_k = L_{k-1} + \frac{\bar{v}_{N,k}(t_k - t_{k-1})}{R_N + h}$$

$$\lambda_k = \lambda_{k-1} + \frac{\bar{v}_{E,k}(t_k - t_{k-1})}{(R_E + h)cosL_k}$$

where R_N is the meridian radius of curvature and R_E is the transverse radius of curvature, which are calculated via using the function Radii_of_curvature with input latitude L_k.

Also, the damped instantaneous DR velocity can be estimated as:

$$\begin{pmatrix} v_{N,k} \\ v_{E,k} \end{pmatrix} = \begin{pmatrix} 1.7\bar{v}_{N,k} - 0.7v_{N,k-1} \\ 1.7\bar{v}_{E,k} - 0.7v_{E,k-1} \end{pmatrix}$$

In order to avoid the accumulative error of wheel sensors measurement. (Groves, 2021)

Also, the initial DR velocity can be expressed as:

$$\begin{pmatrix} v_{N,0} \\ v_{E,0} \end{pmatrix} = \begin{pmatrix} \bar{v}_0 cos\psi_0 \\ \bar{v}_0 sin\psi_0 \end{pmatrix}$$

- **DR/GNSS Integration**
  The integration navigation offers higher accurate and reliable solution compared with the application of a single navigation technique. For the DR/GNSS integration, the Dead-reckoning sensors enhance the robustness of GNSS and GNSS also eliminate the sensor systematic error and the error caused by the time lag (Groves, 2013).

  The integrated solution is the 2D DR/GNSS navigation solution, which is a 4-state vector including the corrected latitude, longitude and DR velocity. Thus, a 4-state Kalman filter is applied to calculate their corresponding error. The 4-state vector can be expressed as:

$$x = \begin{bmatrix} \delta v_N \\ \delta v_E \\ \delta L \\ \delta \lambda \end{bmatrix}$$

  The initial DR errors are unknown, so the state vector should be all zero. The initial state estimation error covariance matrix is:

$$P_0^+ = \begin{bmatrix} \sigma_v^2 & 0 & 0 & 0 \\ 0 & \sigma_v^2 & 0 & 0 \\ 0 & 0 & \dfrac{\sigma_r^2}{(R_N + h_0)^2} & 0 \\ 0 & 0 & 0 & \dfrac{\sigma_r^2}{(R_N + h_0)^2 cos^2 L_0} \end{bmatrix}$$

  where $\sigma_v^2 = 0.05^2 m^2/s^2$ is the initial velocity error variance and $\sigma_r^2 = 10^2 m^2$ is the initial position error variance. The initial height and latitude are the GNSS solution at time 0 in the State_NED.

The closed integration method is mainly based on the Kalman filter. The transition matrix at epoch $k-1$ can be computed as:

$$\phi_{k-1} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ \dfrac{\tau_s}{R_N + h_{k-1}} & 0 & 0 & 0 \\ 0 & 0 & \dfrac{\tau_s}{(R_E + h_{k-1})cosL_{k-1}} & 0 \end{bmatrix}$$

where $\tau_s = 0.5s$ is the propagation interval, h_{k-1} is GNSS-indicated height and L_{k-1} is the integrated solution of latitude from the last epoch.

The system noise covariance matrix is expressed as:

$$Q_{k-1} =$$

$$\begin{bmatrix} S_{DR}\tau_S & 0 & \dfrac{1}{2}\dfrac{S_{DR}{\tau_S}^2}{R_N + h_{k-1}} & 0 \\ 0 & S_{DR}\tau_S & 0 & \dfrac{1}{2}\dfrac{S_{DR}{\tau_S}^2}{(R_N + h_{k-1})cosL_{k-1}} \\ \dfrac{1}{2}\dfrac{S_{DR}{\tau_S}^2}{R_N + h_{k-1}} & 0 & \dfrac{1}{3}\dfrac{S_{DR}{\tau_S}^3}{(R_N + h_{k-1})^2} & 0 \\ 0 & \dfrac{1}{2}\dfrac{S_{DR}{\tau_S}^2}{(R_E + h_{k-1})cosL_{k-1}} & 0 & \dfrac{1}{3}\dfrac{S_{DR}{\tau_S}^3}{(R_E + h_{k-1})^2cos^2L_{k-1}} \end{bmatrix}$$

The DR velocity error power spectral density (PSD), S_{DR}, is assumed as $0.01m^2/s^3$.

The predicated state at epoch k is:

$$\hat{x}_k^- = \phi_{k-1}\hat{x}_{k-1}^+$$

Also, the predicated error covariance matrix:

$$P_k^- = \phi_{k-1}P_{k-1}^+\phi_{k-1}^T + Q_{k-1}$$

Assuming that the position measurements proceed the velocity measurement, the measurement matrix can be computed as:

$$H_k = \begin{bmatrix} 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & -1 \\ -1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

With the given GNSS position measurements noise standard deviation $\sigma_{Gr} = 10m$ and velocity measurements noise standard deviation $\sigma_{Gr} = 10m$ and $\sigma_{Gv} = 0.02m/s$, the measurement noise covariance matrix can be estimates:

$$R_k = \begin{bmatrix} \frac{\sigma_{Gr}^2}{(R_N + h_k)^2} & 0 & 0 & 0 \\ 0 & \frac{\sigma_{Gr}^2}{(R_N + h_k)^2 cos^2 L_k} & 0 & 0 \\ 0 & 0 & \sigma_{Gv}^2 & 0 \\ 0 & 0 & 0 & \sigma_{Gv}^2 \end{bmatrix}$$

According to the measurements above, the Kalman gain matrix can be estimated:

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

Also, the measurement innovation vector is:

$$\delta z_k^- = \begin{pmatrix} L_k^G - L_k^D \\ \lambda_k^G - \lambda_k^D \\ v_{N,k}^D - v_{N,k}^D \\ v_{E,k}^G - v_{E,k}^G \end{pmatrix} - H_k \hat{x}_k^- = \begin{pmatrix} L_k^G - L_k^D + \delta L_k^- \\ \lambda_k^G - \lambda_k^D + \delta \lambda_k^- \\ v_{N,k}^D - v_{N,k}^D + \delta v_{N,k}^- \\ v_{E,k}^G - v_{E,k}^G + \delta v_{E,k}^- \end{pmatrix}$$

where superscript G denotes the GNSS-indicated solution in State_NED and superscript D denotes the DR-indicated from cw1_dead_reckoning.

Therefore, the updated the estimate can be computed by using the Kalman gain:

$$\hat{x}_k^+ = \hat{x}_k^- + K_k \delta z_k^-$$

Finally, the corrected integration solution at each epoch, k, was calculated as:

$$\begin{pmatrix} L_k^C \\ \lambda_k^C \\ v_{N,k}^C \\ v_{E,k}^C \end{pmatrix} = \begin{pmatrix} L_k^D - \delta L_k^+ \\ \lambda_k^D - \delta \lambda_k^- \\ v_{N,k}^D - \delta v_{N,k}^- \\ v_{E,k}^D - \delta v_{E,k}^- \end{pmatrix}$$

What should be noted the corrected integration solution of latitude will be the input for the next epoch.

Result and discussion

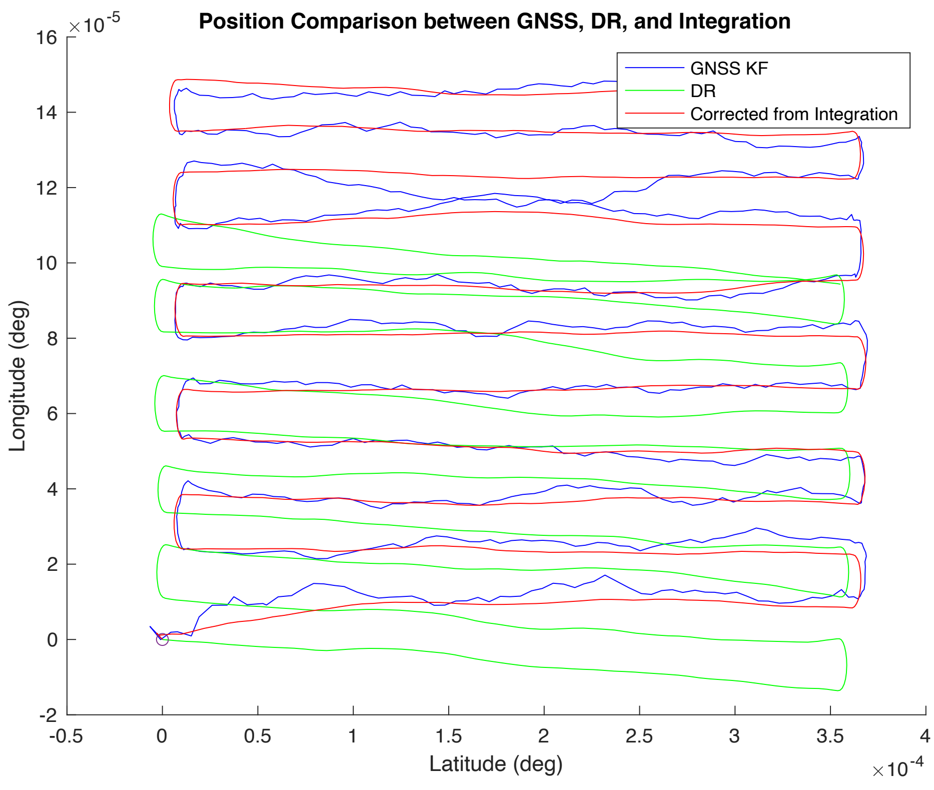

Figure 4: Position comparison between GNSS, DR and Integration solution

As shown in the Figure 4, the start coordinate is set as (0,0). It can be found that all three trajectories have similar S shape. However, the GNSS position has intense noises throughout, which means the applied Kalman filter is not good enough to eliminate the noises from the measurement of GNSS receiver. The DR navigation has relatively smooth change in position, but poor directivity. The deviate of motion may be led by only relying on the interior navigation from the gyroscope and wheel sensors. The GNSS/DR integration brings a more reliable result, the lawnmower only takes a arc trajectory at beginning, followed by the uniform right-angle turning and linear motion.

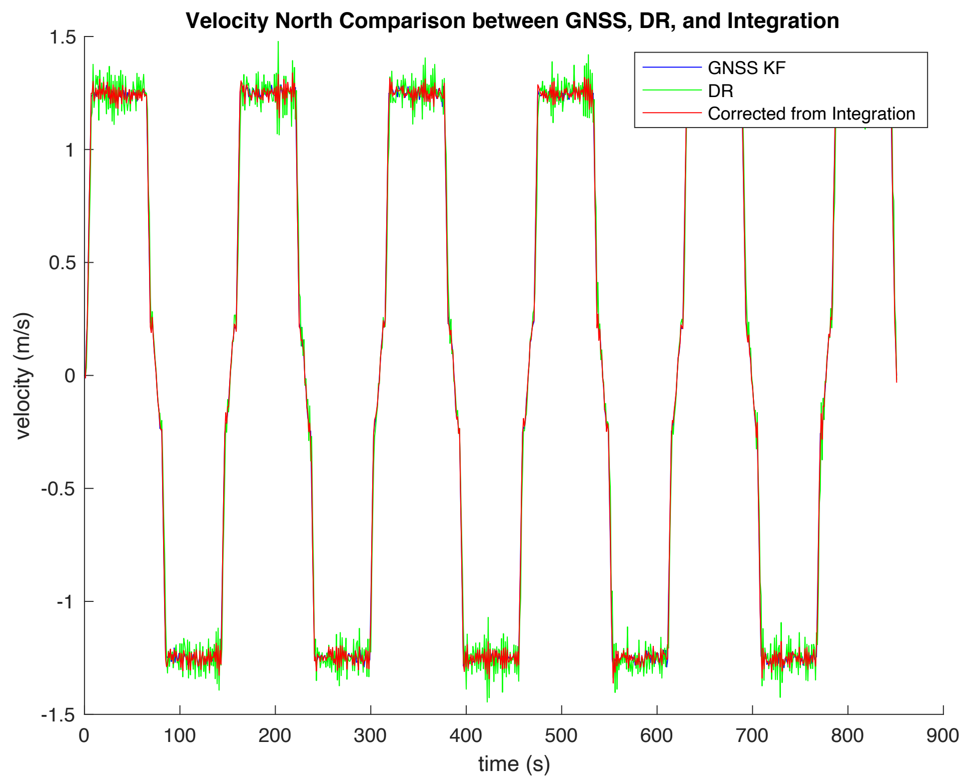

Figure 5: North Velocity comparison between GNSS, DR and Integration solution

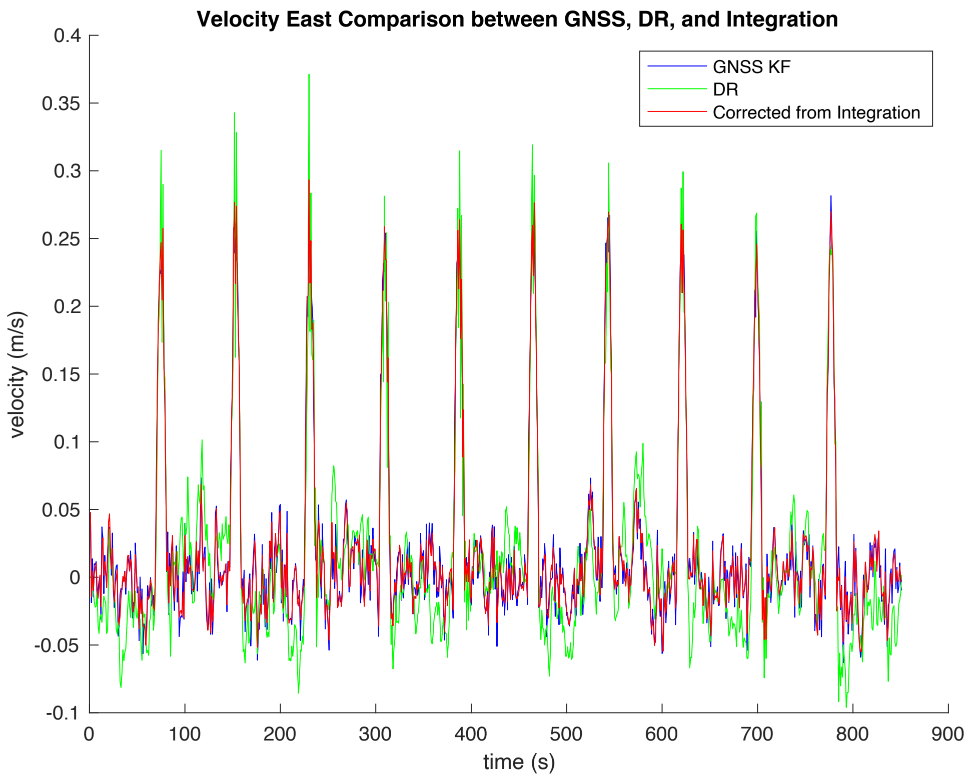

Figure 6: East Velocity comparison between GNSS, DR and Integration solution

Figure 5 and 6 illustrate that both East Velocity and North Velocity has the same result between these three techniques, the DR navigation has the maximal noise. The GNSS and integration solution have no obvious difference, resulting from the large error of GNSS measurement than the error of DR measurement each time point. Therefore, the corrected result is the best overall accuracy obtained by the integration.

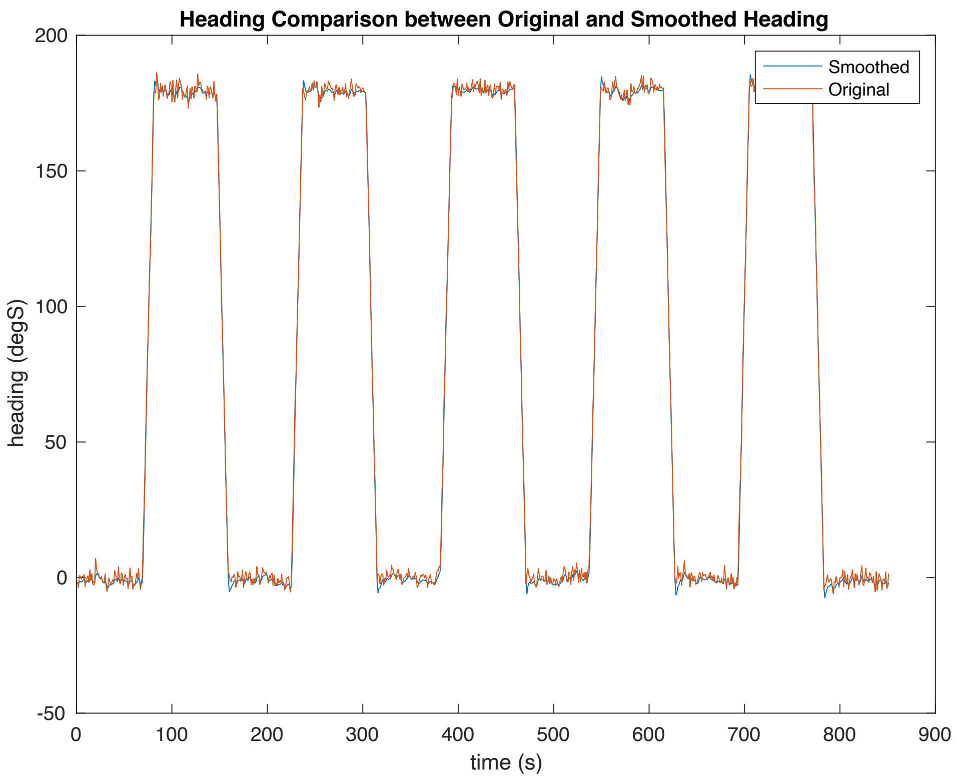

Figure 7: Heading comparison between original and smooth heading

The Figure 7 shows that at no evident magnetic interference situation, even with a bold weighting 0.3, the Gyro-Smoothed Magnetic Heading still did not affect the original heading performance much, on the other hand, its anti-interference ability enhanced significantly. However, the intensity of oscillation during the straight movement has been reduced slightly.

Reference

Groves, P., 2013. *Principles of GNSS, inertial, and multisensor integrated navigation systems*. 2nd ed. pp.647-650.

Groves, P., 2021. *Motion Sensing, Dead Reckoning and Inertial Navigation*. UCL, p.55.

Groves, P., 2021. *Workshop 3: Multisensor Navigation*.

Appendix

Appendix.A

cw1.m

```
% COMP0130 - Group Z
% Cuinai Yuan, Weizheng Zhang, Yuhang Zhang

clear;
Define_Constants;

% ================================================================
load data
%   1          2          3          4          5            6
7
% time s | speed 1 | speed 2 | speed 3 | speed 4 | gyro angular |
heading
D = readtable("Dead_reckoning.csv");

Ranges = readtable("Pseudo_ranges.csv");
Rates = readtable("Pseudo_range_rates.csv");


% ========================= Single-epoch Positioning without
Initialisation
[x_plus] = cw1_single_epoch_positioning(Ranges);

% print the output
[L_b,lambda_b,h_b,v_eb_n] = pv_ECEF_to_NED(x_plus(1:3),[0;0;0]);
lat_deg = L_b * rad_to_deg;
long_deg = lambda_b * rad_to_deg;
disp("Answer:");
disp("lat: " + num2str(lat_deg,'%.6f'));
disp("long: " + num2str(long_deg,'%.6f'));
disp("height: " + num2str(h_b,'%.6f'));
disp("v north: " + v_eb_n(1));
disp("v east: " + v_eb_n(2));
disp("v down: " + v_eb_n(3));

% ================================================= Multi-epoch
Positioning
r_ea_e = [L_b; lambda_b; h_b];
[Solutions, outliers] = cw1_multi_epoch_positioning(Ranges, Rates,
r_ea_e);
% convert to NED
```

```
Solutions_NED = [];
for i=1:size(Solutions,2)
    [L_b,lambda_b,h_b,v_eb_n] =
pv_ECEF_to_NED(Solutions(1:3,i),Solutions(4:6,i));
    lat_deg = L_b * rad_to_deg;
    long_deg = lambda_b * rad_to_deg;
    Solutions_NED = [Solutions_NED; lat_deg long_deg h_b v_eb_n'];
end
% Solutions_NED


% print the outlier detected
for i=1:size(outliers, 1)
    disp("Found outlier at time: ["+outliers(i,1)+"], iter: ["+...
        outliers(i,2)+"], satellite: ["+outliers(i,3)+"], residual:
["+...
        outliers(i,4)+"]");
end
% ======================== GNSS Kalman Filter outlier with all
measurements
state_init = Solutions(:,1);
P_matrix =  zeros(8);
P_matrix(1,1) = 100;
P_matrix(2,2) = 100;
P_matrix(3,3) = 100;
P_matrix(4,4) = 0.05^2;
P_matrix(5,5) = 0.05^2;
P_matrix(6,6) = 0.05^2;
P_matrix(7,7) = 100000;
P_matrix(8,8) = 200;

[States_out] = cw1_GNSS_kalman_filter_multi(Ranges, Rates,
state_init, P_matrix);

% convert to NED
States_out_NED = [];
for i=1:size(States_out,2)
    [L_b,lambda_b,h_b,v_eb_n] =
pv_ECEF_to_NED(States_out(1:3,i),States_out(4:6,i));
    lat_deg = L_b * rad_to_deg;
    long_deg = lambda_b * rad_to_deg;
    States_out_NED = [States_out_NED; lat_deg long_deg h_b v_eb_n'];
end
```

```
% ========================== GNSS Kalman Filter outlier measurement
removed
% remove measurement which has largest residuals
ind_remove = outliers(1,5);
Ranges(:,ind_remove+1) = [];
Rates(:,ind_remove+1) = [];

[States] = cw1_GNSS_kalman_filter_multi(Ranges, Rates, state_init,
P_matrix);

% convert to NED
States_NED = [];
for i=1:size(States,2)
    [L_b,lambda_b,h_b,v_eb_n] =
pv_ECEF_to_NED(States(1:3,i),States(4:6,i));
    lat_deg = L_b * rad_to_deg;
    long_deg = lambda_b * rad_to_deg;
    States_NED = [States_NED; lat_deg long_deg h_b v_eb_n'];
end

% ===================================================== DR/GNSS
Integration
[Heading] = cw1_smooth_heading(D);
[L_K, Lam_K, VN, VE] = cw1_dead_reckoning(States_NED, D, Heading);
% new integration function defined
[correct_DR] = cw1_DR_GNSS_integration(States_NED, D, L_K, Lam_K, VN,
VE);
% convert to degree
correct_Deg = [];
for i=1:size(correct_DR,2)
    lat_deg = correct_DR(2,i) * rad_to_deg;
    long_deg = correct_DR(3,i) * rad_to_deg;
    correct_Deg = [
        correct_Deg;
        correct_DR(1,i) lat_deg long_deg correct_DR(4,i)
correct_DR(5,i)];
end
% ====================================================== output csv
to file

output = [correct_Deg Heading];
csvwrite("Output_Profile.csv", output);
```

Appendix.B

cw1_single_epoch_positioning.m

```
function [x_plus] = cw1_single_epoch_positioning(Ranges)
% cw1_single_epoch_positioning() - compute the position at time 0
using the
% first set of pseudo-range measurements of the pseudo ranges, which
is
% processed without using the initial position. Assume an initial
position
% at the centre of the Earth.
%
% COMP0130: ROBOT VISION AND NAVIGATION
% Workshop 1: Task 1a
%
% This function created 22/02/2021 by Yuhang Zhang
%
% Inputs:
%   Ranges        Pseudo-ranges read from files
%
% Outputs:
%   x_plus        Cartesian position of body frame w.r.t. ECEF frame

% Copyright 2021, Yuhang Zhang
% License: BSD; see license.txt for details

%
======================================================================
====
% Begins

% satellite numbers
numbers = Ranges{1,2:end};

% cartesian ECEF positions of satellites
r_e0_e = zeros(size(numbers,2), 3);

for i=1:size(r_e0_e,1)
    [sat_r_es_e,sat_v_es_e] = Satellite_position_and_velocity(0,
numbers(i));
    r_e0_e(i,:) = sat_r_es_e;
end
% disp(r_e0_e);
% load constants
Define_Constants;

% initialise predicted user position
r_ea_e = [0;0;0];

% initialise line-of-sight unit vector
u_aj_e = zeros(size(numbers,2), 3);

% initialise measurement innovation vector
delta_z = zeros(size(numbers,2), 1);

% initialise measurement matrix
H = ones(size(numbers,2), 4);

x_plus = zeros(4,1);
clock_offset = 0;

disp("################ iteration Starts");
for itr=1:10
    % predict pseudo ranges
    r_aj_minus = zeros(size(numbers,2),1);
    for i=1:size(numbers,2)
        f = eye(3)*r_e0_e(i,:)'-r_ea_e;
        r_a = sqrt(f.'*f);
        for j=1:5
            % Sagnac effect compensation matrix
            C_e_I = [
                1 omega_ie*r_a/c 0;
                -omega_ie*r_a/c 1 0;
                0 0 1
            ];

            f = C_e_I*r_e0_e(i,:)'-r_ea_e;
            r_a = sqrt(f.'*f);
        end
        r_aj_minus(i) = r_a;

        % Sagnac effect compensation matrix
        C_e_I = [
            1 omega_ie*r_aj_minus(i)/c 0;
            -omega_ie*r_aj_minus(i)/c 1 0;
            0 0 1
        ];
        u_a = (C_e_I*r_e0_e(i,:)'-r_ea_e)/r_aj_minus(i);
        u_aj_e(i,:) = u_a';

        delta_z(i) = Ranges{2,i+1} - r_aj_minus(i) - clock_offset;
        H(i,1:3) = -u_aj_e(i,:);

    end

    % predited state vector
    x_minus = [r_ea_e; clock_offset];

    % least squares
    x_plus = x_minus + (H.'*H)\H.' * delta_z;

    err = abs(norm(r_ea_e) - norm(x_plus(1:3)));
    disp("iter["+itr+"] error: " + num2str(err,'%.6f'));

    if(err<0.1)
        break;
    end

    r_ea_e = x_plus(1:3);
end

% Ends
```

Appendix.C

cw1_multi_epoch_positioning.m

```
function [Solutions, outliers] = cw1_multi_epoch_positioning(Ranges,
Rates, r_ea_e)
% cw1_multi_epoch_positioning() - compute the position at multiple
epochs
% using the all rows of the pseudo-range measurements. Starting by an
% initial position at the centre of the Earth.
%
% Reference:
% COMP0130: ROBOT VISION AND NAVIGATION
% Workshop 1: Task 2
%
% This function created 22/02/2021 by Yuhang Zhang
%
% Inputs:
%   Ranges            Pesudo-ranges table read from files.
%   Rates             Pesudo-range rates table read from files.
%   r_ea_e            initial position (in NED).
%
% Outputs:
%   Solutions         Computed positions and velocity for every epochs
with outliers
%                     removed. (ECEF).
%                     row 1: position x
%                     row 2: position y
%                     row 3: position z
%                     row 4: velocity x
%                     row 5: velocity y
%                     row 6: velocity z
%                     row 7: clock offset
%                     row 8: clock drift
%   outliers          Detected outliers.
%                     Column 1: Epoch
%                     Column 2: iteration
%                     Column 3: satellite number
%                     Column 4: residual
%                     Column 5: index of the measurement that the
outlier
%                               is at.

% Copyright 2021, Yuhang Zhang
% License: BSD; see license.txt for details

%
======================================================================
====
% Begins

% load constants
Define_Constants;

% satellite numbers
numbers = Ranges{1,2:end};
times = Ranges{2:end,1};

prev_offset = 0;
prev_drift = 0;
Solutions = zeros(8, size(times,1));
outliers = [];
% r_ea_e = [-33.821075*deg_to_rad;151.188496*deg_to_rad;120];

[r_ea_e,v_ea_e] =
pv_NED_to_ECEF(r_ea_e(1),r_ea_e(2),r_ea_e(3),[0;0;0]);

for epoch=1:size(times,1)
    disp("Epoch ["+epoch+"]");
    % initialise cartesian ECEF positions of satellites
    r_ej_e = zeros(3, size(numbers,2));
    % initialise cartesian ECEF velocities of satellites
    v_ej_e = zeros(3, size(numbers,2));

    for i=1:size(r_ej_e,2)
        [sat_r_es_e,sat_v_es_e] =
Satellite_position_and_velocity(times(epoch), numbers(i));
        r_ej_e(:, i) = sat_r_es_e;
        v_ej_e(:, i) = sat_v_es_e;
    end
    % disp(r_e0_e);

    % initialise line-of-sight unit vector
    u_aj_e = zeros(3, size(numbers,2));

    % initialise measurement innovation vector
    delta_z = zeros(size(numbers,2), 1);
```

```
% initialise measurement innovation vector for velocity
delta_z_v = zeros(size(numbers,2), 1);

% initialise measurement matrix
H = ones(size(numbers,2), 4);

x_plus = zeros(4,1);
clock_offset = prev_offset;
clock_drift = prev_drift;

disp("################ iteration Starts");
for itr=1:10
    % initialise pseudo ranges
    r_aj_minus = zeros(size(numbers,2),1);
    % initialise pseudo range rages
    r_dot_aj_minus = zeros(size(numbers,2),1);
    for i=1:size(numbers,2)
        % predict pseudo ranges
        f = eye(3)*r_ej_e(:,i)-r_ea_e;
        r_a = sqrt(f.'*f);

        for j=1:5
            % Sagnac effect compensation matrix
            C_e_I = [
                1 omega_ie*r_a/c 0;
                -omega_ie*r_a/c 1 0;
                0 0 1
            ];

            f = C_e_I*r_ej_e(:,i)-r_ea_e;
            r_a = sqrt(f.'*f);
        end
        r_aj_minus(i) = r_a;

        % Sagnac effect compensation matrix
        C_e_I = [
            1 omega_ie*r_aj_minus(i)/c 0;
            -omega_ie*r_aj_minus(i)/c 1 0;
            0 0 1
        ];
        u_a = (C_e_I*r_ej_e(:,i)-r_ea_e)/r_aj_minus(i);
        u_aj_e(:,i) = u_a;
        % measurement innovation vector for position
```

```
            delta_z(i) = Ranges{epoch+1,i+1} - r_aj_minus(i) -
clock_offset;
            % measurement matrix
            H(i,1:3) = -u_aj_e(:,i)';

            % predict pseudo range rates
            r_dot_aj_minus(i) = u_aj_e(:,i)'*(...
                C_e_I*(v_ej_e(:,i)+Omega_ie*r_ej_e(:,i))-...
                (v_ea_e+Omega_ie*r_ea_e));

            % measurement innovation vector for velocity
            delta_z_v(i) = Rates{epoch+1,i+1} - r_dot_aj_minus(i) -
clock_drift;

        end

        % =============================================== Outlier
Detection
        % residuals vector
        v = (H*inv(H'*H)*H'-eye(size(numbers,2)))*delta_z;

        sigma_p = 5;
        % residuals covariance matrix
        C = (eye(size(numbers,2))-H*inv(H'*H)*H') * sigma_p^2;

        th = 6;
        max_residual = 0;
        max_ind = -1;
        idx = [];
        for j=1:8
            if (abs(v(j))>sqrt(C(j,j))*th)
                disp(abs(v(j))+">"+sqrt(C(j,j))*th);
                idx = [idx j];

                if (abs(v(j))>max_residual)
                    max_residual = abs(v(j));
                    max_ind = j;

                end
            end
        end


        if (max_ind > -1)
```

```
            outliers = [
                outliers;
                times(epoch) itr numbers(max_ind) abs(v(max_ind))
max_ind
            ];
            delta_z(max_ind) = [];
            H(max_ind,:) = [];
            delta_z_v(max_ind) = [];
        end

        % predited position vector
        x_minus = [r_ea_e; clock_offset];
        % predited velocity vector
        x_minus_v = [v_ea_e; clock_drift];

        % least squares for position
        x_plus = x_minus + (H.'*H)\H.' * delta_z;
        % least squares for position
        x_plus_v = x_minus_v + (H.'*H)\H.' * delta_z_v;

        err = abs(norm(r_ea_e) - norm(x_plus(1:3)));
        disp("End of iter["+itr+"] error: " + num2str(err,'%.6f'));

        if(err<0.1 || max_ind>-1)
            break;
        end

        r_ea_e = x_plus(1:3);
        v_ea_e = x_plus_v(1:3);
    end

    Solutions(:,epoch) = [
        x_plus(1:3);
        x_plus_v(1:3);
        x_plus(4);
        x_plus_v(4)
    ];

    prev_offset = x_plus(4);
    prev_drift = x_plus_v(4);
end

% Ends
```

Appendix.D

cw1_GNSS_kalman_filter_multi.m

```
function [States] = cw1_GNSS_kalman_filter_multi(Ranges, Rates,
state_init, P_matrix)
% cw1_GNSS_kalman_filter_multi() - compute the position and velocity
% directly from the pseudo-range and pseudo-range rate measurements.
%
% Reference:
% COMP0130: ROBOT VISION AND NAVIGATION
% Workshop 1: Task 2
%
% This function created 22/02/2021 by Yuhang Zhang
%
% Inputs:
%   Ranges           Pesudo-ranges table read from files.
%   state_init       initial state (in ECEF).
%                    row 1: position x
%                    row 2: position y
%                    row 3: position z
%                    row 4: velocity x
%                    row 5: velocity y
%                    row 6: velocity z
%                    row 7: clock offset
%                    row 8: clock drift
%   P_matrix         initial error covariance matrix.
%
% Outputs:
%   States           Computed positions and velocities for every
epochs
%                    with outliers removed. (ECEF).

% Copyright 2021, Yuhang Zhang
% License: BSD; see license.txt for details

%
======================================================================
====
% Begins
States = [];

% load constants
Define_Constants;

% time list
times = Ranges{2:end,1};
```

```
% satellite number list
numbers = Ranges{1, 2:end};

% initialize the Kalman filter state vetor estimate
% & error covariance matrix
% [x_est,P_matrix] = Initialise_GNSS_KF;
x_0_plus = state_init;
P_0_plus = P_matrix;


ts = 0.5; % propagation interval (epoch every 0.5s)
% compute the transition matrix
Phi_k_minus_1 = [
    eye(3) ts*eye(3) zeros(3,1) zeros(3,1);
    zeros(3) eye(3) zeros(3,1) zeros(3,1);
    zeros(1,3) zeros(1,3) 1 ts;
    zeros(1,3) zeros(1,3) 0 1
];


% power spectral desity (PSD)
S_a_e = 5;
% clock phase PSD
S_c_phi_a = 0.01;
% clock frequency PSD
S_c_f_a = 0.04;

% compute noise covariance matrix
Q_k_minus_1 = [
    S_a_e*ts^3*eye(3)/3 S_a_e*ts^2*eye(3)/2 zeros(3,1) zeros(3,1);
    S_a_e*ts^2*eye(3)/2 S_a_e*ts*eye(3) zeros(3,1) zeros(3,1);
    zeros(1,3) zeros(1,3) S_c_phi_a*ts+S_c_f_a*ts^3/3 S_c_f_a*ts^2/2;
    zeros(1,3) zeros(1,3) S_c_f_a*ts^2/2 S_c_f_a*ts;
];


for epoch=1:size(times,1)
% for epoch=1:2
    % propagate the state estimates:
    x_k_minus = Phi_k_minus_1*x_0_plus;
%     x_k_minus

    % propagate the error covariance matrix
    P_k_minus = Phi_k_minus_1*P_0_plus*Phi_k_minus_1'+Q_k_minus_1;
    %     P_k_minus

    % initialise postion and velocity of each satellite
    r_ej_e = zeros(3, size(numbers, 2));
    v_ej_e = zeros(3, size(numbers, 2));

    % initialise the range rates
    r_aj_minus = zeros(size(numbers,2),1);

    % initialise the line-of-sight unit vector
    u_aj_e = zeros(3, size(numbers,2));

    % initialise the range rates
    r_dot_aj_minus = zeros(size(numbers,2), 1);

    for i=1:size(numbers,2)
        % predict the ranges from approximated user position
        [sat_r_es_e,sat_v_es_e] =
Satellite_position_and_velocity(times(epoch), numbers(i));
        r_ej_e(:, i) = sat_r_es_e;
        v_ej_e(:, i) = sat_v_es_e;

        f = eye(3) * r_ej_e(:,i) - x_k_minus(1:3);
        r_a = sqrt(f'*f);

        for itr=1:5
            C_e_I = [
                1 omega_ie*r_a/c 0;
                -omega_ie*r_a/c 1 0;
                0 0 1
            ];

            f = C_e_I * r_ej_e(:,i) - x_k_minus(1:3);
            r_a = sqrt(f'*f);
        end
        r_aj_minus(i) = r_a;
        C_e_I = [
            1 omega_ie*r_aj_minus(i)/c 0;
            -omega_ie*r_aj_minus(i)/c 1 0;
            0 0 1
        ];
        % compute the line-of-sight unit vector
        u_aj_e(:,i) = (C_e_I*r_ej_e(:,i) - x_k_minus(1:3)) /
r_aj_minus(i);
```

```
        % predict the range rates from approximated user position
        r_dot_aj_minus(i) = u_aj_e(:,i)' * (...
            C_e_I*(v_ej_e(:,i)+Omega_ie*r_ej_e(:,i))- ...
                (x_k_minus(4:6)+Omega_ie*x_k_minus(1:3)) ...
        );
    end

    % compute the measurement matrix
    H_k = [
        -u_aj_e' zeros(size(numbers,2),3) ...
        ones(size(numbers,2),1) zeros(size(numbers,2),1);
        zeros(size(numbers,2),3) -u_aj_e' ...
        zeros(size(numbers,2),1) ones(size(numbers,2),1)
    ];

    % compute the measurement noise covariance matrix
    sigma_p = 10;
    sigma_r = 0.05;
    R_k = eye(size(numbers,2)*2);
    for i=1:size(R_k,1)
        if i<=size(numbers,2)
            R_k(i,i) = sigma_p^2;
        else
            R_k(i,i) = sigma_r^2;
        end
    end
%     R_k
    % compute the Kalman gain matrix
    K_k = P_k_minus*H_k'/(H_k*P_k_minus*H_k'+R_k);

    % compute the measurement innovation vector
    ranges = Ranges{epoch+1, 2:end}';
    rates = Rates{epoch+1, 2:end}';
    delta_z = [
        ranges-r_aj_minus-x_k_minus(7);
        rates-r_dot_aj_minus-x_k_minus(8)
    ];
%     delta_z

    % update the state estimates
    x_k_plus = x_k_minus + K_k*delta_z;
    %     x_k_plus
    % update the error covariance matrix
    P_k_plus = (eye(8)-K_k*H_k)*P_k_minus;
%     P_k_plus

    % store the state at each epoch
%     [L_b,lambda_b,h_b,v_eb_n] =
pv_ECEF_to_NED(x_k_plus(1:3),x_k_plus(4:6));
%     disp("Answer:");
%     disp("latitude: " + L_b * rad_to_deg );
%     disp("longitude: " + lambda_b * rad_to_deg );
%     disp("height: " + h_b );
%     disp(v_eb_n);
%     States = [States; times(epoch) L_b * rad_to_deg lambda_b *
rad_to_deg h_b v_eb_n'];
    States = [States x_k_plus(1:6)];


    % prepare for next epoch
    x_0_plus = x_k_plus;
    P_0_plus = P_k_plus;
end

% Ends
```

Appendix.E

cw1_smooth_heading.m

```
function [Heading] = cw1_smooth_heading(D)

Define_Constants;
weight = 0.30;
DR = table2array(D);
%inport the heading measurements data and gyroscope angular speed%
H = DR(:,7)*deg_to_rad;
R = DR(:,6);
%pick initial status for heading%
Heading = [H(1)];
heading = H(1);
for i = 2:length(DR)
    %Gyro-Smoothed Magnetic Heading%
    heading = weight*H(i)+(1-weight)*(heading+0.5*(R(i-1)+R(i))*0.5);
    Heading = [Heading; heading];
end
Heading = Heading*rad_to_deg;
end
```

Appendix.F

cw1_dead_reckoning.m

```
function [L_K, Lam_K, VN, VE] = cw1_dead_reckoning(States_NED, D,
Heading)


% load constants
Define_Constants;

% Dead Reckoning
DR = table2array(D);
% average speed
v_mean = (DR(:,2)+DR(:,3)+DR(:,4)+DR(:,5))/4;
% heading
heading = Heading*deg_to_rad;
V_n = [];
V_e = [];
% average velocity
for i = 2:length(heading)
    v_n = (1/2)*(cos(heading(i)) + cos(heading(i-1)))*v_mean(i);
    v_e = (1/2)*(sin(heading(i)) + sin(heading(i-1)))*v_mean(i);
    V_n = [V_n; v_n];
    V_e = [V_e; v_e];
end
% time
t = DR(:,1);
% inital geodectic latitude and longitude at time 0
L = States_NED(1,1)*deg_to_rad;
Lam = States_NED(1,2)*deg_to_rad;
% height
h = States_NED(:,3);
% DR latitude and longitude
L_K = [L];
Lam_K = [Lam];
for i = 2:length(t)
    [R_N, R_E] = Radii_of_curvature(L);
    L = L + V_n(i-1)*(t(i)-t(i-1))/(R_N + h(i));
    Lam = Lam + V_e(i-1)*(t(i)-t(i-1))/((R_E + h(i))*cos(L));
    L_K = [L_K; L];
    Lam_K = [Lam_K; Lam];
end
L_K = L_K*rad_to_deg;
Lam_K = Lam_K*rad_to_deg;

% damped instantaneous DR velocity
```

```
damped_vn = v_mean(1)*cos(heading(1));
damped_ve = v_mean(1)*sin(heading(1));
VN = [damped_vn];
VE = [damped_ve];

for i = 1:length(V_n)
    damped_vn = 1.7*V_n(i) - 0.7*damped_vn;
    damped_ve = 1.7*V_e(i) - 0.7*damped_ve;
    VN = [VN; damped_vn];
    VE = [VE; damped_ve];
end
```

Appendix.G

cw1_DR_GNSS_integration.m

```
function [Corrected] = cw1_DR_GNSS_integration(States_NED, D, L_K,
Lam_K, VN, VE)
% load constants
Define_Constants;

Corrected = [];



% define constants ====================================
% velocity uncertainty
sigma_v = 0.05;
% position uncertainty
sigma_r = 10;
% propagation interval
ts = 0.5;
% GNSS position error standard deviation
sig_Gr = 10;
% GNSS velocity error standard deviation
sig_Gv = 0.05;
% DR velocity error power spectral density (PSD)
S_DR = 0.01;
% =====================================================

% declare arrays
times = D{:,1};

lats_G = States_NED(:,1)*deg_to_rad;
longs_G = States_NED(:,2)*deg_to_rad;
v_n_G = States_NED(:,4);
v_e_G = States_NED(:,5);
heights = States_NED(:,3);

lats_D = L_K*deg_to_rad;
longs_D = Lam_K*deg_to_rad;
v_n_D = VN;
v_e_D = VE;

% init states and matrix
x_0_plus = [0;0;0;0];
lat_0 = lats_G(1);
[R_N,R_E]= Radii_of_curvature(lat_0);
h_0 = heights(1);
```

```
P_0_plus = diag([
    sigma_v^2;
    sigma_v^2;
    sigma_r^2/(R_N+h_0)^2;
    sigma_r^2/((R_E+h_0)^2*cos(lat_0)^2)
]);

% initialize for time0
Corrected = [Corrected [
    times(1);
    lats_G(1);
    longs_G(1);
    v_n_G(1);
    v_e_G(1)
]];

for epoch=2:size(times,1)
    h_0 = heights(epoch-1);
%     lat_0 = L(epoch-1)*deg_to_rad;

    % transition matrix
    Phi_0 = eye(4);
    Phi_0(3,1) = ts/(R_N+h_0);
    Phi_0(4,2) = ts/((R_E+h_0)*cos(lat_0));
    % Phi_0

    % noise covariance matrix
    Q_0 = [
        S_DR*ts 0 1/2*S_DR*ts^2/(R_N+h_0) 0;
        0 S_DR*ts 0 1/2*S_DR*ts^2/((R_E+h_0)*cos(lat_0));
        1/2*S_DR*ts^2/(R_N+h_0) 0 1/3*S_DR*ts^3/(R_N+h_0)^2 0;
        0 1/2*S_DR*ts^2/((R_E+h_0)*cos(lat_0)) 0
1/3*S_DR*ts^3/((R_E+h_0)^2*cos(lat_0)^2);
    ];
    % state estimates
    x_k_minus = Phi_0*x_0_plus;
    %error covariance matrix
    P_k_minus = Phi_0*P_0_plus*Phi_0'+Q_0;
%     P_k_minus

    H_k = zeros(4);
    H_k(3,1) = -1;
    H_k(4,2) = -1;
    H_k(1,3) = -1;
```

```matlab
H_k(2,4) = -1;

% meridian radius and transverse radius
lat_k = lats_G(epoch);
[R_N,R_E]= Radii_of_curvature(lat_k);
h_k = heights(epoch);

% measurement noise covariance matrix
R_k = diag([
    sig_Gr^2/(R_N+h_k)^2;
    sig_Gr^2/((R_E+h_k)^2*cos(lat_k)^2);
    sig_Gv^2;
    sig_Gv^2;
]);

% Kalman gain matrix
K_k = P_k_minus*H_k'/(H_k*P_k_minus*H_k'+R_k);

% measurement innovation vector
d_z = [
    (lats_G(epoch)-lats_D(epoch));
    (longs_G(epoch)-longs_D(epoch));
    v_n_G(epoch) - v_n_D(epoch);
    v_e_G(epoch) - v_e_D(epoch);
] - H_k*x_k_minus;

% update state estimates
x_k_plus = x_k_minus + K_k*d_z;
% update the error covariance matrix
P_k_plus = (eye(4)-K_k*H_k)*P_k_minus;

x_0_plus = x_k_plus;
P_0_plus = P_k_plus;

% compute corrected data
lat_C = lats_D(epoch) - x_k_plus(3);
long_C = longs_D(epoch) - x_k_plus(4);
v_nk_C = v_n_D(epoch) - x_k_plus(1);
v_ek_C = v_e_D(epoch) - x_k_plus(2);

Corrected = [Corrected [
    times(epoch);
    lat_C;
    long_C;
```

```
        v_nk_C;
        v_ek_C
    ]];

    % prepare for next epoch
    lat_0 = lat_C;

end
```