COMP0130 - ROBOT VISION AND NAVIGATION

Coursework 1: Integrated Navigation for a Robotic Lawnmower

Name: **Yinji Zhu** ID: **18062537** Email: **ucaby22@ucl.ac.uk**

February 8, 2019

Introduction

The Coursework assumes to have a robotic lawnmower equipped with a GNSS receiver, wheel speed sensors, magnetic compass and a low-cost MEMS gyroscope. Using the values detected by these sensors, the position, velocity and speed of the lawnmower are able to be determined. In this case, an efficient route of the lawnmower will be able to be determined. The simulated data from all of these sensors have been provided with errors. The details of the supplied files with simulated data and the sensor specifications, which are about the features of the sensors such as the range of errors, are provided as well. Furthermore, several Matlab files have been provided as supplied software resources. The purpose is using the sensor data to compute the best possible horizontal position, horizontal velocity and heading solution for the lawnmower at each point in time.

Methods

To develop the Matlab codes as the requirements, several Matlab functions which have been provided need to be used.

Satellite_position_and_velocity.m is used to obtain the correct satellite positions and velocities in Cartesian Earth-centred Earth-fixed (ECEF) coordinates. The input values of it are current simulation time and the number of satellite. The return values of it are the position and the velocity of the satellite [1].

pv_ECEF_to_NED.m is used to compute geodetic, latitude, longitude and height and Earth-referenced velocity in the north, east and down directions by inputting the Cartesian ECEF position and Earth-referenced velocity resolved in the ECEF frame [1].

pv_NED_to_ECEF.m is used to compute the Cartesian ECEF position and Earth-referenced velocity resolved in the ECEF frame by inputting geodetic, latitude, longitude and height and Earth-referenced velocity in the north, east and down directions [1].

Radii_of_curvature.m is used to compute the meridian and transverse radii of curvature by inputting geodetic latitude in rad [1].

Skew_symmetric.m is used to compute the skew-symmetric matrix of the input vector [1].

To complete the whole process of the task, a Matlab script, **main.m**, was developed, shown in Listing 1.

The basic idea is: Firstly, to avoid the successive errors, the initial value needs to be recalculated. Using least-squares estimation, the initial position, velocity, receiver clock offset, and receiver clock drift are obtained. These values can be regarded as the initial value of the Kalman filter state vector estimate. Secondly, since 8 satellites are used to measure position and velocity and the states are changing with time, the sequential least squares are used in this part. Furthermore, since the Kalman filter is in the environment of linear, Gaussian, and have white noise and the real measurement of lawnmower is nonlinear, nonGaussian and have time-corrected noise. Consequently, the extended Kalman filter is used in this stage. Using the dataset given in Pseudo_ranges.csv and Pseudo_range_rates.csv, the whole GNSS solutions can be obtained. Thirdly, the calculation of heading is based on the Kalman filter integration of gyroscope and magnetic compass, where magetometer heading solutions help to correct gyro heading. Fourthly, after knowing the height in GNSS and corrected heading, the dead reckoning instantaneous position and velocity can be obtained. Since the odometer counts wheel rotations to calculate the distance travelled, the output speed is the average speed rather than the instantaneous speed. Consequently, the instantaneous speed needs to be recalculated. Finally, the integration value is the integrated DR/GNSS navigation solution using Kalman filtering, which uses the difference between GNSS and DR to estimates DR position and velocity errors. Adding the errors to DR positions and velocities, the corrected DR navigation solution is obtained, which is the final integration values.

The detailed five core steps can be seen be seen below:

Step 1: The initialization of the position, velocity, offset and drift

The function **Clock_offset_drift.m** was developed without inputs in this step. The return value is a vector which contains the Cartesian ECEF position, Cartesian ECEF velocity, receiver clock offset standard deviation and receiver clock drift standard deviation. At the beginning of the function, the return parameters and other parameters such as Cartesian ECEF positions of the satellites and Cartesian ECEF velocities of the satellites are initialized to distribute memory space for these variables. After that, the Cartesian ECEF positions of the satellites and the Cartesian ECEF velocities of the satellites will be calculated by the function **Satellite_position_and_velocity.m**.

After all the variables are initialized and the ECEF satellite position and the ECEF satellite velocity are captured, a 20 times iteration is developed to calculate the position, velocity, receiver clock offset and receiver clock drift. The function of the iteration is to reduce the linearisation error, 20 times are large enough to converge the solution. Since only the initial solution needs to be considered, the epoch is set to 1. In the iteration, firstly, the ranges from the approximate user position to each satellite, $\hat{r}_{aj}^{-}$, will be calculated by the formula, shown in Equation 1 [2].

$$\hat{r}_{aj}^{-} = \sqrt{[\mathbf{C}_e^I(\hat{r}_{aj}^{-})\hat{\mathbf{r}}_{ej}^e - \hat{\mathbf{r}}_{ea}^{e-}]^T[\mathbf{C}_e^I(\hat{r}_{aj}^{-})\hat{\mathbf{r}}_{ej}^e - \hat{\mathbf{r}}_{ea}^{e-}]} \tag{1}$$

In the formula, the $\mathbf{C}_e^I(\hat{r}_{aj}^-)$ is calculated by the formula, shown in Equation 2 [2].

$$\mathbf{C}_e^I(\hat{r}_{aj}^-) = \begin{bmatrix} 1 & \omega_{ie} r_{aj}/c & 0 \\ -\omega_{ie} r_{aj}/c & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2}$$

In Equation 1, the values of C and $\hat{r}_{aj}^-$ also need iterations to update the accurate ranges.

Secondly, the line-of-sight unit vector from the approximate user position to each satellite, $\mathbf{u}_{aj}^e$, will be calculated by the formula, shown in Equation 3 [2].

$$\mathbf{u}_{aj}^e = \frac{\mathbf{C}_e^I(\hat{r}_{aj}^-)\hat{\mathbf{r}}_{ej}^e - \hat{\mathbf{r}}_{ea}^{e-}}{\hat{r}_{aj}^-} \tag{3}$$

Thirdly, the range rates from the approximate user position to each satellite, $\hat{\dot{r}}_{aj}^-$, will be calculated by the formula, shown in Equation 4, where $\mathbf{\Omega}_{ie}^e = \begin{bmatrix} 0 & -\omega_{ie} & 0 \\ \omega_{ie} & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$ [2].

$$\hat{\dot{r}}_{aj}^- = \hat{\mathbf{u}}_{aj}^{e-\mathbf{T}}[\mathbf{C}_e^I(\hat{r}_{aj}^-)(\hat{\mathbf{v}}_{ej}^e + \mathbf{\Omega}_{ie}^e \hat{\mathbf{r}}_{ej}^e) - (\hat{\mathbf{v}}_{ea}^{e-} + \mathbf{\Omega}_{ie}^e \hat{\mathbf{r}}_{ea}^{e-})] \tag{4}$$

Fourthly, the position $\hat{\mathbf{r}}_{ea}^e$ and the receiver clock offset will be updated by the formula shown in Equation 5, where there are $\mathbf{H}_G^e = \begin{bmatrix} -u_{aj,x}^e & -u_{aj,y}^e & -u_{aj,z}^e & 1 \\ \vdots & \vdots & \vdots & 1 \end{bmatrix}$, $\delta\mathbf{z}^- = \begin{bmatrix} \widetilde{\dot{\rho}}_a^{\,j} - \hat{\dot{r}}_{aj}^- - \delta\hat{\dot{\rho}}_c^{a-} \\ \vdots \end{bmatrix}$ and $\hat{\mathbf{x}}^- = \begin{bmatrix} \hat{\mathbf{r}}_{ea}^{e-} \\ \delta\hat{\rho}_c^{a-} \end{bmatrix}$ [2].

$$\begin{bmatrix} \hat{\mathbf{r}}_{ea}^{e+} \\ \delta\hat{\rho}_c^{a+} \end{bmatrix} = \hat{\mathbf{x}}^+ = \hat{\mathbf{x}}^- + (\mathbf{H}_G^{e\mathbf{T}} \mathbf{H}_G^e)^{-1} \mathbf{H}_G^{e\mathbf{T}} \delta\mathbf{z}^- \tag{5}$$

Also, the outlier needs to be detected to have more precise estimation. The residuals vector is shown in Equation 6:

$$\mathbf{v} = \left[\mathbf{H}_G^e \left(\mathbf{H}_G^{e\mathrm{T}} \mathbf{H}_G^e\right)^{-1} \mathbf{H}_G^{\mathrm{T}} - \mathbf{I}_m\right] \delta\mathbf{z}^- \tag{6}$$

The residuals convariance matrix is calculated using the Equation 7:

$$\mathbf{C}_{\mathrm{v}} = \left[\mathbf{I}_m - \mathbf{H}_G^e \left(\mathbf{H}_G^{e\mathrm{T}} \mathbf{H}_G^e\right)^{-1} \mathbf{H}_G^{e\top}\right] \sigma_\rho^2 \tag{7}$$

Comparing the values of $|v_j|$ and $\sqrt{C_{vjj}}T$, where T is the outlier detection threshold, if $|v_j|$ is larger than $\sqrt{C_{vjj}}T$ the position of the largest difference will be regarded as the outlier satellite. In this epoch, all difference values are less than 0, so it does not need to recalculate the position.

Finally, the velocity $\hat{\mathbf{v}}_{ea}^e$ and the receiver clock drift solution will be updated by the formula shown in Equation 8, where $\delta\mathbf{z}^- = \begin{bmatrix} \widetilde{\dot{\rho}}_a^{\,j} - \hat{\dot{r}}_{aj}^- - \delta\hat{\dot{\rho}}_c^{a-} \\ \vdots \end{bmatrix}$ and $\hat{\mathbf{x}}^- = \begin{bmatrix} \hat{\mathbf{v}}_{ea}^{e-} \\ \delta\hat{\dot{\rho}}_c^{a-} \end{bmatrix}$ [2]. The $\mathbf{H}_G^e$ is the same as the one used to calculate the position and the receiver clock offset.

$$\begin{bmatrix} \hat{\mathbf{v}}_{ea}^{e+} \\ \delta\hat{\dot{\rho}}_c^{a+} \end{bmatrix} = \hat{\mathbf{x}}^+ = \hat{\mathbf{x}}^- + (\mathbf{H}_G^{e\mathbf{T}} \mathbf{H}_G^e)^{-1} \mathbf{H}_G^{e\mathbf{T}} \delta\mathbf{z}^- \tag{8}$$

After the iteration is finished, the obtained position, velocity, receiver clock offset and receiver clock drift solution will be returned be taken as the initial values of the position, velocity, receiver clock offset and receiver clock drift.

The code of the function, **Clock_offset_drift.m**, is shown in Listing 2.

Step 2: The calculation of GNSS position and velocity by Kalman Filter

To complete this step, a function named **GNSS_extended_Kalman_filter.m**, will be developed with inputting the initial values of the position, velocity, receiver clock offset and receiver clock drift.

In the function, firstly, the state estimates, $\hat{\mathbf{x}}_k$, and the error covariance, $\mathbf{P}_k$, will be initialized. The initial $\hat{\mathbf{x}}_0^+$ will be equal to the input value, [position, velocity, offset, drift]$^{\mathbf{T}}$. The initial $\mathbf{P}_0^+$ will be equal to $\begin{bmatrix} 100 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 100 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 100 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.01 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.01 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.01 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 100 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.01 \end{bmatrix}$ [3].

Secondly, the transition matrix, $\mathbf{\Phi}_{k-1}$, will be calculated by the formula shown in Equation 9, where the propagation interval τ equals to 0.5 [3].

$$\mathbf{\Phi}_{k-1} = \begin{bmatrix} \mathbf{I}_3 & \tau_s\mathbf{I}_3 & \mathbf{0}_{3,1} & \mathbf{0}_{3,1} \\ \mathbf{0}_3 & \mathbf{I}_3 & \mathbf{0}_{3,1} & \mathbf{0}_{3,1} \\ \mathbf{0}_{1,3} & \mathbf{0}_{1,3} & 1 & \tau_s \\ \mathbf{0}_{1,3} & \mathbf{0}_{1,3} & 0 & 1 \end{bmatrix} \tag{9}$$

Thirdly, the system noise convariance matrix, $\mathbf{Q}_{k-1}$, will be calculated by the formula shown in Equation 10, where the acceleration PSD S_a is 5 m^2s^{-3}, the clock phase PSD is 0.01 m^2s^{-3}, and the clock frequency PSD is 0.04 m^2s^{-1} [3].

$$\mathbf{Q}_{k-1} = \begin{bmatrix} \frac{1}{3}S_a\tau_s^3\mathbf{I}_3 & \frac{1}{2}S_a\tau_s^2\mathbf{I}_3 & \mathbf{0}_{3,1} & \mathbf{0}_{3,1} \\ \frac{1}{2}S_a\tau_s^2\mathbf{I}_3 & S_a\tau_s\mathbf{I}_3 & \mathbf{0}_{3,1} & \mathbf{0}_{3,1} \\ \mathbf{0}_{1,3} & \mathbf{0}_{1,3} & S^a_{c\phi}\tau_s + \frac{1}{3}S^a_{cf}\tau_s^3 & \frac{1}{2}S^a_{cf}\tau_s^2 \\ \mathbf{0}_{1,3} & \mathbf{0}_{1,3} & \frac{1}{2}S^a_{cf}\tau_s^2 & S^a_{cf}\tau_s \end{bmatrix} \tag{10}$$

After that, there are 851 epochs to calculate positions and velocities from time 0 to 425s. In each epoch, firstly, the transition matrix is used to propagate the state estimates, shown in Equation 11 [3].

$$\hat{\mathbf{x}}_k^- = \mathbf{\Phi}_{k-1}\hat{\mathbf{x}}_{k-1}^+ \tag{11}$$

Secondly, the transition matrix is used to propagate the error covariance matrix, shown in Equation 12 [3].

$$\mathbf{P}_k^- = \mathbf{\Phi}_{k-1}\mathbf{P}_{k-1}^+\mathbf{\Phi}_{k-1}^{\mathrm{T}} + \mathbf{Q}_{k-1} \tag{12}$$

Thirdly, the ECEF satellite position and the ECEF satellite velocity of each satellite will be calculated by the function **Satellite_position_and_velocity.m** with inputting the time, $0.5 \times (num\_of\_epoch - 1)$, and the code of the satellite. With the ECEF satellite position and the ECEF satellite velocity, the ranges from the approximateuser position to each satellite, $\hat{r}_{aj}^{-}$, the line-of-sight unit vector, $\mathbf{u}_{aj}^{e}$ and the range rates, $\hat{\dot{r}}_{aj}^{-}$, will be able to be worked out by the formula shown in Equation 1 3 and 4. The used $\hat{\mathbf{r}}_{ea}^{e-}$ and $\hat{\mathbf{v}}_{ea}^{e-}$ are in the here are in the propagated state state estimates $\hat{\mathbf{x}}_{k}^{-}$.

Fourthly, the measurement matrix $\mathbf{H_k}$ will be calculated by the formula shown in Equation 13 [3].

$$\mathbf{H_k} = \begin{bmatrix} -u_{a4,x}^{e} & -u_{a4,y}^{e} & -u_{a4,z}^{e} & 0 & 0 & 0 & 1 & 0 \\ -u_{a4,x}^{e} & -u_{a5,y}^{e} & -u_{a5,z}^{e} & 0 & 0 & 0 & 1 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ -u_{a30,x}^{e} & -u_{a30,y}^{e} & -u_{a30,z}^{e} & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -u_{a4,x}^{e} & -u_{a4,y}^{e} & -u_{a4,z}^{e} & 0 & 1 \\ 0 & 0 & 0 & -u_{as,x}^{e} & -u_{as,y}^{e} & -u_{as,z}^{e} & 0 & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & -u_{a30,x}^{e} & -u_{a30,y}^{e} & -u_{a30,z}^{e} & 0 & 1 \end{bmatrix} \tag{13}$$

Fifthly, the measurement noise convariance matrix $\mathbf{R_k}$ will be calculated by the formula shown in Equation 14 [3].

$$\mathbf{R_k} = \begin{bmatrix} \sigma_\rho^2 & 0 & \cdots & 0 & 0 & 0 & \cdots & 0 \\ 0 & \sigma_\rho^2 & \cdots & 0 & 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \sigma_\rho^2 & 0 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 & \sigma_r^2 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 & 0 & \sigma_r^2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 & 0 & 0 & \cdots & \sigma_r^2 \end{bmatrix} \tag{14}$$

where the noise standard deviation on pseudo-range measurements $\sigma_\rho = 10$m and the noise standard deviation on pseudo-range rate measurements $\sigma_r = 0.05$ m/s.

Sixth, the Kalman gain matrix $\mathbf{K}_k$ will be calculated by the formula shown in Equation 15 [3].

$$\mathbf{K}_k = \mathbf{P}_k^{-}\mathbf{H}_k^{\mathrm{T}}\left(\mathbf{H}_k\mathbf{P}_k^{-}\mathbf{H}_k^{\mathrm{T}} + \mathbf{R}_k\right)^{-1} \tag{15}$$

Seventh, the measurement innovation vector will be formulated, which is shown in Equa-

tion 16 [3].

$$\delta \mathbf{z}^- = \begin{pmatrix} \widetilde{\rho}_a^4 - \hat{r}_{a4}^- - \delta\hat{\rho}_c^{a-} \\ \widetilde{\rho}_a^5 - \hat{r}_{a5}^- - \delta\hat{\rho}_c^{a-} \\ \vdots \\ \widetilde{\rho}_a^{30} - \hat{r}_{a30}^- - \delta\hat{\rho}_c^{a-} \\ \tilde{\rho}_a^5 - \hat{r}_{a4}^- - \delta\hat{\rho}_c^{a-} \\ \tilde{\rho}_a^5 - \hat{r}_{a4}^- - \delta\hat{\rho}_c^{a-} \\ \vdots \\ \tilde{\rho}_a^{30} - \hat{r}_{a30}^- - \delta\hat{\rho}_c^{a-} \end{pmatrix} \tag{16}$$

Eighth, knowing the values of δ_z^-, the outlier satellites needs to be detected. The calculation of the residuals vector and residuals convariance matrix still use the Equation 6 and Equation 7, where $\mathbf{H}_G^e = \begin{bmatrix} -u_{aj,x}^e & -u_{aj,y}^e & -u_{aj,z}^e & 1 \\ \vdots & \vdots & \vdots & 1 \end{bmatrix}$. If $|v_j|$ is larger than $\sqrt{C_{vjj}}T$, the position of the largest difference between them will be regarded as the outlier satellite. In this case, the data of the outlier satellite needs to be deleted. It means that the H_k changes from a 16 by 8 matrix to a 14 by 8 matrix and R_k changes from a 16 by 16 matrix to a 14 by 14 matrix. The calculated K_k and δ_z^- change from 8 by 16 and 16 by 1 to 8 by 14 and 14 by 1 respectively. Consequently, more accurate error value can be obtained.

Ninth, the state estimates will be updated by the formula shown in Equation 17 [3].

$$\hat{\mathbf{x}}_k^+ = \hat{\mathbf{x}}_k^- + \mathbf{K}_k \delta \mathbf{z}_k^- \tag{17}$$

Tenth, the error covariance matrix will be updated by the formula shown in Equation 18 [3].

$$\mathbf{P}_k^+ = (\mathbf{I} - \mathbf{K}_k \mathbf{H}_k)\, \mathbf{P}_k^- \tag{18}$$

Finally, the position and velocity will be converted from ECEF to NED. The results of the outlier judgements, the NED positions and NED velocities of the platform will be returned as the final results of GNSS values.

The code of this function, **GNSS_extended_Kalman_filter.m**, is shown in Listing 3.

Step 3: Gyro-Magnetometer corrected

The purpose of this step is to calculate the gyroscope heading by the data in the file **Dead_reckoning.csv**, which contains gyroscope angular rate measurements and heading measurements from the magnetic compass.

Knowing the data of gyroscope angular rate, the gyroscope heading will be calculated by equation: $\Psi_k^G = \Psi_{k-1}^G + \tau_s \times \omega_{Gyro}$, where ω_{Gyro} is gyroscope angular rate and Ψ_k^G is magnetic heading solution. During the whole process, an important operation is transforming the units of all the parameters from rad to degree during the calculation. After the gyroscope heading has been worked out, the unit of it will be transformed to rad.

The calculation of Gyro-Magnetometer integration is the same as GNSS, it uses Kalman Filters to correct gyro heading. The transition matrix $\mathbf{\Phi}_{k-1} = \begin{bmatrix} 1 & \tau \\ 0 & 1 \end{bmatrix}$ is used to update

the heading error $\delta\Psi_k^G$. The initial state and error convariance matrix are: $\mathbf{x}_{pre} = \begin{bmatrix}0\\0\end{bmatrix}$ and $\mathbf{P}_0^+ = \begin{bmatrix}0.01^2 & 0\\ 0 & 0.01745329252^2\end{bmatrix}$ respectively. [4].

Secondly, the system noise convariance matrix, $\mathbf{Q}_{k-1}$, will be worked out by the formula shown in Equation 19 [4].

$$\mathbf{Q}_{k-1} = \begin{bmatrix}0.5S_{rg} + \frac{0.5^3}{3}S_{bgd} & \frac{0.5^2}{2}S_{bgd}\\ \frac{0.5^2}{2}S_{bgd} & 0.5S_{bgd}\end{bmatrix} \tag{19}$$

where gyro random noise with PSD S_{rg} equals to 3×10^{-6} and gyro bias variation with PSD S_{bgd} equals to 0.01745329252^2.

Thirdly, using the values above, the state estimates and error convariance matrix can be propagated: $\mathbf{x}_k^- = \mathbf{\Phi}_{k-1}\cdot\mathbf{x}_{pre}$ and $\mathbf{P}_k^- = \mathbf{\Phi}_{k-1}\mathbf{P}_{k-1}\mathbf{\Phi}_{k-1} + \mathbf{Q}_{k-1}$ [4].

Fourthly, measurement innovation $\delta\mathbf{z}_k^-$ will be calculated by the formula shown in Equation 20, with the Ψ_k^M being Magnetic heading solution and the Ψ_k^G being Gyro-derived heading solution [4].

$$\delta\mathbf{z}_k^- = (\Psi_k^M - \Psi_k^G) - \mathbf{H}_k\cdot\mathbf{x}_k^- \tag{20}$$

Fifthly, the Kalman gain matrix $\mathbf{K}_k$ will be calculated by the formula shown in Equation 21, where measurement noise convariance $\mathbf{R}_k = \sigma_M^2$. σ_M is magnetic heading noise variance. Knowing that the standard deviance of magnetic heading is 4°, so $\sigma = 4\times0.01745329252$. Also the measurement matrix is $\mathbf{H}_k = \begin{bmatrix}-1 & 0\end{bmatrix}$. [4]

$$\mathbf{K}_k = \mathbf{P}_k^-\mathbf{H}_k^\mathbf{T}(\mathbf{H}_k\mathbf{P}_k^-\mathbf{H}_k^\mathbf{T} + \mathbf{R}_k)^{-1} \tag{21}$$

Sixth, based on the gotten measurement innovation $\delta\mathbf{z}_k^-$ and the Kalman gain matrix $\mathbf{K}_k$, the heading error $\mathbf{x}_{pre}$ will be able to be updated by the formula shown in Equation 22 [4].

$$\mathbf{x}_k^+ = \mathbf{x}_k^- + \mathbf{K}_k\cdot\delta\mathbf{z}_k^- \tag{22}$$

Finally, the $\mathbf{P}_{k-1}$ will be updated for next epochs by $\mathbf{P}_k^-$, $\mathbf{K}_k$ and $\mathbf{H}_k$. Moreover, the Gyro-derived heading will be added in heading error by formula, $\Psi_k^G = \Psi_k^G - \mathbf{x}_{pre}(1)$. The $\mathbf{x}_{pre}(1)$ is the first element of the $\mathbf{x}_{pre}$. After the errors added in the Gyro-derived heading, the corrected Gyro heading will be returned as results.

The whole process of this step was developed as a function, named **Gyro_heading.m**, which is shown in Listing 4.

Step 4: The calculation of Dead Reckoning position and velocity

To achieve this step, a function named **Dead_Reckoning.m** was developed. The input parameters of this function are GNSS position and Gyro-derived heading, obtained by the function **Gyro_heading.m**. The return values of this function are Dead Reckoning position and velocity.

In this function, firstly, the data in the file **Dead_reckoning.csv** will be taken and the average speed, $\bar{v}_k$, will be worked out. The average speed is the average speed of four wheels since driving wheels sometimes rotate faster than the vehicle moves and the non-driving wheels

sometimes rotate slower than the vehicle moves. In this case, the average velocity between epochs $k-1$ and k will be able to be calculated by the formula shown in Equation 23 [5].

$$\begin{bmatrix} \bar{v}_{N,k} \\ \bar{v}_{E,k} \end{bmatrix} = \frac{1}{2} \begin{bmatrix} cos\Psi_k + cos\Psi_{k-1} \\ sin\Psi_k + sin\Psi_{k-1} \end{bmatrix} \bar{v}_k \tag{23}$$

After that, the latitude and longitude will be worked out. The number of the pair of latitude and longitude is the same as the number of the input Gyro-derived heading. The first pair of latitude and longitude is the same as the first pair of GNSS position input as initialization. The other latitude and longitude under different epoch, L_k and λ_k, will be worked out by the formula shown in Equation 24, where the R_N is the meridian radius of curvature, the R_E is the transverse radius of curvature, and the h is the height, where the value of h is obtained from GNSS current height [5].

$$L_k = L_{k-1} + \frac{\bar{v}_{N,k}(t_k - t_{k-1})}{R_N + h} \qquad \lambda_k = \lambda_{k-1} + \frac{\bar{v}_{E,k}(t_k - t_{k-1})}{(R_E + h)cosL_k} \tag{24}$$

After the latitude and longitude under all the epochs are worked out, the instantaneous DR velocity at each epoch will be calculated. Firstly, the instantaneous velocity at time zero will be set to $v_{N,0} = \bar{v}_0 cos\Psi_0$ and $v_{E,0} = \bar{v}_0 sin\Psi_0$ [5]. Secondly, the instantaneous DR velocity at each epoch will be worked out by the formula shown in Equation 25 [5]. In this case, the damping factor d is set to 0.6.

$$v_{N,k} = (2-d)\bar{v}_{N,k} - (1-d)v_{N,k-1} \qquad v_{E,k} = (2-d)\bar{v}_{E,k} - (1-d)v_{E,k-1} \tag{25}$$

Finally, the DR position will be [latitude, longitude] and the DR velocity will be $[v_N, v_E]$. The DR position and the DR velocity will be returned as results of this function.

The code of this function, **Dead_Reckoning.m**, is shown in Listing 5.

Step 5: The integration of GNSS and Dead Reckoning

The purpose of this step is computing an integrated horizontal-only DR/GNSS navigation solution using Kalman filtering. To achieve the purpose, a function named **Itegration.m** was developed. The inputs of this function are GNSS position, GNSS velocity, DR position, DR velocity and Gyro-derived heading.

At the beginning of the function, the state estimation error covariance matrix $\mathbf{P}_{pre}$ will be initialized at

$$\mathbf{P}_{pre} = \begin{bmatrix} \sigma_v^2 & 0 & 0 & 0 \\ 0 & \sigma_v^2 & 0 & 0 \\ 0 & 0 & \frac{\sigma_r^2}{(R_N+h_0)^2} & 0 \\ 0 & 0 & 0 & \frac{\sigma_r^2}{(R_E+h_0)^2 cos^2 L_0} \end{bmatrix}$$

In the formula, initial position uncertainty $\sigma_v = 0.05$ and initial position velocity uncertainty $\sigma_r = 10$ [5]. Also, the R_N is the meridian radius of curvature, the R_E is the transverse radius of curvature, and the h_0 is the height.

After that, the process of the integration will be done with a for loop for all time epochs. Firstly, the transition matrix $\mathbf{\Phi}_{k-1}$ will be calculated by the formula below, where the $\tau_s = 0.5$ [5].

$$\mathbf{\Phi}_{k-1} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ \frac{\tau_s}{R_N+h_{k-1}} & 0 & 1 & 0 \\ 0 & \frac{\tau_s}{(R_E+h_{k-1})cosL_{k-1}} & 0 & 1 \end{bmatrix}$$

Secondly, the system noise convariance matrix $\mathbf{Q}_{k-1}$ will be calculated by the formula below, where the DR PSD $S_{DR} = 0.01m^2s^{-3}$ [5].

$$\mathbf{Q}_{k-1} = \begin{bmatrix} S_{DR}\tau_s & 0 & \frac{1}{2}\frac{S_{DR}\tau_s^2}{R_N+h_{k-1}} & 0 \\ 0 & S_{DR}\tau_s & 0 & \frac{1}{2}\frac{S_{DR}\tau_s^2}{(R_E+h_{k-1})cosL_{k-1}} \\ \frac{1}{2}\frac{S_{DR}\tau_s^2}{R_N+h_{k-1}} & 0 & \frac{1}{3}\frac{S_{DR}\tau_s^3}{(R_N+h_{k-1})^2} & 0 \\ 0 & \frac{1}{2}\frac{S_{DR}\tau_s^2}{(R_E+h_{k-1})cosL_{k-1}} & 0 & \frac{1}{3}\frac{S_{DR}\tau_s^3}{(R_E+h_{k-1})^2cos^2L_{k-1}} \end{bmatrix}$$

Thirdly, the state estimate $\mathbf{x}_k^-$ will be propagated by $\mathbf{x}_k^- = \mathbf{\Phi}_{k-1}\mathbf{x}_{k-1}^+$ [5].

Fourthly, the error covariance matrix $\mathbf{P}_k^- = \mathbf{\Phi}_{k-1}\mathbf{P}_{pre}\mathbf{\Phi}_{k-1}^\mathbf{T} + \mathbf{Q}_{k-1}$ [5].

Fifthly, the measurement matrix $\mathbf{H}_k$ will be set to $\mathbf{H}_k = \begin{bmatrix} 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & -1 \\ -1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \end{bmatrix}$ [5].

Sixth, the measurement noise convariance matrix $\mathbf{R}_k$ will be calculated by the formula [5].

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

The GNSS position measurement error standard deviation is calculated by the sum of 1. signal in space error standard deviation, 2. residual ionosphere error standard deviation at zenith, 3. Residual troposphere error standard deviation at zenith, 4. code tracking and multi-path error standard deviation. Consequently, the GNSS position measurement error standard deviation σ_{Gr} is equal to $1 + 2 + 0.2 + 2 = 5.2$m and the GNSS velocity measurement error standard deviation $\sigma_{Gv} = 0.02$ m/s.

Seventh, the Kalman gain matrix $\mathbf{K}_k$ will be worked out by $\mathbf{K}_k = \mathbf{P}_k^-\mathbf{H}_k^\mathbf{T}(\mathbf{H}_k\mathbf{P}_k^-\mathbf{H}_k^\mathbf{T} + \mathbf{R}_k)^{-1}$ [5].

Eighth, the measurement innovation vector $\delta\mathbf{z}_k^-$ will be worked out by the formula below [5].

$$\delta\mathbf{z}_k^- = \begin{bmatrix} L_k^G - L_k^D \\ \lambda_k^G - \lambda_k^D \\ v_{N,k}^G - v_{N,k}^D \\ v_{E,k}^G - v_{E,k}^D \end{bmatrix} - \mathbf{H}_k\mathbf{x}_k^-$$

Ninth, the state estimates will be updated and there will be $\mathbf{x}_k^+ = \mathbf{x}_k^- + \mathbf{K}_k\delta\mathbf{z}_k^-$ [5].

Tenth, the error covariance matrix will be updated and there will be $\mathbf{P}_k^+ = (\mathbf{I} - \mathbf{K}_k\mathbf{H}_k)\mathbf{P}_k^-$ [5].

Finally, the DR solutions will be corrected by errors, of which the formulas are shown in Equation 26.

$$\begin{aligned} L_k^C &= L_k^D - \delta L_k^+ \\ L_k^C &= L_k^D - \delta L_k^+ \\ v_{N,k}^C &= v_{N,k}^D - \delta v_{N,k}^+ \\ v_{E,k}^C &= v_{E,k}^D - \delta v_{E,k}^+ \end{aligned} \tag{26}$$

There are 6 columns in the results matrix. The first column contains the information about time. The second column contains the corrected latitude. The third column contains the corrected longitude. The fourth column contains the corrected velocity to North. The fifth column contains the corrected velocity to East. The last column contains the input Gyro-derived heading as a part of results. The results matrix will be used as return value of this function.

At the end of this function, several figures are plotted as results.

The code of this function, **Itegration.m**, is shown in Listing 6.

Results and Discussion

The position of GNSS without outlier detection can be seen in fig.1. As what can be seen in the graph, some positions are weird since they move to the positions which are far from neighbors. Consequently, outliers need to be detected and deleted then.

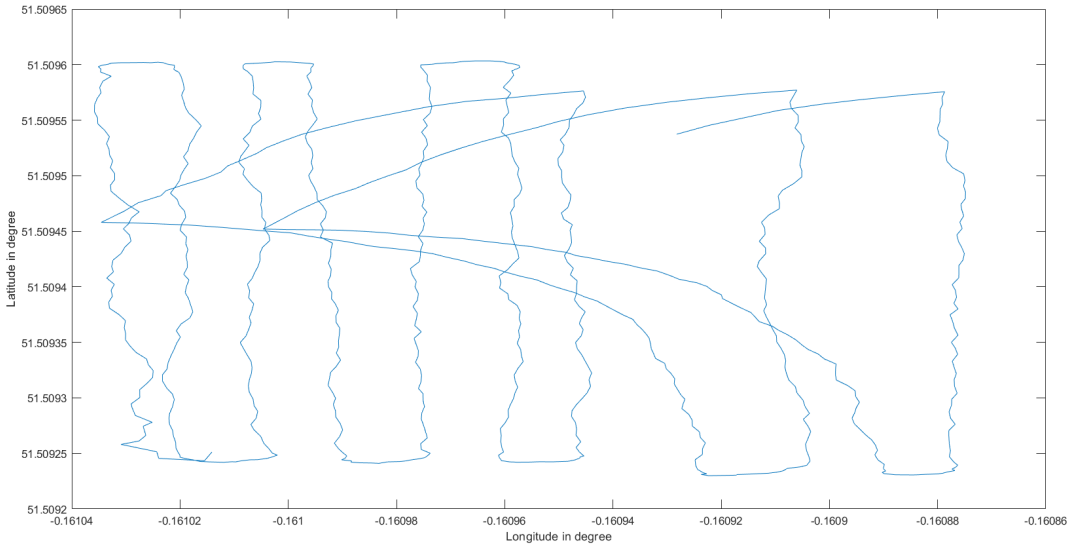

Figure 1: The position of GNSS without outlier detection

The position of GNSS with the outlier detection threshold equals to 6 can be seen in fig.2. As what can be seen in the graph, the position graph still has some confusing points, which means that the outliers have not been deleted thoroughly.

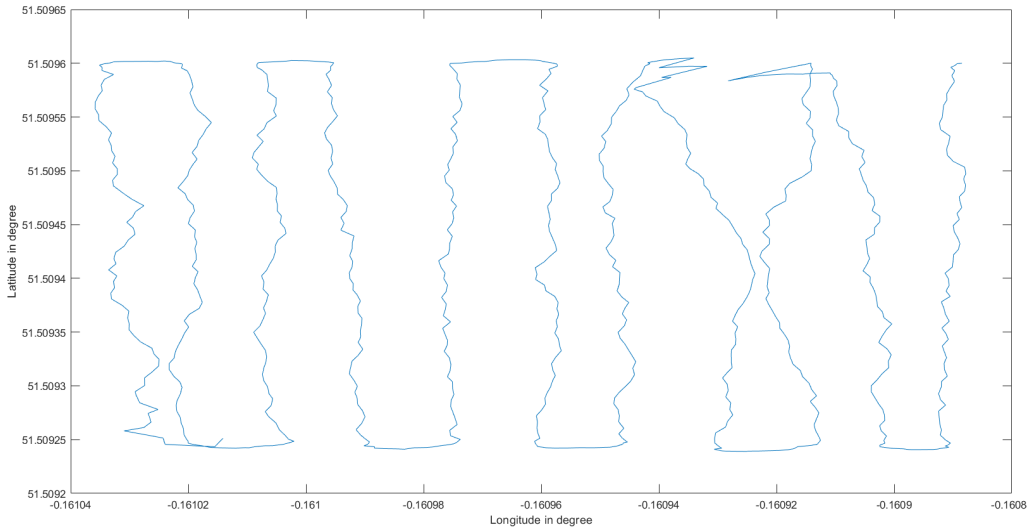

Figure 2: The position of GNSS with T = 6

In this case, the threshold is changed to 4, since it can have a better detection of outliers. As what can be seen in fig.3, the trajectory is smooth enough. If the threshold changes less, it will misjudge some measurements of satellites. Consequently, the threshold value is set to 4.

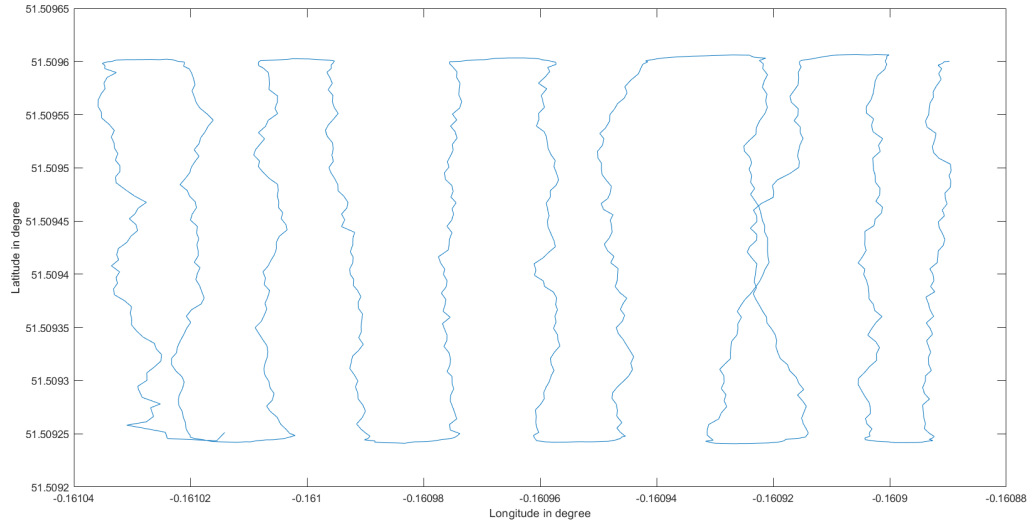

Figure 3: The position of GNSS with T = 4

The fig.4 shows the position of Integration, GNSS and Dead Reckoning, where x and y axes are longitude and latitude respectively. As what can be seen in fig.4, all kinds of algorithms start at the same point.The integration trajectory is similar to the GNSS trajectory. This result seems reasonable since the GNSS has much larger weighting than Dead Reckoning in extended Kalman Filter. It is also believable that the shape of integration is similar with the shape of DR. The ideal DR trajectory should approach to the GNSS firstly since DR has low short-term precision. However, the position error in dead reckoning grows with time, so the

error goes up with time. However, both of integration and GNSS turn left but DR moves forward directly, which is different with the expectation. Consequently, the measurements of GNSS or Dead Reckoning at the start may have wrong dataset. Comparing with the velocity, the measurements of GNSS at the start may have wrong dataset rather than Dead Reckoning which having undetectable outliers. The figure of integration velocity can be seen in fig.5, where

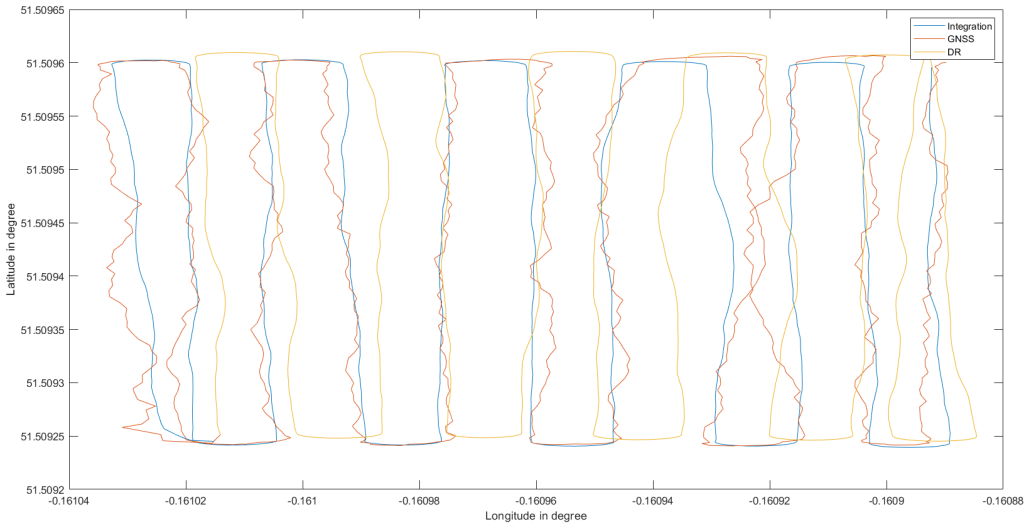

Figure 4: The position of Integration, GNSS and Dead Reckoning

the blue path is the velocity in the north direction and the red path is the velocity in the east direction. The north velocity increases rapidly in the beginning and keeps stable. It is because the robotic lawnmower moves to the latitude direction. When the north velocity approaches to zero, the east velocity is in the peak for a while since the lawnmower turns right. Moreover, both velocities go to zero in the end. In this case, the results seem to be reasonable.

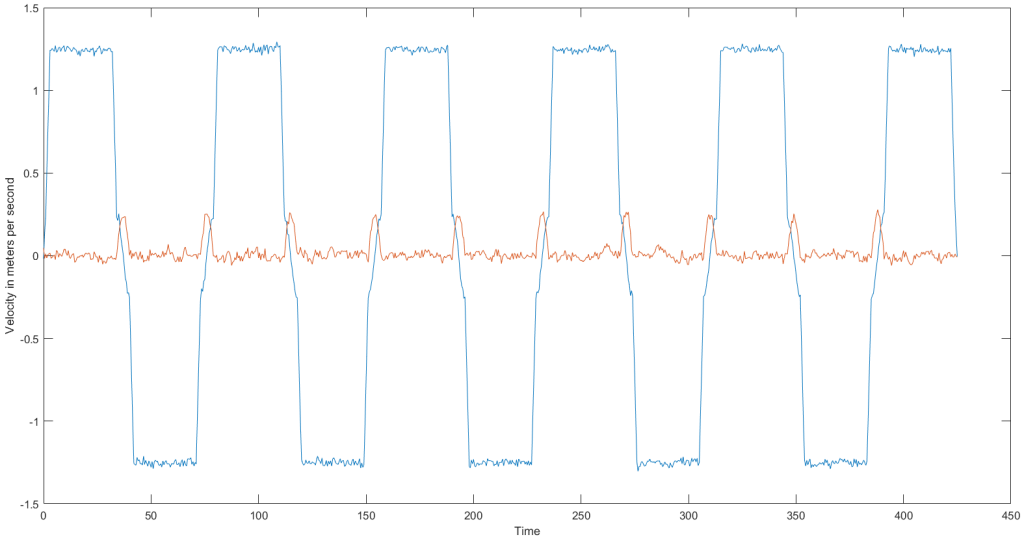

Figure 5: The velocity of Integration with d = 0.6

The figure shows the integration velocity without damping. Comparing with the figure of d = 0.6, the integration velocity with d = 0 has a large oscillatory error. Using a damping ratio,

this measurement error can be reduced.

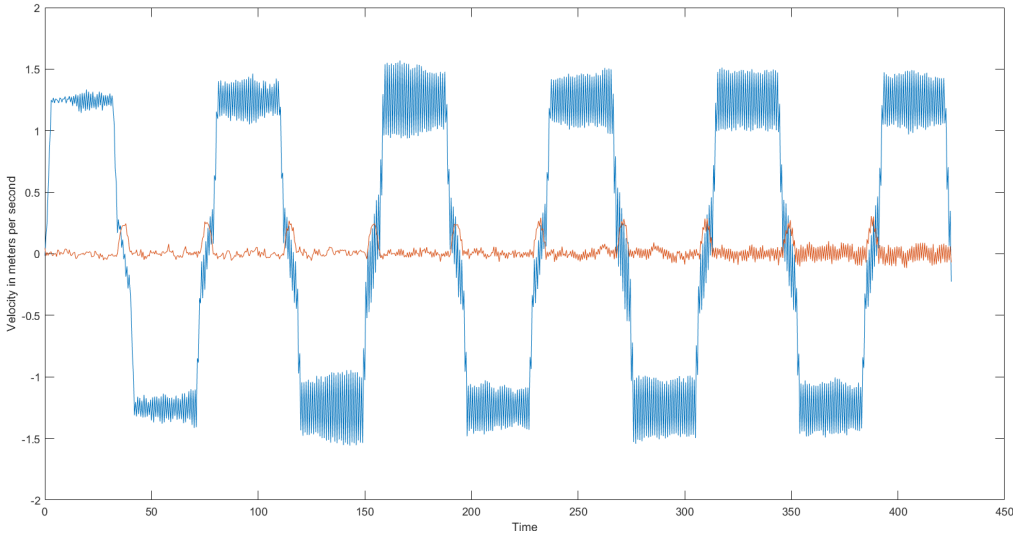

Figure 6: The velocity of Integration with d = 0

The heading value is similar with the path of velocity in the north direction. The peak angle is around 180° and the minimum angle is around 0°, which also causes by turning around. Consequently, the heading result seems to be reasonable.

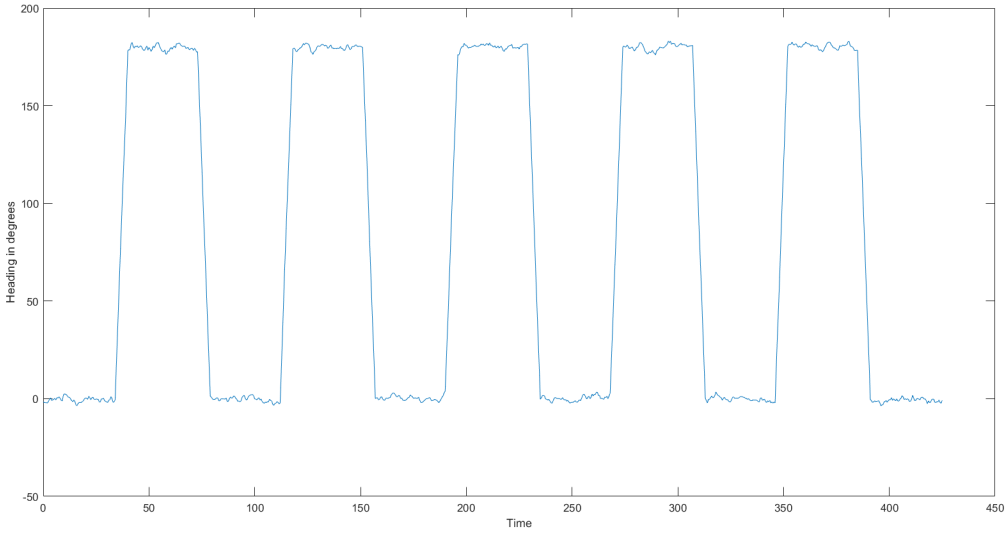

Figure 7: The velocity of Integration

Conclusion

The coursework requires to obtain the optimal position, velocity and heading of a robotic lawnmower. After correcting the DR measurement values by GNSS measurements based on Kalman Filtering, the optimal results are obtained. Most of the results are reasonable, however, the first few GNSS position may have some outliers which cannot be detected. In general, the result is satisfactory.

References

[1] P. Groves. (2019) Comp0130: Robot vision and navigation coursework 1: Integrated navigation for a robotic lawnmower. [Online]. Available: https://moodle-1819.ucl.ac.uk/pluginfile.php/368919/mod_resource/content/10/COMP0130Coursework1instructions2019.pdf

[2] P. D. Groves. (2019) Comp0130: Robot vision and navigation workshop1: Mobile gnss positioning using least-squares estimation. [Online]. Available: https://moodle-1819.ucl.ac.uk/pluginfile.php/368953/mod_resource/content/9/COMP0130Workshop1instructions.pdf

[3] P. Groves. (2019) Comp0130: Robot vision and navigation workshop 2: Aircraft navigation using gnss and kalman filtering. [Online]. Available: https://moodle-1819.ucl.ac.uk/pluginfile.php/368975/mod_resource/content/11/COMP0130Workshop2instructions.pdf

[4] P. D. Groves. (2019) Comp0130: Robot vision and navigation lecture 6: Multisensor integrated navigation. [Online]. Available: https://moodle-1819.ucl.ac.uk/pluginfile.php/368972/mod_resource/content/4/RVNLecture6(1perpage).pdf

[5] P. Groves. (2019) Comp0130: Robot vision and navigation workshop 3: Multisensor navigation. [Online]. Available: https://moodle-1819.ucl.ac.uk/pluginfile.php/368979/mod_resource/content/9/COMP0130Workshop3instructions.pdf

Appendices

A Matlab code Listings

Listing 1: The function **main.m**

```
clear all;
% obtain the initial value of position, velocity, offset and drift
x_init = Clock_offset_drift();
% use extended Kalman Filter to calculate GNSS position and velocity
[outlier_judge,position,velocity] = GNSS_extended_Kalman_filter(x_init);
% Gyro-Magnetometer corrected
gyro_heading_deg = Gyro_heading();
% calculate Dead Reckoning position and velocity
[position_DR,velocity_DR] = Dead_Reckoning(position,gyro_heading_deg);
% DR/GNSS integration
results = Itegration(position,velocity,position_DR,velocity_DR,gyro_heading_deg);
% create motion profile file
csvwrite('Output_Profile.csv',results);
```

Listing 2: The function **Clock_offset_drift.m**

```
function [x_init] = Clock_offset_drift()

% define parameters
deg_to_rad = 0.01745329252;
rad_to_deg = 1/deg_to_rad; % Radians to degrees conversion factor
c = 299792458; % Speed of light in m/s
omega_ie = 7.292115E-5;  % Earth rotation rate in rad/s

% convert the position to Cartesian ECEF position
latitude = 0;
longitude = 0;
height = 0;
[r_eb_e,v_eb_e] = pv_NED_to_ECEF(latitude*deg_to_rad,longitude*deg_to_rad,height,[0; 0; 0]);

% initialize values
Sat_r_es_e = zeros(8,3);
Sat_v_es_e = zeros(8,3);
position = zeros(1,3);
velocity = zeros(1,3);
clock_offset = 100000;
der_clock_offset = 200;
raj = zeros(8,1);
der_raj = zeros(8,1);

csv_Pseudo_ranges = csvread('Pseudo_ranges.csv');
csv_Pseudo_range_rates = csvread('Pseudo_range_rates.csv');
for epoch = 1:1

    % compute the Cartesian ECEF positions of the satellites
    j = csv_Pseudo_ranges(1,2:size(csv_Pseudo_ranges,2));% j:satellite numbers
    for i = 1:size(j,2)
        % sat_r_es_e:ECEF satellite position, sat_v_es_e:ECEF satellite velocity
        [sat_r_es_e,sat_v_es_e] = Satellite_position_and_velocity(0.5*(epoch-1),j(i));
        Sat_r_es_e(i,:) = sat_r_es_e;%%r_ej
        Sat_v_es_e(i,:) = sat_v_es_e;%%v_ej
    end

    %predict the ranges
    for int = 1:20
        for i = 1:size(j,2)
            raj_tem = 0;
            for iteration = 1:6
                C = [1,omega_ie*raj_tem/c,0;-omega_ie*raj_tem/c,1,0;0,0,1];
                raj_tem = sqrt((C*Sat_r_es_e(i,:)' - r_eb_e)'*(C*Sat_r_es_e(i,:)' - r_eb_e));
```

```
45
46              end
47              raj(i) = raj_tem;
48          end
49
50          % compute the line-of-sight unit vector
51          u = zeros(8,3);
52          for i = 1:size(j,2)
53              C = [1,omega_ie*raj(i)/c,0;-omega_ie*raj(i)/c,1,0;0,0,1];
54              u(i,:) = (C*Sat_r_es_e(i,:)' - r_eb_e)/raj(i);
55          end
56
57          % predict the range rates
58          omega = [0 -omega_ie 0;omega_ie 0 0;0 0 0];
59          for i = 1:size(j,2)
60              der_raj_tem = 0;
61              C = [1,omega_ie*raj(i)/c,0;-omega_ie*raj(i)/c,1,0;0,0,1];
62              for iteration = 1:6
63                  der_raj_tem = u(i,:)*(C*(Sat_v_es_e(i,:)' + omega*Sat_r_es_e(i,:)')-...
64                  (v_eb_e+omega*r_eb_e));
65              end
66              der_raj(i) = der_raj_tem;
67          end
68
69          %formulate the predicted state vector
70          last_col = [1;1;1;1;1;1;1;1];
71          x_old = [r_eb_e;clock_offset];
72          rho = csv_Pseudo_ranges(epoch+1,2:size(csv_Pseudo_ranges,2))';
73          delta_z = rho-raj-clock_offset;
74          H_G = [-u last_col];
75          x_new = x_old + inv(H_G'*H_G)*H_G'*delta_z;
76
77          %update r_ea+ and clock offset
78          r_eb_e = x_new(1:3);
79          clock_offset = x_new(4);
80
81          der_x_old = [v_eb_e;der_clock_offset];
82          der_rho = csv_Pseudo_range_rates(epoch+1,2:size(csv_Pseudo_ranges,2))';
83          der_delta_z = der_rho-der_raj-der_clock_offset;
84          H_G = [-u last_col];
85          der_x_new = der_x_old + inv(H_G'*H_G)*H_G'*der_delta_z;
86          %update r_ea+ and clock offset
87          v_eb_e = der_x_new(1:3);
88          der_clock_offset = der_x_new(4);
89      end
90
91      % outlier detection
92      v = (H_G/(H_G'*H_G)*H_G'-eye(8))*delta_z;
93      sigma_rho = 5;
94      C_v = (eye(8) - H_G/(H_G'*H_G)*H_G')*sigma_rho^2;
95      outlier_judge = zeros(8,1);
96      Threshold = 6;
97      for number = 1:8
98          tem = abs(v(number))- sqrt(C_v(number,number))*Threshold;
99          if tem>0
100         outlier_judge(number) = tem;
101         end
102     end
103
104     % convert the Cartesian position and velocity solution
105     [L_b,lambda_b,h_b,v_eb_n] = pv_ECEF_to_NED(r_eb_e,v_eb_e);
106     latitude = L_b * rad_to_deg;
107     longitude = lambda_b * rad_to_deg;
108     position(epoch,:) = [latitude,longitude,h_b];
109     velocity(epoch,:) = v_eb_n;
110     x_init = [r_eb_e',v_eb_e',clock_offset,der_clock_offset]';
111 end
```

Listing 3: The function **GNSS_extended_Kalman_filter.m**

```
function [outlier_judge,position,velocity] = GNSS_extended_Kalman_filter(x_init)
% parameters
deg_to_rad = 0.01745329252;
rad_to_deg = 1/deg_to_rad; % Radians to degrees conversion factor
omega_ie = 7.292115E-5;  % Earth rotation rate in rad/s
c = 299792458; % Speed of light in m/s

% initialize x and P
x_est = x_init;
P_matrix = [eye(3)*10^2,zeros(3,5);
            zeros(3,3),eye(3)*0.1^2,zeros(3,2);
            zeros(1,6),10^2,0;
            zeros(1,7),0.1^2];

% compute the transition matrix
tau_s = 0.5;
I3 = eye(3);
zero3_1 = zeros(3,1);
zero1_3 = zeros(1,3);
phi = [I3,tau_s*I3,zero3_1,zero3_1;
       zeros(3,3),I3,zero3_1,zero3_1;
       zero1_3,zero1_3,1,tau_s;
       zero1_3,zero1_3,0,1];

% compute the system noise convariance matrix
S_a = 5;
S_c_phi = 0.01;
S_c_f = 0.04;
Q = [S_a*tau_s^3*I3/3,S_a*tau_s^2*I3/2,zero3_1,zero3_1;
     S_a*tau_s^2*I3/2,S_a*tau_s*I3,zero3_1,zero3_1;
     zero1_3,zero1_3,S_c_phi*tau_s+S_c_f*tau_s^3/3,S_c_f*tau_s^2/2;
     zero1_3,zero1_3,S_c_f*tau_s^2/2,S_c_f*tau_s];

position = zeros(851,3);
velocity = zeros(851,3);

outlier_judge = zeros(851,8);
for epoch = 1:851
    % use the transition matrix to prograte the state estimates
    x_minus = phi*x_est;

    % prograte the state estimates
    P_minus = phi*P_matrix*phi' + Q;

    % predict the ranges
    % calculate r_ej
    Sat_r_es_e = zeros(8,3);
    Sat_v_es_e = zeros(8,3);
    csv_Pseudo_ranges = csvread('Pseudo_ranges.csv');
    j = csv_Pseudo_ranges(1,2:size(csv_Pseudo_ranges,2));% j:satellite numbers
    % sat_r_es_e:ECEF satellite position, sat_v_es_e:ECEF satellite velocity
    for i = 1:size(j,2)
        [sat_r_es_e,sat_v_es_e] = Satellite_position_and_velocity(0.5*(epoch-1),j(i));
        Sat_r_es_e(i,:) = sat_r_es_e;%%r_ej
        Sat_v_es_e(i,:) = sat_v_es_e;%%v_ej
    end

    raj = zeros(8,1);
    for i = 1:8
        raj_tem = 0;
        for iteration = 1:10
            C = [1,omega_ie*raj_tem/c,0;-omega_ie*raj_tem/c,1,0;0,0,1];
            raj_tem = sqrt((C*Sat_r_es_e(i,:)' - x_minus(1:3))'*(C*Sat_r_es_e(i,:)' - x_minus(1:3)));
        end
        raj(i) = raj_tem;
    end

    % compute line-of-sight
    u = zeros(8,3);
```

```
    for i = 1:8
        C = [1,omega_ie*raj(i)/c,0;-omega_ie*raj(i)/c,1,0;0,0,1];
        u(i,:) = (C*Sat_r_es_e(i,:)' - x_minus(1:3))/raj(i);
    end

    % predict range rates
    der_raj = zeros(8,1);
    omega = [0 -omega_ie 0;omega_ie 0 0;0 0 0];
    for i = 1:8
        der_raj_tem = 0;
        C = [1,omega_ie*raj(i)/c,0;-omega_ie*raj(i)/c,1,0;0,0,1];
        for iteration = 1:10
            der_raj_tem = u(i,:)*(C*(Sat_v_es_e(i,:)' + omega*Sat_r_es_e(i,:)')-...
            (x_minus(4:6)+omega*x_minus(1:3)));
        end
        der_raj(i) = der_raj_tem;
    end

    % compute the measurement matrix
    H_k = [-u,zeros(8,3),ones(8,1),zeros(8,1);
           zeros(8,3),-u,zeros(8,1),ones(8,1)];

    % compute the measurement noise convariance matrix
    sigma_rho = 10;
    sigma_r = 0.05;
    sigma_rho_matrix = sigma_rho^2*eye(8);
    sigma_r_matrix = sigma_r^2*eye(8);
    R_k = [sigma_rho_matrix,zeros(8,8);zeros(8,8),sigma_r_matrix];

    % compute the Kalman gain matrix
    K_k = P_minus*H_k'/(H_k*P_minus*H_k'+R_k);

    % formulate the measurement innovation vector
    % load measured pseudo-range
    csv_Pseudo_ranges = csvread('Pseudo_ranges.csv');
    rho = csv_Pseudo_ranges(epoch+1,2:size(csv_Pseudo_ranges,2))';
    % load measured pseudo-range rate
    csv_Pseudo_range_rates = csvread('Pseudo_range_rates.csv');
    der_rho = csv_Pseudo_range_rates(epoch+1,2:size(csv_Pseudo_range_rates,2))';
    delta_z = [rho-raj-x_minus(7)*ones(8,1);
               der_rho-der_raj-x_minus(8)*ones(8,1)];

    % detect outliers
    H_G = [-u,ones(8,1)];
    v = (H_G/(H_G'*H_G)*H_G'-eye(8))*delta_z(1:8);
    sigma_rho = 10;
    C_v = (eye(8) - H_G/(H_G'*H_G)*H_G')*sigma_rho^2;
    Threshold = 4;
    for number = 1:8
        tem = abs(v(number))- sqrt(C_v(number,number))*Threshold;
        if tem>0
        outlier_judge(epoch,number) = tem';
        end
    end
    if sum(outlier_judge(epoch,:) == 0) ~= 8
        [~,satellite] = max(outlier_judge(epoch,:));
%         satellite = 3;
        H_k(satellite+8,:)=[];
        H_k(satellite,:)=[];
        sigma_rho_matrix = sigma_rho^2*eye(7);
        sigma_r_matrix = sigma_r^2*eye(7);
        R_k = [sigma_rho_matrix,zeros(7,7);zeros(7,7),sigma_r_matrix];
        K_k = P_minus*H_k'/(H_k*P_minus*H_k'+R_k);
        delta_z(satellite+8,:)=[];
        delta_z(satellite,:)=[];
    end


    % update the state estimates
    x_pulse =  x_minus + K_k * delta_z;
```

```
    r_eb_e = x_pulse(1:3);
    v_eb_e = x_pulse(4:6);
    x_est = x_pulse;


    % error convariance matrix
    P_pulse = (eye(8) - K_k*H_k)*P_minus;
    P_matrix = P_pulse;

    % convert from ECEF to NED
    [L_b,lambda_b,h_b,v_eb_n] = pv_ECEF_to_NED(r_eb_e,v_eb_e);
    latitude = L_b * rad_to_deg;
    longitude = lambda_b * rad_to_deg;
    position(epoch,:) = [latitude,longitude,h_b];
    velocity(epoch,:) = v_eb_n;

end
```

Listing 4: The function **Gyro_heading.m**

```
function gyro_heading_deg = Gyro_heading()
% parameters
deg_to_rad = 0.01745329252;
rad_to_deg = 1/deg_to_rad; % Radians to degrees conversion factor

Dead_reckoning = csvread('Dead_reckoning.csv');
gyro_angular_rate = Dead_reckoning(:,6);
magnetic_heading_deg = Dead_reckoning(:,7);
magnetic_heading_rad = magnetic_heading_deg*deg_to_rad;

% calculate the gyroscope headling
gyro_heading_deg = zeros(851,1);
gyro_heading_deg(1) = magnetic_heading_deg(1);
for i = 2:851
    gyro_heading_deg(i) = gyro_heading_deg(i-1) + (gyro_angular_rate(i)*0.5*rad_to_deg);
end
gyro_heading_rad = gyro_heading_deg * deg_to_rad;

% update heading error
x_pre = [0;0];
phi_k_minus_1 = [1,0.5;0,1];
P_k_minus_1 = [0.01^2,0;0,(1*deg_to_rad)^2];
%P_k_minus_1 = [1,0;1,0];
S_rg = 3*10^-6;
S_bgd = (1*deg_to_rad)^2;
%S_bgd = 1*10^-12;
H_k = [-1,0];

% calculate heading in each epoch
for t = 2:851
    Q_k_minus_1 = [S_rg*0.5+S_bgd*0.5^3/3,S_bgd*0.5^2/2;S_bgd*0.5^2/2,S_bgd*0.5];
    x_k_minus = phi_k_minus_1 * x_pre;
    P_k_minus = phi_k_minus_1 * P_k_minus_1 * phi_k_minus_1'+ Q_k_minus_1;
    delta_z_k_minus = (magnetic_heading_rad(t) - gyro_heading_rad(t)) - H_k*x_k_minus;
    R_k = (4*deg_to_rad)^2;
    K_k = P_k_minus*H_k'/(H_k * P_k_minus * H_k' + R_k);
    % update x
    x_pre = x_k_minus + K_k * delta_z_k_minus;
    % update P
    P_k_minus_1 = (eye(2) - K_k*H_k) * P_k_minus;
    gyro_heading_deg(t) = (gyro_heading_rad(t) - x_pre(1))*rad_to_deg;
end
```

Listing 5: The function **Dead_Reckoning.m**

```
function [position_DR,velocity_DR] = Dead_Reckoning(position,gyro_heading_deg)
% parameters
deg_to_rad = 0.01745329252; % Degrees to radians conversion factor
rad_to_deg = 1/deg_to_rad; % Radians to degrees conversion factor
```

```
5
6  Dead_reckoning = csvread('Dead_reckoning.csv');
7  speed_average = (Dead_reckoning(:,2)+Dead_reckoning(:,3)+Dead_reckoning(:,4)+Dead_reckoning(:,5))/4;
8  heading_deg = gyro_heading_deg;
9  heading_rad = heading_deg * deg_to_rad;
10
11 times = size(gyro_heading_deg,1); %calculate the number of terms
12
13 %calculate average velocity
14 v_N_average = zeros(times,1);
15 v_E_average = zeros(times,1);
16 for i = 2:times
17     v_N_average(i) = (cos(heading_rad(i)) + cos(heading_rad(i-1)))*speed_average(i)/2;
18     v_E_average(i) = (sin(heading_rad(i)) + sin(heading_rad(i-1)))*speed_average(i)/2;
19 end
20
21 %calculate latitude and longitude
22 latitude_rad = zeros(times,1);
23 longitude_rad = zeros(times,1);
24 latitude_rad(1) = position(1,1)*deg_to_rad;
25 longitude_rad(1) = position(1,2)*deg_to_rad;
26
27 for i = 2:times
28     [R_N,R_E]= Radii_of_curvature(latitude_rad(i-1));
29     latitude_rad(i) = latitude_rad(i-1) + v_N_average(i)*0.5/(R_N+position(i-1,3));
30     longitude_rad(i) = longitude_rad(i-1) + v_E_average(i)*0.5/((R_E+position(i-1,3))*...
31     cos(latitude_rad(i)));
32 end
33
34 latitude_deg = latitude_rad * rad_to_deg;
35 longitude_deg = longitude_rad * rad_to_deg;
36
37 % calculate the instantaneous DR velocity
38 v_N_instant = zeros(times,1);
39 v_E_instant = zeros(times,1);
40 v_N_instant(1) = speed_average(1)*cos(heading_rad(1));
41 v_E_instant(1) = speed_average(1)*sin(heading_rad(1));
42
43 d = 0.6;
44 for i = 2:times
45     v_N_instant(i) = (2-d)*v_N_average(i) - (1-d)*v_N_instant(i-1);
46     v_E_instant(i) = (2-d)*v_E_average(i) - (1-d)*v_E_instant(i-1);
47 end
48
49 position_DR = [latitude_deg,longitude_deg];
50 velocity_DR = [v_N_instant,v_E_instant];
```

Listing 6: The function **Itegration.m**

```
1  function results = Itegration(position,velocity,position_DR,velocity_DR,gyro_heading_deg)
2  %parameters
3  deg_to_rad = 0.01745329252; % Degrees to radians conversion factor
4  rad_to_deg = 1/deg_to_rad; % Radians to degrees conversion factor
5
6  latitude_deg = position(:,1);
7  longitude_deg = position(:,2);
8  height = position(:,3);
9  v_N = velocity(:,1);
10 v_E = velocity(:,2);
11 latitude_rad = latitude_deg*deg_to_rad;
12 longitude_rad = longitude_deg*deg_to_rad;
13
14 DR_latitude_deg = position_DR(:,1);
15 DR_longitude_deg = position_DR(:,2);
16 DR_latitude_rad = DR_latitude_deg*deg_to_rad;
17 DR_longitude_rad = DR_longitude_deg*deg_to_rad;
18 DR_v_N_instant = velocity_DR(:,1);
19 DR_v_E_instant = velocity_DR(:,2);
20
21 %initialize P_0
```

```
x_pre = zeros(4,1);
%initialize P_0
sigma_v = 0.05;
sigma_r = 10;

times = size(position,1); %calculate the number of terms

[R_N,R_E]= Radii_of_curvature(latitude_rad(1));
P_pre = [sigma_v^2*eye(2),zeros(2,2);
         zeros(1,2),sigma_r^2/(R_N+height(1))^2,0;
         zeros(1,3),sigma_r^2/((R_E+height(1))^2*cos(latitude_rad(1))^2)];

results = zeros(351,6);
for i = 1:times-1

    %1.compute the transition matrix
    [R_N,R_E]= Radii_of_curvature(latitude_rad(i));
    tau = 0.5;
    phi_k_min_one = eye(4);
    phi_k_min_one(3,1) = tau/(R_N + height(i));
    phi_k_min_one(4,2) = tau/((R_E + height(i))*cos(latitude_rad(i)));

    %2.compute the system noise convariance matrix
    S_DR = 0.01;
    Q_k_min_one = zeros(4,4);
    Q_k_min_one(1,1) = S_DR*tau;
    Q_k_min_one(2,2) = Q_k_min_one(1,1);
    Q_k_min_one(3,3) = S_DR*tau^3/(R_N+height(i))^2/3;
    Q_k_min_one(4,4) = S_DR*tau^3/((R_E+height(i))^2*cos(latitude_rad(i))^2)/3;
    Q_k_min_one(1,3) = S_DR*tau^2/(R_N+height(i))/2;
    Q_k_min_one(3,1) = Q_k_min_one(1,3);
    Q_k_min_one(2,4) = S_DR*tau^2/((R_E+height(i))*cos(latitude_rad(i)))/2;
    Q_k_min_one(4,2) = Q_k_min_one(2,4);

    %3.compute state estimate
    x_k_minus = phi_k_min_one*x_pre;

    %4.compute the error convariance matrix
    P_k_minus = phi_k_min_one*P_pre*phi_k_min_one' + Q_k_min_one;

    %5.measurement matrix
    H_k = [zeros(2,2),-eye(2);-eye(2),zeros(2,2)];

    %6.measurement noise convariance matrix
    sigma_Gr = 5.2;
    sigma_Gv = 0.02;
    R_k = zeros(4,4);
    [R_N,R_E]= Radii_of_curvature(latitude_rad(i+1)); %new R_N and R_E
    R_k(1,1) = sigma_Gr^2/(R_N + height(i+1))^2;
    R_k(2,2) = sigma_Gr^2/((R_E + height(i+1))^2*cos(latitude_rad(i+1))^2);
    R_k(3,3) = sigma_Gv^2;
    R_k(4,4) = sigma_Gv^2;

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

    %8.measurment innovation vector
    delta_z_k_minus = [latitude_rad(i+1)-DR_latitude_rad(i+1);
                       longitude_rad(i+1)-DR_longitude_rad(i+1);
                       v_N(i+1) - DR_v_N_instant(i+1);
                       v_E(i+1) - DR_v_E_instant(i+1)] - H_k*x_k_minus;

    %9.update the state estimates
    x_k_plus = x_k_minus + K_k*delta_z_k_minus;
    x_pre = x_k_plus;

    %10.update the error convariance matrix
    P_k_plus = (eye(4) - K_k*H_k)*P_k_minus;
    P_pre = P_k_plus;

```

```
92      %11. results
93      v_N_corrected = DR_v_N_instant(i+1) - x_k_plus(1);
94      v_E_corrected = DR_v_E_instant(i+1) - x_k_plus(2);
95
96      latitude_corrected_rad = DR_latitude_rad(i+1) -  x_k_plus(3);
97      longitude_corrected_rad = DR_longitude_rad(i+1) -  x_k_plus(4);
98      latitude_corrected_deg = latitude_corrected_rad * rad_to_deg;
99      longitude_corrected_deg = longitude_corrected_rad * rad_to_deg;
100
101     results(i+1,1) = 0.5*i;
102     results(i+1,2) = latitude_corrected_deg;
103     results(i+1,3) = longitude_corrected_deg;
104     results(i+1,4) = v_N_corrected;
105     results(i+1,5) = v_E_corrected;
106 end
107
108 results(1,1) = 0;
109 results(1,2) = latitude_deg(1);
110 results(1,3) = longitude_deg(1);
111 results(1,4) = v_N(1);
112 results(1,5) = v_E(1);
113 results(:,6) = gyro_heading_deg;
114
115 %plot position figure
116 plot(results(:,3),results(:,2));hold on;
117 plot(longitude_deg,latitude_deg);hold on;
118 plot(DR_longitude_deg,DR_latitude_deg);hold on;
119 xlabel('Longitude in degree');
120 ylabel('Latitude in degree');
121 legend('Integration','GNSS','DR')
122 figure;
123
124 %plot velocity figure
125 plot(results(:,1),results(:,4));hold on;
126 plot(results(:,1),results(:,5));hold on;
127 xlabel('Time');
128 ylabel('Velocity in meters per second');
129 legend('North velocity','East velocity')
130 figure;
131
132 %plot heading
133 plot(results(:,1),results(:,6));hold on;
134 xlabel('Time');
135 ylabel('Heading in degrees');
```