COMP0130: ROBOT VISION AND NAVIGATION
Coursework 1: Integrated Navigation for a Robotic Lawnmower
Group K

Hyungjoo Kim
170103694
ucabhki@ucl.ac.uk

Ahmed Salem
20208009
zcemasa@ucl.ac.uk

Zhonghao Wang
20061720
ucabzw6@ucl.ac.uk

1. Introduction

Real-time positioning of a moving body is essential when navigating from one location to another whilst avoiding collisions and obstacles [1]. Typical applications include intelligent transport systems, and robot navigation where navigation solutions are required immediately. A navigation solution refers to the output of a navigation system and is generally comprised of time, geodetic latitude and longitude, velocity and heading data. Navigation generally includes various systems and sensors such as Global Navigation Satellite System (GNSS) receivers, wheel speed sensors, magnetic compasses, and gyroscopes to estimate a solution [1]. This is due to the fact that no single navigation technology is robust enough to provide high accuracy and reliability when subjected to a range of different and/or challenging environments [2].

GNSS is a type of signal-based positioning mechanism that uses three segments to function, a user-based receiver, known control points and multiple space satellites that are formed in a GNSS constellation [1]. In a particular position, the receiver is designed to measure the signal arrival time from at least 4 satellites and use the signal modulation from the satellite to derive a transmission time and generate a pseudo-range as shown in equation 1 (lecture 1-D from source [3]).

$$\rho = c(t_{sa} - t_{st}) \tag{1}$$

ρ is the raw pseudo-range measurement, c is the speed of light, t_{sa} is the signal arrival time from satellite to receiver, and t_{st} is the transmission time which is derived by the receiver [3]. This positioning occurrence in three spatial dimensions is achieved by passive ranging since each satellite signal includes parameters in the form of ranging codes and navigation data messages which are used to compute position known as the *ephemeris* [1]. The receiver uses three satellites pseudo-range measurements and triangulation to derive a position that is not corrected by a clock offset, which needs to be corrected for since a navigation solution must be in four dimensions with the last dimension being time [3]. A fourth satellite pseudo-range measurement is needed to resolve this error where a corrected pseudo-range is expressed as shown in equation 2 (equation 8.4 from source [1]).

$$\rho_C = \sqrt{(r_{is}(t_{st}) - r_{ia}(t_{sa}))^T (r_{is}(t_{st}) - r_{ia}(t_{sa}))} + \delta\rho_c(t_{sa}) \tag{2}$$

ρ_C is the pseudo-range measurement corrected for satellite clock error, $r_{is}(t_{st})$is the satellite position at the time of signal transmission, $r_{ia}(t_{sa})$ is the receiver (antenna) position at the time of signal arrival, and $\delta\rho_c(t_{sa})$ is the receiver clock offset at the time of signal arrival [1]. GNSS can provide a basic position that is accurate to a few meters however it can also be subject to interference and signal interruptions which lead to larger inaccuracies that must be corrected for [1].

Dead reckoning is another type of positioning mechanism which either measures the change in position or the velocity and integrates it in a series of time-steps following a known initial position [1]. Heading is measured using a magnetic compass and to account for changes in altitude, roll and pitch components can be determined using a gyroscope [1]. When combining these sensors to compute position, velocity, and speed, this is described as an inertial navigation system [1].

The aim of this paper is to fuse the positioning sub-systems, GNSS and dead reckoning, using an integration algorithm to compute an accurate navigation solution using supplied simulated data from a robotic lawnmower located in London. The lawnmower is equipped with a GNSS receiver, wheel speed sensors, magnetic compass, and a low-cost gyroscope. Approaches to correct for the errors from the sub-systems are described and the final trajectory solution is shown.

2. Algorithm Description and Methods

2.1. Initial GNSS Solution with Least Squares Estimation

In practical application, a number of sources of error from GNSS must be accounted for sourcing from the ephemeris broadcast, satellite clock errors, and refraction in the ionosphere and troposphere. The data that has been provided has been described as corrected for raw errors, however there are a variety of residual errors that are still present. This section describes the type of errors encountered and their correction strategies. For example, algorithm 1 corrected the position measurements in the position solution for the Sagnac effect by using a Sagnac effect compensation matrix (C_e^I) shown in line 9 of algorithm 1. Sagnac effect is when there is a change in the apparent speed of light due to the coordinate frame rotation (in this case the spin of the earth) with respect to inertial space [3]. It was essential to correct the measurements prior to the calculations presented below.
A navigation processor may not be able to use equation 2 to estimate position analytically, particularly when the aim is to improve location accuracy by using more than four satellites to determine position [1]. In this situation, there are 5 or more measurements, and only 4 unknowns shown in equation 2. Therefore, the solution is considered overdetermined and least-squares estimation must be used. By performing a Taylor series expansion about the receiver position (r_{ea}^e) and the clock offset ($\delta\rho_c^a$), the equation is linearised such that least squares estimation can be used to initialise a Kalman filter with an accurate position (more details in section 2.2) [1]. Equation 2 can be replaced by equation 3 shown below (equation 9.130 from [1]),

$$\begin{pmatrix} \rho_C^1 - \rho_C^{1-} \\ \rho_C^2 - \rho_C^{2-} \\ \vdots \\ \rho_C^m - \rho_C^{m-} \end{pmatrix} = \mathrm{H}_G \begin{pmatrix} \hat{x}_{ia}^{i+} - \hat{x}_{ia}^{i-} \\ \hat{y}_{ia}^{i+} - \hat{y}_{ia}^{i-} \\ \hat{z}_{ia}^{i+} - \hat{z}_{ia}^{i-} \\ \delta\rho_c^{a+} - \delta\rho_c^{a-} \end{pmatrix} + \begin{pmatrix} \delta\rho_C^{1+} \\ \delta\rho_C^{2+} \\ \vdots \\ \delta\rho_C^{m+} \end{pmatrix} \tag{3}$$

where H_G is a geometry matrix representing the partial differentiation of the pseudo-range measurement with respect to the x, y, z locations of the user position and the clock offset that yields the line-of-sight unit vectors ($\hat{u}_{a1,x}^i$) from the receiver antenna to the satellites as shown in equation 4 (equation 9.133 from [1]). The notation (*i*) denotes a point in time, and (m) denotes the number of satellites [1].

$$\mathrm{H}_G = \begin{pmatrix} -\hat{u}_{a1,x}^{i-} & -\hat{u}_{a1,y}^{i-} & -\hat{u}_{a1,z}^{i-} & 1 \\ -\hat{u}_{a2,x}^{i-} & -\hat{u}_{a2,y}^{i-} & -\hat{u}_{a2,z}^{i-} & 1 \\ \vdots & \vdots & \vdots & \vdots \\ -\hat{u}_{am,x}^{i-} & -\hat{u}_{am,y}^{i-} & -\hat{u}_{am,z}^{i-} & 1 \end{pmatrix} \tag{4}$$

Finally, when iterating and using more than four pseudo-range measurements equation 3 can be transformed to solve for the position and clock-offset solution as shown in equation 5.

$$\begin{pmatrix} \hat{x}_{ia}^{i+} \\ \hat{y}_{ia}^{i+} \\ \hat{z}_{ia}^{i+} \\ \delta\rho_c^{a+} \end{pmatrix} = \begin{pmatrix} \hat{x}_{ia}^{i-} \\ \hat{y}_{ia}^{i-} \\ \hat{z}_{ia}^{i-} \\ \delta\rho_c^{a-} \end{pmatrix} + \left(\mathrm{H}_G{}^T \mathrm{H}_G\right)^{-1} \mathrm{H}_G{}^T \begin{pmatrix} \rho_C^1 - \rho_C^{1-} \\ \rho_C^2 - \rho_C^{2-} \\ \vdots \\ \rho_C^m - \rho_C^{m-} \end{pmatrix} \tag{5}$$

Based on the above concepts, the GNSS position and receiver clock of the lawnmower was computed to initialise the parameters by using the least squares estimation as shown the below algorithm. The velocity of the lawnmower is also computed using least-squares and is shown in pseudo-code format below. Algorithm 1 followed the step-by-step instructions from [4].

Algorithm 1 Initial GNSS Solution with Least Square Estimation (Single epoch) [4]

```
Input: pseudo ranges $\boldsymbol{r}$, pseudo range rates $\dot{\boldsymbol{r}}$, satellite position $\boldsymbol{p_s}$, satellite velocity $\boldsymbol{v_s}$,
       clock offset $\boldsymbol{\delta p_0}$, clock drift $\boldsymbol{\delta v_0}$
1:  $p_0 \leftarrow (0, 0, 0)$                                  {$p_s$: (Latitude, Longitude, Height}
2:  $v_0 \leftarrow (0, 0, 0)$                                  {$v_s$: (North, East, Down)}
3:  $(\delta\rho_0, \delta\dot{\rho}_0) \leftarrow (0, 0)$      {$\delta\rho_0, \delta\dot{\rho}_0$: (clock offset, clock drift)}
4:  Computes the positions and velocities for each satellite
5:  Initialise a state vector estimation $x_0 = (p_0, v_0, \delta p_0, \delta v_0)^T$
6:  while true do
7:      for the number of satellite k = 1...8 do
8:          Set the initial range ($\hat{r}_{ak}^- = r_1$) and range rate ($\hat{\dot{r}}_{ak}^- = \dot{r}_1$) from the user position to each satellite
9:          $C_e^I \leftarrow \begin{pmatrix} 1 & \omega_{ie} r_{ak}/c & 0 \\ -\omega_{ie} r_{ak}/c & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$      {$\omega_{ie}$: Earth rotation rate, $c$: the speed of light}
10:         Computes the predict the ranges and range rates from user position to each satellite
11:         $\hat{r}_{ak}^- \leftarrow \sqrt{[C_e^I(\hat{r}_{ak}^-)\hat{r}_{ek}^e - \hat{r}_{ea}^e]^T [C_e^I(\hat{r}_{ak}^-)\hat{r}_{ek}^e - \hat{r}_{ea}^e]}$  {$\hat{r}_{ek}^e$: satellite position, $\hat{r}_{ea}^e$: user position}
12:         $\hat{\dot{r}}_{ak}^- \leftarrow \hat{u}_{ak}^{e-T}[C_e^I(\hat{r}_{ak}^-)(\hat{v}_{ek}^e + \Omega_{ie}^e \hat{r}_{ek}^e) - (\hat{v}_{ek}^{e-} + \Omega_{ie}^e \hat{r}_{ek}^{e-})]$      {$\Omega_{ie}^e$: skew symmetric matrix}
13:         $u_{ak}^e \leftarrow (\frac{C_e^I(\hat{r}_{ak}^-)\hat{r}_{ek}^e - \hat{r}_{ea}^e}{\hat{r}_{ak}^-})$      {$u_{ak}^e$: line-of-sight from the user position to each satellite}
14:     end for
15:     $(\delta\hat{\rho}_c^{a-}, \delta\hat{\dot{\rho}}_c^{a-}) \leftarrow (100000, 200)$  {$\delta\hat{\rho}_c^{a-}, \delta\hat{\dot{\rho}}_c^{a-}$: Initial clock offset and drift standard deviation}
16:     $\hat{p}^- \leftarrow (\begin{smallmatrix} \hat{r}_{ea}^{e-} \\ \delta\hat{\rho}_c^{a-} \end{smallmatrix})$, $\hat{v}^- \leftarrow (\begin{smallmatrix} \hat{v}_{ea}^{e-} \\ \delta\hat{\dot{\rho}}_c^{a-} \end{smallmatrix})$      {$\hat{p}^-, \hat{v}^-$: predicted state vector for position and velocity}
17:     $\delta z_p^- \leftarrow \begin{pmatrix} \tilde{\rho}_a^1 - \hat{r}_{a1}^- - \delta\hat{\rho}_c^{a-} \\ \tilde{\rho}_a^2 - \hat{r}_{a2}^- - \delta\hat{\rho}_c^{a-} \\ \tilde{\rho}_a^3 - \hat{r}_{a3}^- - \delta\hat{\rho}_c^{a-} \\ \tilde{\rho}_a^4 - \hat{r}_{a4}^- - \delta\hat{\rho}_c^{a-} \\ \tilde{\rho}_a^5 - \hat{r}_{a5}^- - \delta\hat{\rho}_c^{a-} \\ \tilde{\rho}_a^6 - \hat{r}_{a6}^- - \delta\hat{\rho}_c^{a-} \\ \tilde{\rho}_a^7 - \hat{r}_{a7}^- - \delta\hat{\rho}_c^{a-} \\ \tilde{\rho}_a^8 - \hat{r}_{a8}^- - \delta\hat{\rho}_c^{a-} \end{pmatrix}$, $\delta z_v^- \leftarrow \begin{pmatrix} \tilde{\dot{\rho}}_a^1 - \hat{\dot{r}}_{a1}^- - \delta\hat{\dot{\rho}}_c^{a-} \\ \tilde{\dot{\rho}}_a^2 - \hat{\dot{r}}_{a2}^- - \delta\hat{\dot{\rho}}_c^{a-} \\ \tilde{\dot{\rho}}_a^3 - \hat{\dot{r}}_{a3}^- - \delta\hat{\dot{\rho}}_c^{a-} \\ \tilde{\dot{\rho}}_a^4 - \hat{\dot{r}}_{a4}^- - \delta\hat{\dot{\rho}}_c^{a-} \\ \tilde{\dot{\rho}}_a^5 - \hat{\dot{r}}_{a5}^- - \delta\hat{\dot{\rho}}_c^{a-} \\ \tilde{\dot{\rho}}_a^6 - \hat{\dot{r}}_{a6}^- - \delta\hat{\dot{\rho}}_c^{a-} \\ \tilde{\dot{\rho}}_a^7 - \hat{\dot{r}}_{a7}^- - \delta\hat{\dot{\rho}}_c^{a-} \\ \tilde{\dot{\rho}}_a^8 - \hat{\dot{r}}_{a8}^- - \delta\hat{\dot{\rho}}_c^{a-} \end{pmatrix}$      {$\delta z_p^-, \delta z_v^-$: innovation vector}
18:     $H_G^e \leftarrow \begin{pmatrix} -u_{a1,x}^e & -u_{a1,y}^e & -u_{a1,z}^e & 1 \\ -u_{a2,x}^e & -u_{a2,y}^e & -u_{a2,z}^e & 1 \\ -u_{a3,x}^e & -u_{a3,y}^e & -u_{a3,z}^e & 1 \\ -u_{a4,x}^e & -u_{a4,y}^e & -u_{a4,z}^e & 1 \\ -u_{a5,x}^e & -u_{a5,y}^e & -u_{a5,z}^e & 1 \\ -u_{a6,x}^e & -u_{a6,y}^e & -u_{a6,z}^e & 1 \\ -u_{a7,x}^e & -u_{a7,y}^e & -u_{a7,z}^e & 1 \\ -u_{a8,x}^e & -u_{a8,y}^e & -u_{a8,z}^e & 1 \end{pmatrix}$      {$H_G^e$: measurement matrix}
19:     $\hat{p}^+ \leftarrow \hat{p}^- + (H_G^{e^T} H_G^e)^{-1} H_G^{e^T} \delta z_p^-$  {computes the position and clock offset using least squares}
```

```
20:     $\hat{v}^+ \leftarrow \hat{v}^- + ({H_G^e}^T H_G^e)^{-1} {H_G^e}^T \delta z_v^-$   {computes the velocity and clock drift using least squares}
21:     $x^+ \leftarrow (\hat{p}^+, \hat{v}^+)^T$   {update the GNSS position and velocity}
22:     $converge\ test \leftarrow norm(x_0 - x^+)$   {computes the distance between two positions}
23:     if $converge\ test \geq \epsilon$ then   {$\epsilon$: small number (0.001)}
24:         $update\ x_0 \leftarrow x^+$
25:     else
26:         $break\ the\ loop$
27:     end if
28: end while
29: $\hat{x}_0^+ \leftarrow x^+$
30: $(\sigma_p, \sigma_r) \leftarrow (10, 0.05)$   {$\sigma_p, \sigma_r$: a noise standard deviation on all pseudo-range and range rates}
31: $P_0^+ \leftarrow \begin{pmatrix} {\sigma_p}^2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & {\sigma_p}^2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & {\sigma_p}^2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & {\sigma_r}^2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & {\sigma_r}^2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & {\sigma_r}^2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & {\sigma_p}^2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & {\sigma_r}^2 \end{pmatrix}$
Output: state vector estimation x, error covariance matrix P
```

From lines 19 and 20 in algorithm 1, least square estimation was used to compute the initial GNSS position and velocity of the lawnmower. Furthermore, line 22 represents that a distance between two predicted positions of the lawnmower was measured to compare how close they are, and the epsilon was assigned to be a small number, 0.001. If the distance between them was sufficiently close enough, it can be seen that the solution converges to a ground truth. After satisfying the above conditions, the state vector estimation and error covariance matrix can be updated to compute the next epoch into the Kalman Filter function.

2.2. Kalman Filter

A Kalman filter was used to provide estimates of the position and velocity due to the continually changing state of the robotic lawnmower. The Kalman filter is initially supplied with a set of estimates based on the measurements taken by the receiver and recursively operates to use a weighted average of the previous position's measurement values and the new measurement values to provide an estimate [1]. The steps detailed in [1] section 3.2.2 were followed which indicated that the Kalman filter comprised of 10 steps.

It is important to define the elements of the Kalman filter algorithm to understand the ensuing code explanation. The Kalman filter uses a state vector ***x*** as an input, which is computed using the least squares estimation mentioned earlier which includes an estimated position, velocity, clock offset and rate of clock offset (clock drift). The second element is the error covariance matrix, ***P***, which represents the uncertainties in the output estimates by quantifying the correlation between error and the output by the Kalman filter. The algorithm further uses the pseudo range and the pseudo-range rates measurements provided by the simulated data.

Initially, the transition matrix, Φ_{i-1} , is computed to indicate how the state vector changes with time and is defined in the code under Algorithm 2, Line 5. Since the measurement uncertainties and error (coming from systematic errors, white noise and Gauss-Markov sequences) accumulate with time, the next step involved computing the system noise covariance matrix. This is a symmetric matrix that uses the inputs of power spectral density (PSD), S_a, which is a function of clock frequency PSD, S_{cf}^a, and has a clock phase PSD, $S_{c\emptyset}^a$ [1].

Steps 3 and 4 involves propagation of the state vector and the error covariance using the transition matrix, as defined by equations 6-7 (3.14-3.15 from [1]).

$$\hat{x}_i^- = \Phi_{i-1}\hat{x}_{i-1}^+ \tag{6}$$
$$P_i^- = \Phi_{i-1}P_{i-1}^+\Phi_{i-1}^T + Q_{i-1} \tag{7}$$

After the system-propagation phase of the filter, the measurement-update phase begins. Step 5 involves use of the measurement matrix, H_i, that comprises of a series of measurements from the receiver and the computed line of sight from each satellite used. This matrix is used to show how the measurement vector varies with the state vector and is assumed to have a linear relationship. Step 6, accounts for the measurement noise by computing the measurement noise covariance matrix, R_i. This is a diagonal matrix where the pseudo-range and pseudo-range rate measurements error standard deviations are accounted for. They have been assumed to be constant in this implementation of the algorithm.
In step 7, the Kalman gain matrix, K_i, uses inputs of H_i, P_i^- and R_i to determine the weighting of the measurements when updating the state estimates as shown in equation 8 (equation 3.21 from [1]).

$$K_i = P_i^- H_i^T (H_i P_i^- H_i^T + R_k)^{-1} \tag{8}$$

Step 8, computes the measurement innovation vector δz^- by differencing the measurement vector by the predicted measurement vector from the predicted state vector as shown in line 20, algorithm 2 (lecture 2A from [3]).
Finally, steps 9 and 10 completes this open-loop implementation of the Kalman filter by updating the state vector estimate and the error covariance matrix using the Kalman gain matrix as shown in equations 9-10.

$$\hat{x}_i^+ \leftarrow \hat{x}_i^- + K_i\delta z_i^- \tag{9}$$
$$P_i^+ \leftarrow (I - K_iH_i)P_i^- \tag{10}$$

Algorithm 2 Computation of GNSS Kalman Filter with the solution of outlier detection [5], [6], [7]

```
Input: state vector estimation x, error covariance matrix P, pseudo ranges r, pseudo range rates ṙ,
       satellite s, time t
1:  initialise a state vector estimation, $\hat{x}_0^+ \leftarrow x$          {$\hat{x}_0^+$ comes from the algorithm 1}
2:  initialise a error covariance matrix estimation, $P_0^+ \leftarrow P$   {$P_0^+$ comes from the algorithm 1}
3:  $\tau_s \leftarrow 0.5$
4:  for the number of epochs i = 1…n do
5:      Step 1: $\Phi_{i-1} \leftarrow \begin{pmatrix} I_3 & \tau_s I_3 & 0_{3,1} & 0_{3,1} \\ 0_3 & I_3 & 0_{3,1} & 0_{3,1} \\ 0_{1,3} & 0_{1,3} & 1 & \tau_s \\ 0_{1,3} & 0_{1,3} & 0 & 1 \end{pmatrix}$
6:      Step 2: $Q_{i-1} \leftarrow \begin{pmatrix} \frac{1}{3}S_a{\tau_s}^3 I_3 & \frac{1}{2}S_a{\tau_s}^2 I_3 & 0_{3,1} & 0_{3,1} \\ \frac{1}{2}S_a{\tau_s}^2 I_3 & S_a\tau_s I_3 & 0_{3,1} & 0_{3,1} \\ 0_{1,3} & 0_{1,3} & S_{c\emptyset}^a\tau_s + \frac{1}{3}S_{cf}^a{\tau_s}^3 & \frac{1}{2}S_{cf}^a{\tau_s}^2 \\ 0_{1,3} & 0_{1,3} & \frac{1}{2}S_{cf}^a{\tau_s}^2 & S_{cf}^a\tau_s \end{pmatrix}$
7:      Step 3: $\hat{x}_i^- \leftarrow \Phi_{i-1}\hat{x}_{i-1}^+$
8:      Step 4: $P_i^- \leftarrow \Phi_{i-1}P_{i-1}^+\Phi_{i-1}^T + Q_{i-1}$
9:      for the number of satellite k = 1…8 do
10:         Set the initial range ($\hat{r}_{ak}^- = r_1$) and range rate ($\hat{\dot{r}}_{ak}^- = \dot{r}_1$) from the user position to each satellite
```

```
11:        $C_e^I \leftarrow \begin{pmatrix} 1 & \omega_{ie} r_{ak}/c & 0 \\ -\omega_{ie} r_{ak}/c & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$
12:        *Computes the predict the ranges and range rates from user position to each satellite*
13:        $\hat{r}_{ak}^- \leftarrow \sqrt{[C_e^I(\hat{r}_{ak}^-)\hat{r}_{ek}^e - \hat{r}_{ea}^e]^T [C_e^I(\hat{r}_{ak}^-)\hat{r}_{ek}^e - \hat{r}_{ea}^e]}$
14:        $\hat{\dot{r}}_{ak}^- \leftarrow \hat{u}_{ak}^{e-T}[C_e^I(\hat{r}_{ak}^-)(\hat{v}_{ek}^e + \Omega_{ie}^e \hat{r}_{ek}^e) - (\hat{v}_{ek}^{e-} + \Omega_{ie}^e \hat{r}_{ek}^{e-})]$
15:        $u_{ak}^e \leftarrow (\frac{C_e^I(\hat{r}_{ak}^-)\hat{r}_{ek}^e - \hat{r}_{ea}^e}{\hat{r}_{ak}^-})$
16:    **end for**
17:    **Step 5:** $H_i \leftarrow \begin{pmatrix} -u_{a1,x}^e & -u_{a1,y}^e & -u_{a1,z}^e & 0 & 0 & 0 & 1 & 0 \\ -u_{a2,x}^e & -u_{a2,y}^e & -u_{a2,z}^e & 0 & 0 & 0 & 1 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ -u_{ak,x}^e & -u_{ak,y}^e & -u_{ak,z}^e & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -u_{a1,x}^e & -u_{a1,y}^e & -u_{a1,z}^e & 0 & 1 \\ 0 & 0 & 0 & -u_{a2,x}^e & -u_{a2,y}^e & -u_{a2,z}^e & 0 & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & -u_{ak,x}^e & -u_{ak,y}^e & -u_{ak,z}^e & 0 & 1 \end{pmatrix}$
18:    **Step 6:** $R_i \leftarrow \begin{pmatrix} {\sigma_p}^2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & {\sigma_p}^2 & 0 & 0 & 0 & 0 & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & {\sigma_p}^2 & 0 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 & {\sigma_r}^2 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 & 0 & {\sigma_r}^2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 & 0 & 0 & \cdots & {\sigma_r}^2 \end{pmatrix}$
19:    **Step 7:** $K_i \leftarrow P_i^- H_i^T (H_i P_i^- H_i^T + R_k)^{-1}$
20:    **Step 8:** $\delta z^- = \begin{pmatrix} \tilde{\rho}_a^1 - \hat{r}_{a1}^- - \delta\hat{\rho}_c^{a-} \\ \tilde{\rho}_a^2 - \hat{r}_{a2}^- - \delta\hat{\rho}_c^{a-} \\ \vdots \\ \tilde{\rho}_a^k - \hat{r}_{ak}^- - \delta\hat{\rho}_c^{a-} \\ \tilde{\dot{\rho}}_a^1 - \hat{\dot{r}}_{a1}^- - \delta\hat{\dot{\rho}}_c^{a-} \\ \vdots \\ \tilde{\dot{\rho}}_a^1 - \hat{\dot{r}}_{a2}^- - \delta\hat{\dot{\rho}}_c^{a-} \\ \tilde{\dot{\rho}}_a^1 - \hat{\dot{r}}_{ak}^- - \delta\hat{\dot{\rho}}_c^{a-} \end{pmatrix}$
21:    **Step 9:** $\hat{x}_i^+ \leftarrow \hat{x}_i^- + K_i \delta z_i^-$
22:    **Step 10:** $P_i^+ \leftarrow (I - K_i H_i) P_i^-$
23:    *computes the* ***outlier detection*** *solution*
24:    *initialise the residual ionosphere error standard deviation at zenith,* $\sigma_{iono} \leftarrow 2$
25:    *initialise the residual troposphere error standard deviation at zenith,* $\sigma_{tropo} \leftarrow 0.2$
26:    *initialise the outlier detection threshold,* $T \leftarrow 6$
27:    $H_G^e \leftarrow \begin{pmatrix} -u_{a1,x}^e & -u_{a1,y}^e & -u_{a1,z}^e & 1 \\ -u_{a2,x}^e & -u_{a2,y}^e & -u_{a2,z}^e & 1 \\ \vdots & \vdots & \vdots & \vdots \\ -u_{ak,x}^e & -u_{ak,y}^e & -u_{ak,z}^e & 1 \end{pmatrix}$, $\delta z^- \leftarrow \begin{pmatrix} \tilde{\rho}_a^1 - \hat{r}_{a1}^- - \delta\hat{\rho}_c^{a-} \\ \tilde{\rho}_a^2 - \hat{r}_{a2}^- - \delta\hat{\rho}_c^{a-} \\ \vdots \\ \tilde{\rho}_a^k - \hat{r}_{ak}^- - \delta\hat{\rho}_c^{a-} \end{pmatrix}$
28:    **Step 1:** $v \leftarrow (H_G^e \left({H_G^e}^T H_G^e\right)^{-1} {H_G^e}^T - I_k)\delta z^-$
```

```
29:       $\sigma_{z0} \leftarrow \begin{pmatrix} (\sigma_{iono} + \sigma_{tropo})_1 \\ (\sigma_{iono} + \sigma_{tropo})_2 \\ \vdots \\ (\sigma_{iono} + \sigma_{tropo})_k \end{pmatrix}$
30:       $\theta_i \leftarrow u^e_{ak,z} \,/\, \hat{r}^-_{ak}$   (Lecture 1D from [3])
31:       $\sigma_{zs} \leftarrow \sigma_{z0} \,/\, sin\theta_i$   (Lecture 1D from [3])
32:       $E_{zs} \leftarrow {\sigma_{zs}}^2$
33:   Step 2: $C_v \leftarrow (I_k - H^e_G (H^{eT}_G H^e_G)^{-1} H^{eT}_G) E_{zs}$
34:   Step 3: $C_{v,kk} \leftarrow diagnal\ of\ element\ (C_v) * E_{zs}$
35:       $w_i \leftarrow v^T \,/\, \sqrt{C_{v,kk}}$
36:   Step 4: $check\ condition \leftarrow |w_i| > T$    $(= |v| > \sqrt{C_{v,kk}} * T)$ [6]
37:       $outlier \leftarrow |w_i| - T$
38:       if $outlier > 0$ then
39:           $O \leftarrow check\ condition$
40:       else
41:           $O \leftarrow empty$
42:       end if
43: end for
Output: state vector estimation $\boldsymbol{x}^+_i$, error covariance matrix $\boldsymbol{P}^+_i$, outlier detection $\boldsymbol{O}$ (depends on the condition)
```

This algorithm followed the steps of the Kalman Filter provided by [5]. Between lines 23 and 42 the pseudocode uses the outlier detection function, which investigated the measurement error hypothesis testing [6]. The equation was based on the residual outlier detection and if the outlier detection condition is not satisfied, the output of the outlier detection function is empty [6]. Furthermore, Kalman Filter can be tuned to make a stable condition and minimise the error by suitable values for the system and measurement noise covariance matrix (Q_i), (R_i), respectively and initial state estimation error covariance matirx (P^+_0) [7]. MATLAB code is used to decrease an error of the Kalman filter by tuning R_i as shown in Appendix A.1. The next computation is moving to algorithm 3 to update a new range, range rates and satellite number. On the other hand, if the condition of outlier detection from line 36 is satisfied, the function of Kalman Filter was recalculated step by step for every epoch to retain the updated version of state vector estimation and error covariance matrix, respectively.

2.3. Outlier Detection

After computing the Kalman Filter algorithm, the condition of the outlier detection can be checked and if the residual condition is not satisfied, the algorithm 3 would be run to recalculate the position, velocity and receiver clock with new ranges and range rates [4]. The main purpose of this algorithm is to find the large residual of the satellite, which contaminates in the solution of satellite position and retains the small residual of the satellite to reduce error. Therefore, new ranges, range rates and satellite number would be computed, and these parameters will be used to recalculate using the Kalman Filter function.

Algorithm 3 Outlier Detections [4]

```
Input: pseudo ranges $\boldsymbol{r}$, pseudo range rates $\dot{\boldsymbol{r}}$, satellite $\boldsymbol{s}$, time $\boldsymbol{t}$, outlier detection $\boldsymbol{O}$
1: for the number of epochs i = 1...n do
2:     computes the Kalman Filter of the algorithm 2 with $(x, P, r, \dot{r}, s, t)$
3:     if outlier detection is empty then
4:         $s_{new} \leftarrow s$
```

```
5:         $r_{new} \leftarrow r$
6:         $\dot{r}_{new} \leftarrow \dot{r}$
7:         *Initialise* $O_0 \leftarrow 0$
8:         **for** *the number of satellite k = 1…8* **do**
9:             **if** *outlier detection(k) occurs* **then**
10:                $O_0 \leftarrow O_0 + 1$
11:                $s_{new} \leftarrow s_{new}(k - O_0)$
12:                $r_{new} \leftarrow r_{new}(k - O_0)$
13:                $\dot{r}_{new} \leftarrow \dot{r}_{new}(k - O_0)$
14:            **end if**
15:        **end for**
16:        **output**: update ranges $\boldsymbol{r_{new}}$, update range rates $\boldsymbol{\dot{r}_{new}}$, update satellite $\boldsymbol{s_{new}}$
17:        *re-computes the Kalman Filter of the algorithm 2 with* $(x, P, r_{new}, \dot{r}_{new}, s_{new}, t)$
18:     **end if**
19: **end for**
```

2.4. Gyro-Magnetometer Integration

After the GNSS computation, the GNSS results were used to initialize the parameters including time, latitude, longitude, North velocity, East velocity, and the height. The heading was corrected by the integration of the gyroscope and magnetometer. Algorithm 4's pseudo-code below details the steps taken to integrate the gyroscope and magnetometer [8].

Algorithm 4 Gyro-Magnetometer Integration using a 2-state Kalman filter

Input: gyroscope bias standard deviation $\boldsymbol{\sigma_b} = \mathbf{1}$, magnetometer noise-like error standard deviation $\boldsymbol{\sigma_m} = \mathbf{4}$,gyroscope angular rate $\dot{\boldsymbol{\alpha}}$, heading measurements from the magnetic compass $\boldsymbol{H^m} = \mathbf{4}$, gyroscope noise standard deviation $\boldsymbol{\sigma_n} = \mathbf{10^{-4}}$, gyroscope random noise with PSD $\boldsymbol{S_{rg}} = \mathbf{3 \times 10^{-6}}$, gyroscope bias variation with PSD $\boldsymbol{S_{bgd}} = \mathbf{3 \times 10^{-6}}$, propagation interval $\boldsymbol{\tau_s} = \mathbf{0.5}$, total epochs $\boldsymbol{i} = \mathbf{851}$.

```
1:  **procedure:** INITIALISATION
2:  $H^G \leftarrow zeros(851, 0)\ and\ H^G(1) = 0$        { heading measurements from the gyroscope}
3:  **for** *j* = 2:*i*
4:      $H^G(j) \leftarrow H^G(j-1) + 0.5 \times \dot{\boldsymbol{\alpha}}(j)$        {calculate heading from the gyroscope}
5:  **end for**
6:  $P_{k-1} \leftarrow \begin{bmatrix} {\sigma_n}^2 & 0 \\ 0 & {\sigma_b}^2 \end{bmatrix}$        {$P_{k-1}$: state estimation error covariance matrix}
7:  $X_{k-1} \leftarrow [0 \quad 0]$        {$X_{k-1}$: state vector}
8:  $H_c \leftarrow zeros(851, 0)$        {$H_c$: corrected heading}
9:  $H_c(1) \leftarrow H^m(1)$        {since $H^G(1) = 0$, we set the first epoch of $H_c = H^m(1)$}
10: *Computes the corrected heading for each epoch using Kalman filter*
11: **for** k = 2:*i*
12:     $\varphi_{k-1} \leftarrow \begin{bmatrix} 1 & \tau_s \\ 0 & 1 \end{bmatrix}$        {Compute the transition matrix}
13:     $Q_{k-1} \leftarrow \begin{bmatrix} S_{rg}\tau_s + \frac{1}{3}S_{bgd}\tau_s^3 & \frac{1}{2}S_{bgd}\tau_s^2 \\ \frac{1}{2}S_{bgd}\tau_s^2 & S_{bgd}\tau_s \end{bmatrix}$        {Compute the system noise covariance matrix}
```

```
14:     $X_k^- \leftarrow Q_{k-1}X_{k-1}$                                  {Propagate the state estimates}
15:     $P_k^- \leftarrow \varphi_{k-1}P_{k-1}\varphi_{k-1}^T + Q_{k-1}$                {Propagate the error covariance matrix}
16:     $H_k \leftarrow [-1 \quad 0]$                                      {Compute the measurement matrix}
17:     $R_k \leftarrow [\sigma_m^2]$                                      {Compute the measurement noise covariance matrix}
18:     $K_k \leftarrow P_k^- H_k^T (H_k P_k^- H_k^T + R_k)^{-1}$             {Compute the Kalman gain matrix}
19:     $\delta z_k \leftarrow [H^m(k) - H^G(k)] + H_k X_k^-$                 {Formulate the measurement innovation vector}
20:     $X_k^+ \leftarrow X_k^- + K_k \delta z_k$                            {Update the state estimates}
21:     $P_k^+ \leftarrow (I - K_k H_k) P_k^-$                              {Update the error covariance matrix}
22:     $H_c(k) \leftarrow H^G(k) - X_k^+(1)$                               {Store the value in corrected heading}
23:     $P_{k-1} \leftarrow P_k^+$                                         {Update state estimation error covariance matrix}
24:     $X_{k-1} \leftarrow X_k^+$                                         {Update state vector}
25: **end for**
**Output**: Integrated heading $\boldsymbol{H_c}$ in degrees
```

2.5. Dead Reckoning Calculation

The corrected heading by the integration of Gyro-Magnetometer was then used to find the Dead Reckoning results [9] in the form of the Geodetic latitude, Geodetic longitude, the North velocity, and the East velocity.

Algorithm 5 Dead Reckoning

```
**Input**: height **h** calculated by GNSS, driving speed $\boldsymbol{v}$, initial Geodetic latitude $\boldsymbol{GNSS_{lat}}$ calculated by GNSS,
initial Geodetic longitude $\boldsymbol{GNSS_{long}}$ calculated by GNSS, corrected heading $\boldsymbol{H_c}$
1:  **procedure:** INITIALISATION
2:  $DR_{lat}(1) \leftarrow GNSS_{lat}(1)$                  $\{DR_{lat}: Dead\ Reckoning\ Geodetic\ latitude\}$
3:  $DR_{long}(1) \leftarrow GNSS_{long}(1)$                $\{DR_{long}: Dead\ Reckoning\ Geodetic\ longitude\}$
4:  $DR_{Nv}(1) \leftarrow 0$                              $\{DR_{Nv}: Dead\ Reckoning\ North\ velocity\}$
5:  $DR_{Ev}(1) \leftarrow 0$                              $\{DR_{Ev}: Dead\ Reckoning\ East\ velocity\}$
6:  *Computes the Dead Reckoning solutions using corrected heading*
7:  **for** *k = 2:i*
8:      $\begin{bmatrix} V_k^N \\ V_k^E \end{bmatrix} \leftarrow \frac{1}{2}\begin{bmatrix} \cos H_c(k) + \cos H_c(k-1) \\ \sin H_c(k) + \sin H_c(k-1) \end{bmatrix} v_k$
                                   $\{V_k^N, V_k^E: average\ North\ and\ East\ velocity\ on\ each\ epoch\}$
9:      $[R_N,\ R_E] \leftarrow$ *Radii_of_curvature*$(DR_{lat}(k-1))$
                                   {Using latitude to compute radii of curvature}
10:     $DR_{lat}(k) \leftarrow DR_{lat}(k-1) + \frac{0.5V_k^N}{R_N + h(k)}$       {Compute Dead Reckoning Geodetic latitude}
11:     $DR_{long}(k) \leftarrow DR_{long}(k-1) + \frac{0.5V_k^E}{R_E + h(k)}$     {Compute Dead Reckoning Geodetic longitude}
12:     $DR_{Nv}(k) \leftarrow 2V_k^N - DR_{Nv}(k-1)$               {Compute Dead Reckoning North velocity}
13:     $DR_{Ev}(k) \leftarrow 2V_k^E - DR_{Ev}(k-1)$               {Compute Dead Reckoning East velocity}
14: **end for**
**output**: Dead Reckoning results: Geodetic latitude $\boldsymbol{DR_{lat}}$, Geodetic longitude $\boldsymbol{DR_{long}}$ **,** North velocity $\boldsymbol{DR_{Nv}}$ ,
East velocity $\boldsymbol{DR_{Ev}}$ .
```

2.6. Integration of GNSS and Dead Reckoning

Finally, the GNSS and Dead Reckoning solutions were integrated into one function using a 4-state Kalman filter [10].

Algorithm 6 Integration of GNSS and Dead Reckoning using a 4-state Kalman filter

Input: initial velocity uncertainty $\boldsymbol{\sigma_v = 0.1}$, initial position uncertainty $\boldsymbol{\sigma_r = 10}$, GNSS position noise standard deviation $\boldsymbol{\sigma_{Gr} = 10}$, GNSS velocity noise standard deviation $\boldsymbol{\sigma_{Gv} = 0.05}$, velocity error variance with PSD $\boldsymbol{S_{DR} = 0.01}$, propagation interval $\boldsymbol{\tau_s = 0.5}$, total epochs $\boldsymbol{i = 851}$, height **h** calculated by GNSS, Geodetic latitude $\boldsymbol{GNSS_{lat}}$ calculated by GNSS, Geodetic latitude $\boldsymbol{DR_{lat}}$ calculated by Dead Reckoning, Geodetic longitude $\boldsymbol{GNSS_{long}}$ calculated by GNSS, Geodetic longitude $\boldsymbol{DR_{long}}$ calculated by Dead Reckoning, North velocity $\boldsymbol{GNSS_{Nv}}$ calculated by GNSS, North velocity $\boldsymbol{DR_{Nv}}$ calculated by Dead Reckoning, East velocity $\boldsymbol{GNSS_{Ev}}$ calculated by GNSS, East velocity $\boldsymbol{DR_{Ev}}$ calculated by Dead Reckoning, heading $\boldsymbol{H_c}$, time **t.**

```
1:  procedure: INITIALISATION
2:  $[R_N{}^{initial},\ R_E{}^{initial;}] \leftarrow Radii\_of\_curvature(GNSS_{lat}(1))$
                                        {Using latitude to compute radii of curvature}
3:  $P_{k-1} \leftarrow \begin{bmatrix} \sigma_v^2 & 0 & 0 & 0 \\ 0 & \sigma_v^2 & 0 & 0 \\ 0 & 0 & \frac{\sigma_r^2}{\left(R_N{}^{initial}+h(1)\right)^2} & 0 \\ 0 & 0 & 0 & \frac{\sigma_r^2}{\left(R_E{}^{initial}+h(1)\right)^2 \cos^2 GNSS_{lat}(1)} \end{bmatrix}$
                                        {$P_{k-1}$: state estimation error covariance matrix}
4:  $X_{k-1} \leftarrow [0 \quad 0 \quad 0 \quad 0]$                    {$X_{k-1}$: state vector}
5:  Results ← zeros(851, 6)   { Results: final results of integration of GNSS and Dead Reckoning}
6:  Results(: , 1) ← t                          {Store time in final result column 1}
7:  Results(: , 6) ← $H_c$                      {Store corrected heading in final result column 6}
8:  Computes the integrated latitude, longitude, north and east velocity for each epoch using Kalman filter
9:  for k = 2: i
10:     $[R_N,\ R_E] \leftarrow Radii\_of\_curvature(GNSS_{lat}(k))$ {Using latitude to compute radii of curvature}
11:     $\varphi_{k-1} \leftarrow \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ \frac{\tau_s}{R_N+h(k-1)} & 0 & 1 & 0 \\ 0 & \frac{\tau_s}{(R_E+h(k-1))\cos GNSS_{lat}(k-1)} & 0 & 1 \end{bmatrix}$   {Compute the transition matrix}
12:     $Q_{k-1} \leftarrow \begin{bmatrix} S_{DR}\tau_s & 0 & \frac{\frac{1}{2}S_{DR}\tau_s{}^2}{R_N+h(k-1)} & 0 \\ 0 & S_{DR}\tau_s & 0 & \frac{\frac{1}{2}S_{DR}\tau_s{}^2}{(R_E+h(k-1))\cos GNSS_{lat}(k-1)} \\ \frac{\frac{1}{2}S_{DR}\tau_s{}^2}{R_N+h(k-1)} & 0 & \frac{\frac{1}{3}S_{DR}\tau_s{}^3}{(R_N+h(k-1))^2} & 0 \\ 0 & \frac{\frac{1}{2}S_{DR}\tau_s{}^2}{(R_E+h(k-1))\cos GNSS_{lat}(k-1)} & 0 & \frac{\frac{1}{3}S_{DR}\tau_s{}^3}{(R_E+h(k-1))^2\cos^2 GNSS_{lat}(k-1)} \end{bmatrix}$
                                        {Compute the system noise covariance matrix}
13:     $X_k{}^- \leftarrow Q_{k-1}X_{k-1}$                 {Propagate the state estimates}
14:     $P_k{}^- \leftarrow \varphi_{k-1}P_{k-1}\varphi_{k-1}{}^T + Q_{k-1}$   {Propagate the error covariance matrix}
15:     $H_k \leftarrow \begin{bmatrix} 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & -1 \\ -1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \end{bmatrix}$            {Compute the measurement matrix}
```

```
16:     $R_k \leftarrow \begin{bmatrix} \frac{{\sigma_{Gr}}^2}{(R_N+h(k))^2} & 0 & 0 & 0 \\ 0 & \frac{{\sigma_{Gr}}^2}{(R_E+h(k))^2 \cos^2 GNSS_{lat}(k)} & 0 & 0 \\ 0 & 0 & {\sigma_{Gv}}^2 & 0 \\ 0 & 0 & 0 & {\sigma_{Gv}}^2 \end{bmatrix}$
            {Compute measurement noise covariance matrix}
17:     $K_k \leftarrow {P_k}^- H_k^T (H_k {P_k}^- H_k^T + R_k)^{-1}$    {Compute the Kalman gain matrix}
18:     $\delta z_k \leftarrow \begin{bmatrix} GNSS_{lat}(k) - DR_{lat}(k) \\ GNSS_{long}(k) - DR_{long}(k) \\ GNSS_{Nv}(k) - DR_{Nv}(k) \\ GNSS_{Ev}(k) - DR_{Ev}(k) \end{bmatrix} + H_k {X_k}^-$  {Formulate the measurement innovation vector}
19:     ${X_k}^+ \leftarrow {X_k}^- + K_k \delta z_k$    {Update the state estimates}
20:     ${P_k}^+ \leftarrow (I - K_k H_k) {P_k}^-$    {Update the error covariance matrix}
21:     $Results(k{:}2) \leftarrow DR_{lat}(k) - {X_k}^+(3)$  {Store integrated Geodetic latitude to final results}
22:     $Results(k{:}3) \leftarrow DR_{long}(k) - {X_k}^+(4)$ {Store integrated Geodetic longitude to final results}
23:     $Results(k{:}4) \leftarrow DR_{Nv}(k) - {X_k}^+(1)$  {Store the integrated North velocity to final results}
24:     $Results(k{:}5) \leftarrow DR_{Ev}(k) - {X_k}^+(2)$  {Store the integrated East velocity to final results}
25:     $P_{k-1} \leftarrow {P_k}^+$    {Update state estimation error covariance matrix}
26:     $X_{k-1} \leftarrow {X_k}^+$    {Update state vector}
27: end for
Output: Integrated results: time $t$, Geodetic latitude $DR_{lat}$, Geodetic longitude $DR_{long}$, North velocity $DR_{Nv}$, East velocity $DR_{Ev}$, heading $H_c$
```

3. Results

The results of the integrated solution using a 4-state Kalman filter are discussed in this section. Figure 1 shows the trajectory of the latitude for the lawnmower plotted against time where a sinusoidal like shape is shown. The is an expected result as if the intention is to cut in an expected rectangular-like fashion, the lawnmower would go up and down (either in the latitude or longitude directions). The maximum and minimum latitudes reached following the initial starting position is largely consistent with each cycle. It follows that Figure 2 shows an expected longitudinal trajectory for the lawn mowing in a rectangular-like fashion, since the lawnmower trajectory incrementally increases with each cycle at discrete steps following the maximum position reached.

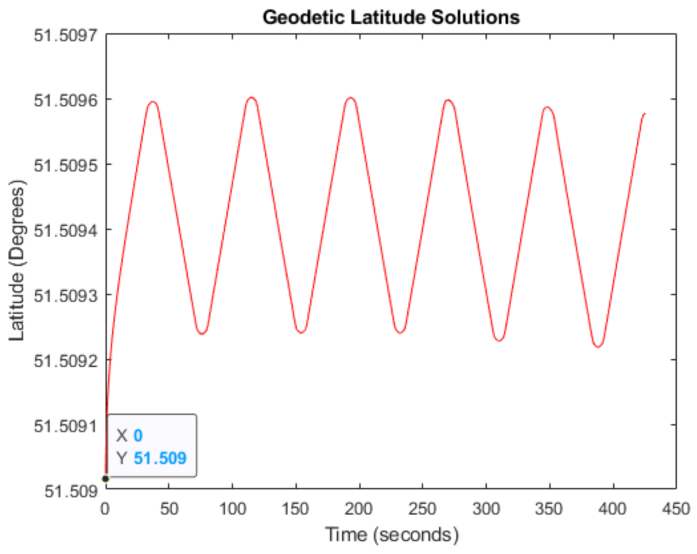

Figure 1: Representation of a latitude trajectory for the lawnmower with a starting point at (0, 51.509).

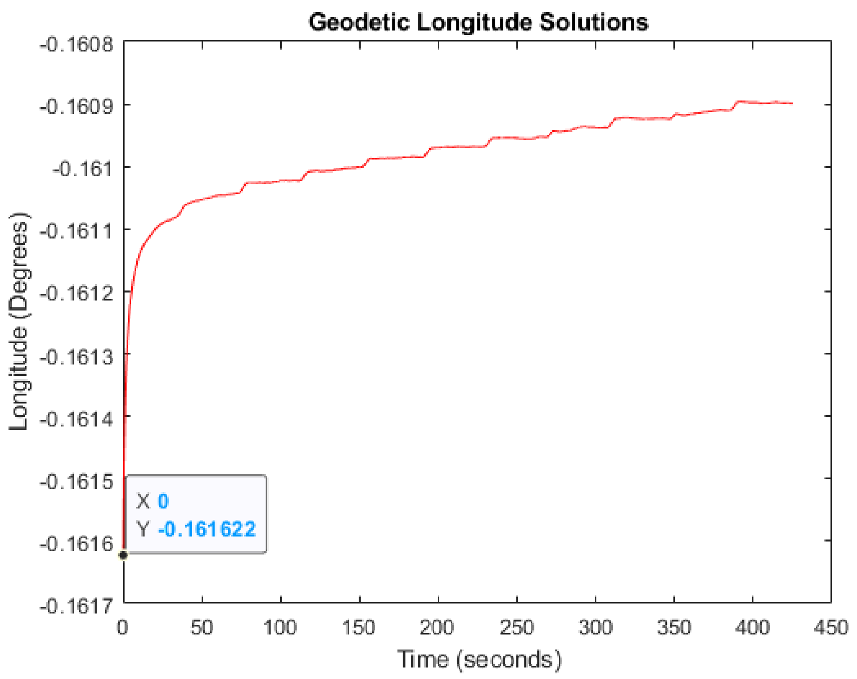

Figure 2. Representation of a longitude trajectory for the lawnmower with a starting point at (0, -0.16162).

The velocity profiles are subject to more noise-like behaviour than the latitude and longitudinal solutions as shown in Figures 3-4. The initial velocity is set as 0, and there is a clear relationship between the north velocity and the variation in latitude as expected. The lawnmower is shown to accelerate to a consistent maximum absolute velocity and spend some time turning when the east velocity increases further affirming that a rectangular mowing behaviour is done. The plot of the heading solutions shown in Figure 5 shows a clear pattern that agrees with Figures 1-4 where the direction of the lawnmower changes from 0 to 180 degrees. The largest difference with figure 5 is that the data has less noise-like behaviour at the point where the lawnmower changes direction relative to Figures 3 and 4.

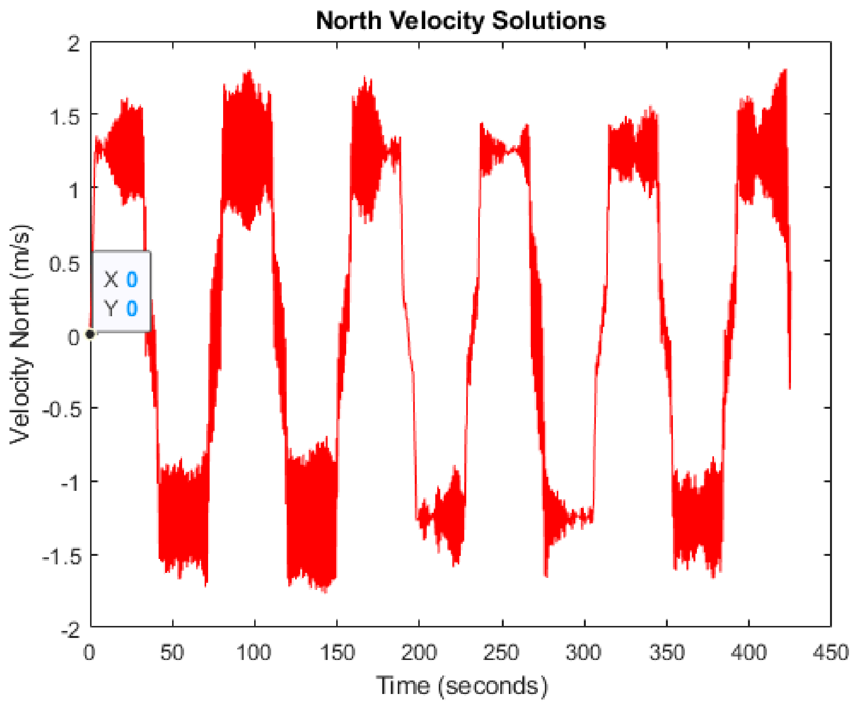

Figure 3: Representation of north velocity (direction) for the lawnmower with a starting point at (0, 0).

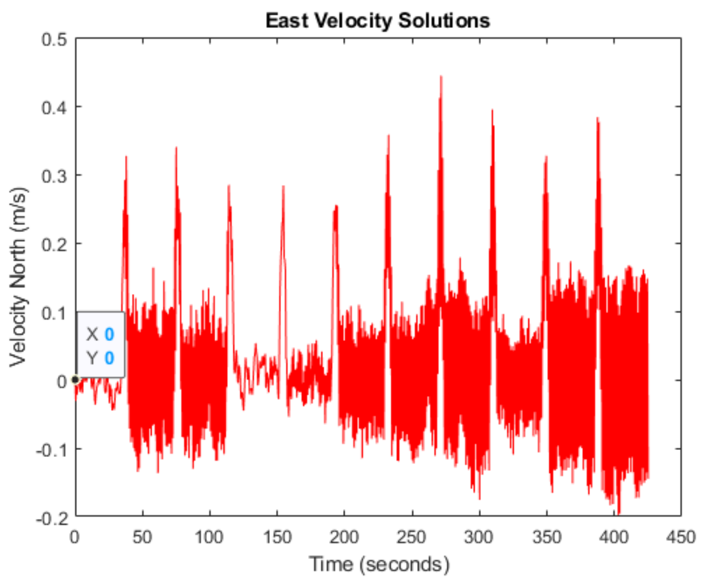

Figure 4: Representation of east velocity (direction) for the lawnmower with a starting point at (0, 0).

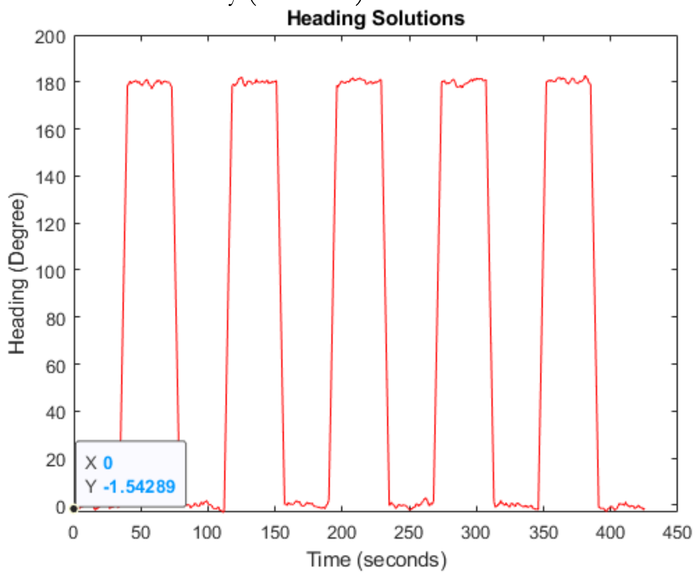

Figure 5: Representation of heading trajectory for the lawnmower with a starting point (0, -1.54289).

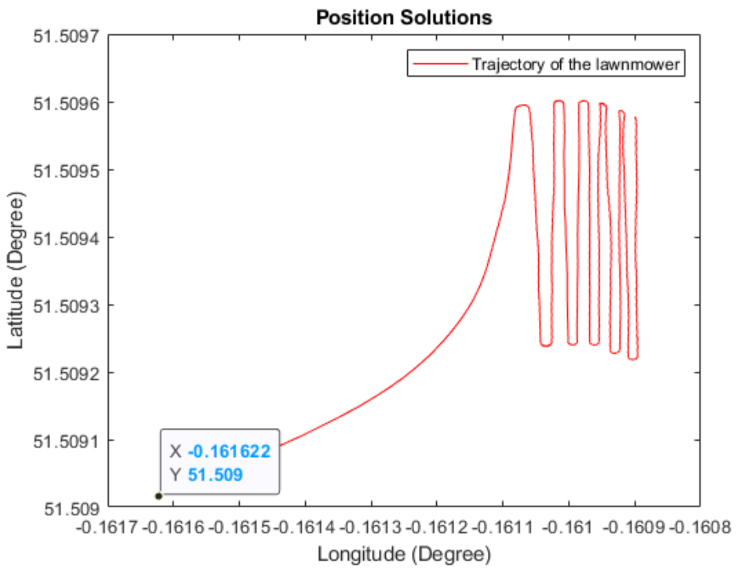

Figure 6: Representation of a position trajectory for the lawnmower with a starting point at (-0.162, 51.509).

Finally, Figure 6 shows the trajectory of the lawnmower plotted on the latitude and longitude axes. There is a large initial movement from the lawnmower initial starting point to where the rectangular mowing takes place.

4. Discussion

Figures 7-8 show the latitude and longitude position solutions for each of the GNSS, Dead Reckoning and Integration solutions plotted against time for comparison. It can be seen that in terms of positioning, the sinusoidal shape for the latitude is present for all three plots with very similar amplitudes. The largest visible difference if the latitude at which the lawnmower position "oscillates" despite an identical starting position. It can be seen that the integrated solution removes some of the noise coming from the GNSS computations while the position is largely weighted towards the GNSS solutions rather than the dead-reckoning.

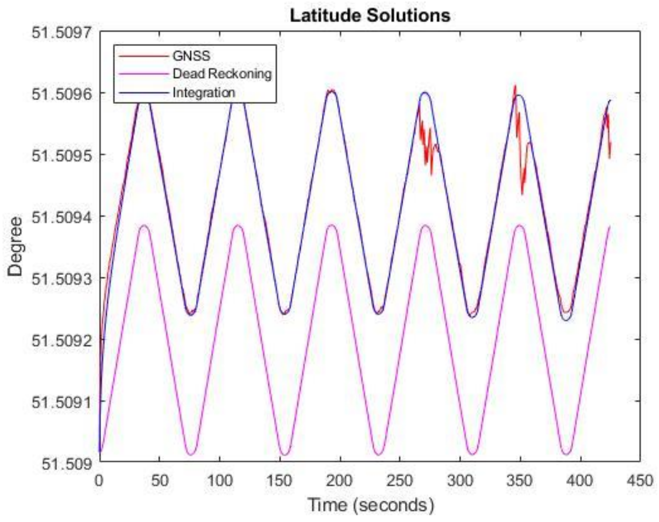

Figure 7: The combined plots from the GNSS results, the dead reckoning results and the integration solution showing the change in latitude with time.

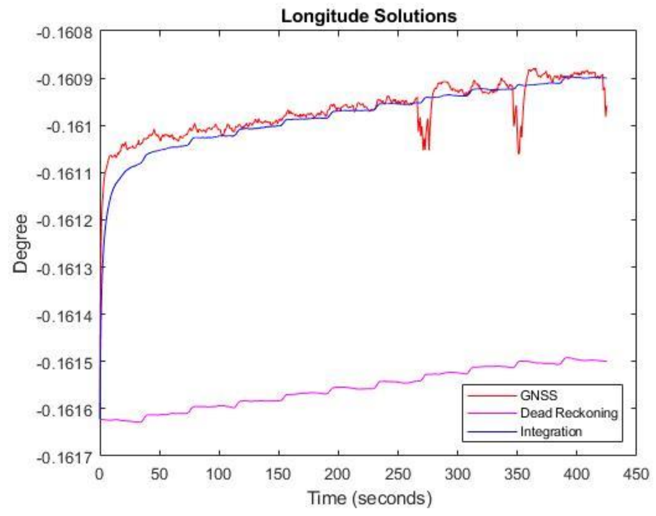

Figure 8: The combined plots from the GNSS results, the dead reckoning results and the integration solution showing the change in longitude with time.

Figures 9-10 show a comparison between the velocity solutions in the north and east directions for each navigation mechanism. From this it is hypothesized that the noise like-behaviour shown in Figures 3 and 4 come from the dead reckoning solution rather than the GNSS. The integrated solution shows a large reduction of the noise, however it is not fully eliminated as explained earlier. The initial east velocity from the GNSS solution is shown to be largely removed from the integrated solution affirming the Kalman filter's effectiveness. This occurred despite the use of an outlier detection algorithm of the GNSS filter which would have removed all points that were considered outliers.

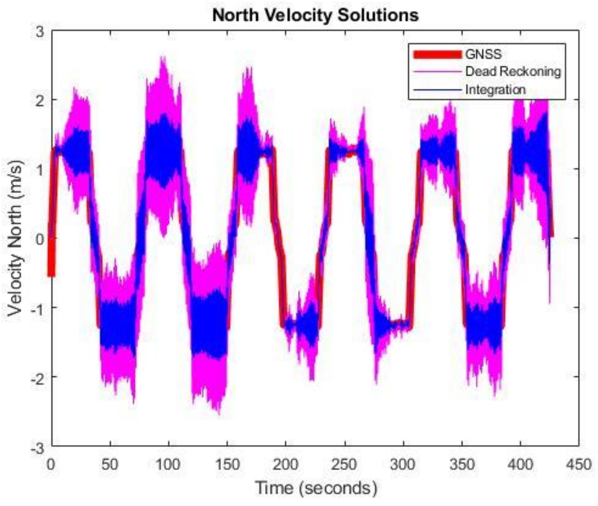

Figure 9: Plots of the GNSS, Dead-Reckoning and Integration solutions for the North Velocity.

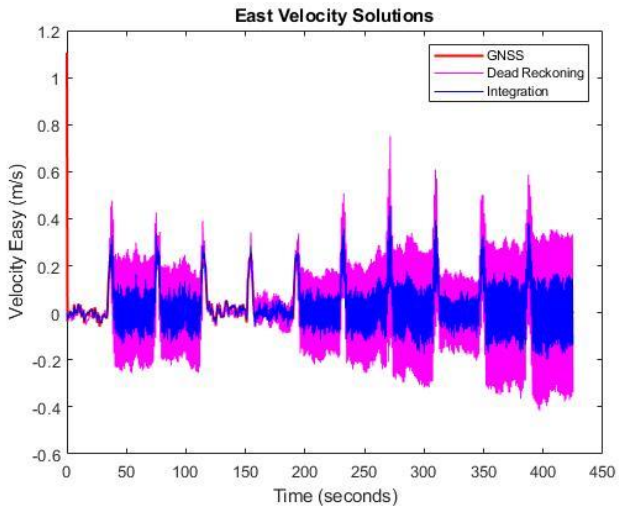

Figure 10: Plots of the GNSS, Dead-Reckoning and Integration solutions for the East Velocity.

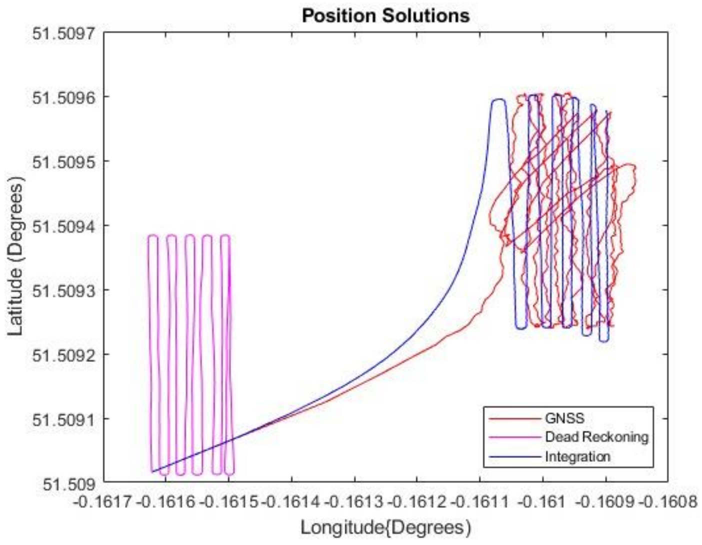

Figure 11: Plots of the GNSS, Dead-Reckoning and Integration position solutions for visual comparison.

As hypothesized in the introduction of the paper, it is clear in figure 10 that the statement from [2] detailing that no single navigation mechanism is reliable enough to show an accurate trajectory is agreeable with our results. It can be seen that there is a large difference in behaviour between the noisy GNSS data and the dead-reckoning. The absolute positions between both mechanisms are very different and the resulting integrated solution takes the advantages of both solutions. Although the lawnmower data was simulated data, the trajectory was plotted on static maps provided by Google's API showing that the lawnmower's starting position was in Hyde Park, London. The lawnmower trajectory was plotted on the map showing mowing of the grass area. It can be seen on Figure 12 that the lawnmower begins on one plot of grass and proceeds to a pedestrian pathway and onto another field. Further, there is a clear issue with the trajectory as during what is assumed to be the mowing cycle for the secondary plot, the lawnmower goes onto the pedestrian pathway again. This shows that there is a clear error in the algorithm, as this would pose a hazard to pedestrians and could damage the lawnmower. It is anticipated that use of an extended and linearized Kalman filter mentioned in [1] could provide a more accurate algorithm as it is considered that the closed-loop correction of the system can overcome sources of error introduced by the linear approximation of the measurement model performed in algorithm 2 [1].

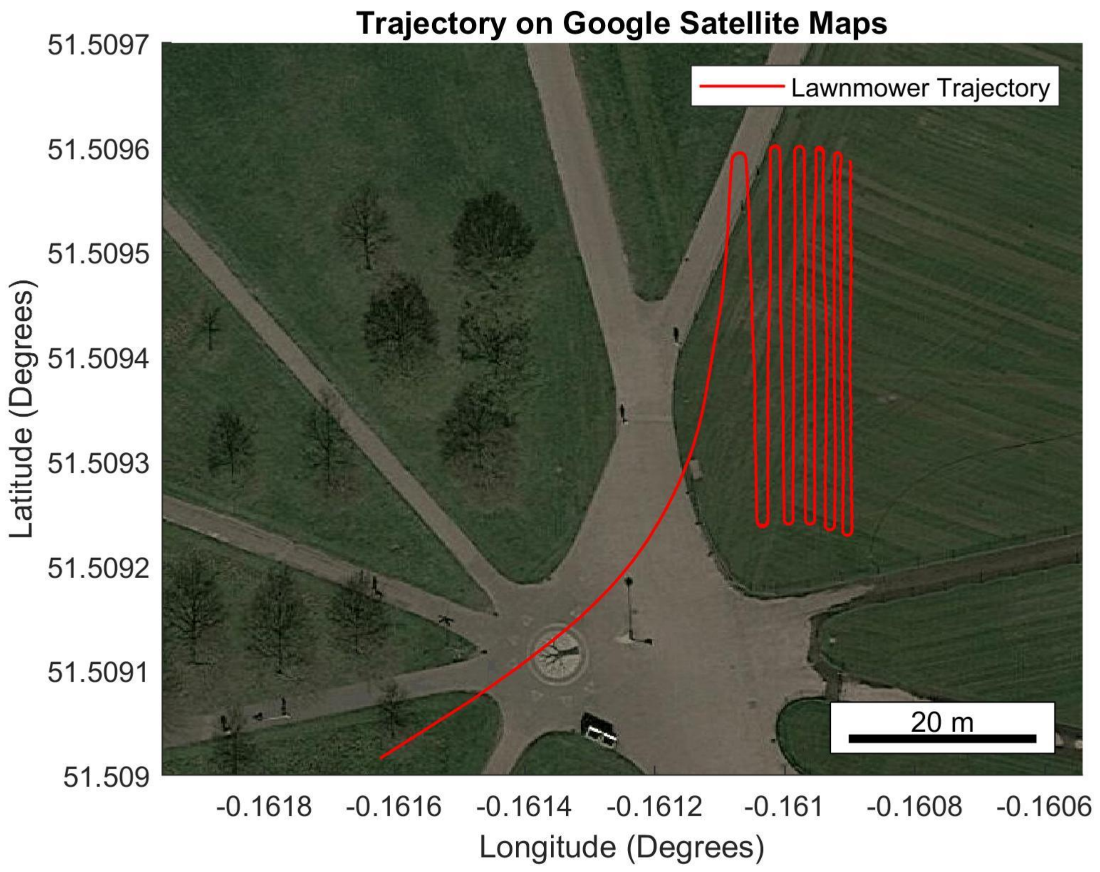

Figure 12: plot of the integrated solution trajectory overlayed on a map provided by Google maps static API.

5. Conclusion

This report aimed to investigate the fusion of outputs from multiple navigation mechanisms and sensors to produce position, velocity, and heading solutions from provided simulated data. The report extensively explained the approaches used, and the associated reasoning for each approach following citations in literature approaches referenced in section 6. Finally, the result of the integrated solution was plotted on a map with the sources of error explained.

6. References

[1] P. D. Groves., Principles of GNSS: Inertial, and Multisensor Integrated Navigation Systems, 2nd ed. Artech House, 2013.
[2] P. D. Groves, L. Wang, D. Walter, H. Martin, K. Voutsis and Z. Jiang, "The four key challenges of advanced multisensor navigation and positioning," 2014 IEEE/ION Position, Location and Navigation Symposium - PLANS 2014, Monterey, CA, 2014, pp. 773-792, doi: 10.1109/PLANS.2014.6851443.
[3] P. D. Groves, "COMP0130 Lecture Notes" 2021, Department of Computer Science, UCL, Jan, 2021.
[4] P. D. Groves, "COMP0130 Workshop 1: Mobile GNSS Positioning using Least-Square Estimation", Department of Computer Science, UCL, Jan, 2021.
[5] P. D. Groves, "COMP0130 Workshop 2: Aircraft Navigation using GNSS and Kalman Filtering", Department of Computer Science, UCL, Jan, 2021.
[6] P. D. Groves, "Step-by-Step Guide to Least-Squares Estimation", COMP0130 Robot Vision and Navigation, Department of Computer Science, UCL, Jan, 2017, pp 9-10.
[7] P. D. Groves, "Chapter 4: Least-Squares and Kalman Filter-Based Estimation", CEGE0044 Data Analysis and COMP0130 Robot Vision and Navigation, Department of Computer Science, UCL, Nov, 2019.
[8] P. D. Groves, "COMP0130 Lecture Notes 3B: Gyro-Magnetometer Integration" 2021, Department of Computer Science, UCL, Jan, 2021.
[9] P. D. Groves, "COMP0130 Workshop 3: Car Dead Reckoning", Department of Computer Science, UCL, Jan, 2021.
[10] P. D. Groves, "COMP0130 Workshop 3: Car DR/GNSS Integration", Department of Computer Science, UCL, Jan, 2021.

A. Appendix

A.1 Calculation GNSS Solution with Outlier Detection

```
%% Computation of GNSS and Kalman filter %%
%% Define the position at all epochs from all data %%
function GNSS_Results = Calculation_GNSS_with_Outlier_detection
%% Define position and velocity from pseudo ranges data
Pseudo_ranges = load('Data_File/Pseudo_ranges.csv'); % Import data from the Pesudo range csv file
Pseudo_range_rate = load('Data_File/Pseudo_range_rates.csv'); % Import data from the Pesudo range rates csv
file
Define_Constants; % Import 'Define Constants m file and this is useful to calculate the results'

% Define the parameters (current_satellite, num_of_satellites)
current_satellite = Pseudo_ranges(1, 2:end); % Means each satellite number
num_of_satellites = size(current_satellite, 2); % Means the number of satellite

% Make an array to save the position and velocity of the satellites at 0 epoch
pos_vel_satellite = zeros(3, num_of_satellites, 2);
for i = 1:num_of_satellites
inform_satellite = current_satellite(1, i); % Define the current satellite number
% Using the Satellite_position_and_velocity.m file to define the position and velocity of the current
satellite
[Pos_satellite, Vel_satellite] = Satellite_position_and_velocity(0, inform_satellite);
pos_vel_satellite(:, i, 1) = Pos_satellite; % Store the position of the current satellite into the array
pos_vel_satellite(:, i, 2) = Vel_satellite; % Store the velocity of the current satellite into the array
end

% From now, we need to define the remaining of the columns what meaning is
% Define the initial state vector estimate(x_0) and error covariance matrix(P)
[state_vector_est_x, e_covariance_m_P] = Initial_GNSS_Position(Pseudo_ranges(2, 2:end),
Pseudo_range_rate(2, 2:end), pos_vel_satellite);

% Let's Define the array of the GNSS_Results format
% Define the number of epoch
num_of_epoch = size(Pseudo_ranges, 1); % Same as the number of times
GNSS_Results = zeros(num_of_epoch - 1, 7); % Design (num_of_epoch x 7) matrix

%% Save all computation results of the satellites at the first epoch %%
%   L_b           latitude (rad)
%   lambda_b      longitude (rad)
%   h_b           height (m)
%   v_eb_n        velocity of body frame w.r.t. ECEF frame, resolved along north, east, and down (m/s) 3x1
column vector
%   x_plus_k      state estimate vector (8x1)matrix [position-x,y,z; velocity-x,y,z; clock-offset; clock-
drift]
[L_b, lambda_b, h_b, v_eb_n] = pv_ECEF_to_NED(state_vector_est_x(1:3), state_vector_est_x(4:6));
% The unit of 1st and 2nd rows is radian, so it should convert from radian to degree by using the
Define_Constant.m file
Latitude = L_b*rad_to_deg; % Latitude (rad -> degrees)
Longtitude = lambda_b*rad_to_deg; % Longitude (rad -> degrees
Heights = h_b; % Height (m)
GNSS_Results(1, 2:5) = [Latitude,Longtitude, v_eb_n(1:2)'];
GNSS_Results(1, 7) = Heights;

% Initialised and save the first position and velocity of the satellites in the array format
% GNSS_Results(1, 2:5) = [Latitude, Longitude, v_eb_n(1:2)'];
% GNSS_Results(1, 7) = Height;
% Explain the format of Final_Solutions          % Explain the format GNSS_Result (same as
Example_Output_Profile)
% rows = epoch, which means each time step       % rows = epoch, which means each time step
% 1st column = Time (s)                          % 1st column = Time (s)
% 2nd column = Latitude (degrees)                % 2nd column = Latitude (degrees)
% 3rd column = Longitude (degrees)               % 3rd column = Longitude (degrees)
% 4th column = Heading (degrees)                 % 4th column = Velocity_x (North) (m/s)
% 5th column = Velocity_x (North) (m/s)          % 5th column = Velocity_y (East) (m/s)
% 6th column = Velocity_y (East) (m/s)           % 6th column = Heading (degrees)
% 7th column = Velocity_z (m/s)                  % 7th column = Height (m)
% Reminder ! -> 7th column Height doesn't need to show, but we just save it
% because we want to check the value is reasonable or not and also the height will use to compute a Dead
Reckoning and Sensor integration later

% From now, define the measured pesudo ranges and pesudo range rates for every epoch
for i = 1:num_of_epoch - 1
times = Pseudo_ranges(i + 1, 1); % Define the times exactly
Pseudo_ranges_data = Pseudo_ranges(i + 1, 2:end); % Define measured pseudo ranges
Pseudo_range_rates_data = Pseudo_range_rate(i + 1, 2:end); % Define measured pseudo range rates

% State the outputs for using the above all parameters
% Compute the updating state estimate vector and error covariance matrix for using Kalman Filter method
[x_plus_k, P_plus_k, Check_Outlier] = Kalman_Filter(state_vector_est_x, e_covariance_m_P,...
Pseudo_ranges_data, Pseudo_range_rates_data,...
current_satellite, times);

%%% Outlier detection should be considered in this computation %%%
% If the computation detects the outliers, recalculate the position at the current epoch without the
measurement that had the largest residual,
% retaining any outlying measurements with smaller residuals. After that, should repeat the outlier
```

```
detection test and keep going on the computation for remaining epoch
if ~isempty(Check_Outlier)
% Check the outlier has contaminated the position solution or not
[New_pseudo_ranges_data,New_pseudo_range_rates_data, New_satellite_number] =
Outlier_Detections(Pseudo_ranges_data, Pseudo_range_rates_data,...
current_satellite, Check_Outlier);
% After checking the outlier detection solutions, recalculate all measurements now past the test
[x_plus_k, P_plus_k, ~] = Kalman_Filter(state_vector_est_x, e_covariance_m_P,...
New_pseudo_ranges_data, New_pseudo_range_rates_data,...
New_satellite_number, times);
end

% Already mentioned these parameters above and the parameters should define again because the computation
is repeated because of outlier detection
[L_b, lambda_b, h_b, v_eb_n] = pv_ECEF_to_NED(x_plus_k(1:3), x_plus_k(4:6));
% The unit of 1st and 2nd rows is radian, so it should convert from radian to degree by using the
Define_Constant.m file
Latitude = L_b * rad_to_deg; % Latitude (rad -> degrees)
Longtitude = lambda_b * rad_to_deg; % Longitude (rad -> degrees)
Heights = h_b; % Height (m)

% Re-initialised and save the first position and velocity of the satellites in the array format
GNSS_Results(i, 1) = times;
GNSS_Results(i,2:5) = [Latitude, Longtitude, v_eb_n(1:2)'];

% 1st(times), 2nd(latitude), 3rd(longitude), 4th(velocity-North), 5th(velocity-East) and 7th(Height)
columns are already defined
% Therefore, remaining column is 6th, which is a heading value. Form now, compute the 6th column(Heading)
Heading = atan(GNSS_Results(i, 5)/ GNSS_Results(i, 4)) .* rad_to_deg;
if GNSS_Results(i, 4) < 0  % Checking the degrees
Heading = Heading + 180;
end

GNSS_Results(i, 6) = Heading; % Save the heading values in the 6th column
GNSS_Results(i, 7) = Heights; % Save the height values in the 7th column
% Final step to update the results of Kalman Filter for computing the next epoch
state_vector_est_x = x_plus_k;
e_covariance_m_P = P_plus_k;
end
end

%% Initialise the Kalman Filter, this computation is followed by Workshop 1 and example of least sqaure
approximation %%
%% State estimate vector and error covariance matrix %%
function [update_state_vector_est_x, update_e_covariance_m_P] = Initial_GNSS_Position(Pseudo_r, Pseudo_r_r,
Pos_Vel_s)
% Compute the linear least square approximation method to initialised the user position
% by using the initial position of the satellite
Define_Constants; % Import 'Define Constants m file and this is useful to calculate the results'
position_satellite = Pos_Vel_s(:, :, 1); % Define the position of satellite by using
Satellite_position_and_velocity.m file
num_of_satellite  = size(position_satellite, 2); % Define the number of satellite
state_vector_est_x = zeros(8, 1); % Define 8x1 matrix for state estimate vector
new_position = zeros(size(position_satellite)); % Define the new position for using Sagnac effect
compensation matrix

% Compute the new position to check the correct position by using the Sagnac effect compensation matrix
for k = 1:num_of_satellite
r_aj = Pseudo_r(k); % Range from the approximation the user position to each satellite
current_position = position_satellite(:, k); % Define the position of the satellites
% Compute the Sagnac effect compensation matrix
C_I_e = [               1    omega_ie*r_aj/c     0;...
-omega_ie*r_aj/c                1     0;...
0                  0     1];
new_position(:, k) = C_I_e * current_position;
end
% After finishing the computation of the Sagnac effect, then compute the new range
new_ranges = Pseudo_r(1);
Total_ranges = Pseudo_r;
Pseudo_r(1) = [];
new_ranges = repmat(new_ranges, [1, num_of_satellite - 1]);

% Also compute the new position
new_positions = new_position(:, 1);
new_positions = repmat(new_positions, [1, num_of_satellite - 1]);
new_position(:, 1) = [];

% Should check the distance between the current satellite of the position and next satellite of the
position
element_1 = (new_position - new_positions)';
cs_ns_distance = sum((new_position(:, :) - new_positions(:, :)).^2, 1);
element_2 = 1/2.*(new_ranges.^2 - Pseudo_r.^2 + cs_ns_distance).';
A = element_1 \ element_2;
state_vector_est_x(1:3) = A + new_positions(:, 1);

% From now, we finished to initialised the position of the state vector [Position-x, Position-y, Position-
z]
% Then, define the velocity of the state vector [Velocity-North, Velocity-East, Velocity-Down]
```

```
state_vector_est_x(4:6) = zeros(3, 1);
% Define the clock offset and clock drift of the state estimate vector
% Uses least-squares to estimate the receiver clock errors before initialise an extended Kalman filter
clock_offset = 100000; % The clock offset (m) would be computed to extend the Kalman filter
clock_drift = 200; % The clock drift (m/s) would be computed to extend the Kalman filter
state_vector_est_x(7:8) = [clock_offset; clock_drift];

% Initialise the state estimate vector when the above computation is finished at the first epoch cycle
update_state_vector_est_x = Calculation_GNSS_Receiver_Clock(state_vector_est_x, Total_ranges, Pos_Vel_s,
Pseudo_r_r);

% Initialise the error covariance matrix at the first epoch cycle and From the coursework 1 instruction,
sigma_pseudo_r = 10; % The noise standard deviation assumes 10 m on all pseudo range measurements
sigma_pseudo_r_r = 0.05; % The noise standard deviation assumes 0.05 m/s on all pseudo range rates
measurements
update_e_covariance_m_P = [sigma_pseudo_r^2 * eye(3)                                   zeros(3, 5);...
zeros(3)   sigma_pseudo_r_r^2 * eye(3)          zeros(3, 2);...
zeros(1, 6)              sigma_pseudo_r^2                  0;...
zeros(1, 7)                          sigma_pseudo_r_r^2];
%     update_e_covariance_m_P = [100      0      0       0      0      0      0      0;     for position-x
%                                  0    100      0       0      0      0      0      0;     for position-y
%                                  0      0    100       0      0      0      0      0;     for position-z
%                                  0      0      0  0.0025      0      0      0      0;     for velocity-x
%                                  0      0      0       0 0.0025      0      0      0;     for velocity-y
%                                  0      0      0       0      0 0.0025      0      0;     for velocity-z
%                                  0      0      0       0      0      0    100      0;     for clock offset
%                                  0      0      0       0      0      0      0 0.0025]    for clock drift
end

%% Initialise the Kalman Filter, this computation is followed by Workshop 1 %%
%% State clock offset, clock drift and compute the update velocity %%
function update_state_vector_est_x = Calculation_GNSS_Receiver_Clock(state_vector_est_x, Total_ranges,
Pos_Vel_s, Pseudo_r_r)
% Define the parameters to compute the results
position_satellite = Pos_Vel_s(:, :, 1); % Define the position of satellite by using
Satellite_position_and_velocity.m file
velocity_satellite = Pos_Vel_s(:, :, 2); % Define the velocity of satellite by using
Satellite_position_and_velocity.m file
num_of_satellite = size(Total_ranges, 2); % Define the number of satellite
Define_Constants; % Import 'Define Constants m file and this is useful to calculate the results

% Check the results to converge to constant values during the computation
while true
%%%%%%%%%%% With respect to position and clock offset %%%%%%%%%
% Compute the clock offset by using the predicted ranges for each satellite
% Make an array of predicted range format
predict_ranges = zeros(1, num_of_satellite);
% Make a for loop and compute the predicted ranges for each statellite
for k = 1:num_of_satellite
current_position = position_satellite(:, k);
distance = current_position - state_vector_est_x(1:3);
% Predict the ranges from the approximatie user position to each satellite
r_aj = sqrt(distance.' * distance);
% Compute the Sagnac effect compensation matrix
C_I_e = [                   1     omega_ie * r_aj / c     0;...
-omega_ie * r_aj /c                     1     0;...
0                      0     1];
% Re-compute the predicted range
new_distance = C_I_e * current_position - state_vector_est_x(1:3);
predict_ranges(:, k) = sqrt(new_distance.' * new_distance);
end

% After finishing the computation of the predict ranges from the approximate user position to each
satellite,
% then need to compute the line-of-sight unit vector from the approximate user position to each satellite
u_e_aj = (position_satellite - repmat(state_vector_est_x(1:3), [1,
num_of_satellite]))./repmat(predict_ranges, [3,1]);
% easy method for using approximation, but this result slightly different results
% so we are not using this equation. However we write this equation to help understanding
% u_e_aj = (current_position - state_vector_est_x(1:3)) / position_satellite;

% Formulate the predicted state vector, measurement innovation vector (delta_z), and measurement matrix
(H_e_G)
% Compute the measurement innovation vector (delta_z)
% Input = Total_ranges: the measured pseudo range from each satellite to the user antenna
%         predict_ranges: Predict the ranges from the approximate user position to each satellite
%         delta_rho_a_c: Predicted receiver clock offset
delta_rho_a_c = ones(1, num_of_satellite) .* state_vector_est_x(4);
delta_z = Total_ranges - predict_ranges - delta_rho_a_c;

% Compute the measurement matrix (H_e_G)
H_e_G = [-u_e_aj.'    ones(num_of_satellite, 1)]; % Concatenates the line-of-sight matrix with ones(8x1)
matrix

% After finishing the compuation of the measurement innovation vector (delta_z) and matrix (H_e_G),
% then compute the position and receiver clock offset using unweighted least sqaures
x_plus_pos = state_vector_est_x(1:4) + (H_e_G.' * H_e_G)^-1 * H_e_G.' * delta_z.';
```

```
%%%%%%%%%%% With respect to velocity and clock drift %%%%%%%%%
% Compute the clock drift by using the predicted ranges for each satellite
% Make an array of predicted range rates format
predict_range_rates= zeros(1, num_of_satellite);
% Make a for loop and compute the predicted ranges for each statellite
for k = 1:num_of_satellite
r_aj = Total_ranges(1, k);
current_position = position_satellite(:, k);
current_velocity = velocity_satellite(:, k);

% Compute the predicted range rates (r_aj_dot)
% Following the equation (9) in Workshop 1,
element_1 = current_velocity + Omega_ie * current_position;
element_2 = state_vector_est_x(5:7) + Omega_ie * state_vector_est_x(1:3);
u_e_aj_vel = u_e_aj(:, k).';
% Compute the Sagnac effect compensation matrix for the velocity
C_I_e = [                  1    omega_ie * r_aj / c    0;...
-omega_ie * r_aj / c                         1    0;...
0                      0    1];
predict_range_rates(1, k) = u_e_aj_vel * (C_I_e * element_1 - element_2);
end

% Formulate the predicted state vector, measurement innovation vector (delta_z), and measurement matrix
(H_e_G)
% Compute the measurement innovation vector (delta_z)
% Input = Total_ranges: the measured pseudo range from each satellite to the user antenna
%         predict_ranges: Predict the ranges from the approximate user position to each satellite
%         delta_rho_a_c: Predicted receiver clock offset
rho_j_a_dot = Pseudo_r_r;
delta_rho_a_c_dot = ones(1, num_of_satellite) * state_vector_est_x(8);
delta_z_vel = rho_j_a_dot - predict_range_rates - delta_rho_a_c_dot;

% Compute the measurement matrix (H_e_G) for velocity
H_e_G_vel = [-u_e_aj.'    ones(num_of_satellite, 1)] ; % Concatenates the line-of-sight matrix with
ones(8x1) matrix

% Update the state estimate vector
x_plus_vel = state_vector_est_x(5:8) + (H_e_G_vel.' * H_e_G_vel)^-1 * H_e_G_vel.' * delta_z_vel.';

% Update the state estimate vector
new_state_vector_est_x = [x_plus_pos;...
x_plus_vel];  % Make a 8x1 vector

% Check the difference bewteen old version of the state vector and new version of the state vector
%converge_test = sqrt(sum((state_vector_est_x - new_state_vector_est_x).^2, 1));
converge_test = norm(state_vector_est_x - new_state_vector_est_x);
% If the value is bigger than 0.0001, keep runing the while loop to make convergence until less than 0.0001
% If the value is less than 0,0001, stop and update the new state estimate vector
if converge_test >= 0.01  % Reduce the error
state_vector_est_x = new_state_vector_est_x;
else
break;
end
end
% Final update and use these vector (8x1)
update_state_vector_est_x = [x_plus_pos;...
x_plus_vel];
end

%% We finished to compute the Initialised Kalman Filter at the 1st epoch %%
%% Now the Kalman Filter at all epochs should be computed to get position and velocity %%
function [x_plus_k, P_plus_k, Outlier_Detection] = Kalman_Filter(state_vector_est_x, e_covariance_m_P,...
New_pseudo_ranges_data, New_pseudo_range_rates_data,...
New_satellite_number, times)
% Define the parameters to compute the results
Define_Constants; % Import 'Define Constants m file and this is useful to calculate the results'

% Compute the every epoch (time) and Following the workshop 2 Task 2A
%% Kalman Filter Step 1 %%
% Compute the transition matrix (input: Identity matrix and tau_s)
tau_s = 0.5; % Propagation interval tau_s(s)
transition_m = [    eye(3) tau_s * 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       tau_s;...
zeros(1, 3)     zeros(1, 3)            0           1];

%% Kalman Filter Step 2 %%
% Compute the system noise covariance matrix
% (input: the acceleration power spectral density (PSD) S_e_a = 3 (m^2/s^3)
S_e_a = 5 ; % Value of the acceleration for a pedestrian as workshop 2 and this accounts for the
propagation of the system noise onto the position state during the propagation interval
S_a_cphi = 0.01; % Clock phase PSD for the GNSS receiver clock
S_a_cf =  0.04; % Clock frequency PSD for the GNSS receiver clock
Q_k = [(S_e_a * tau_s^3 * eye(3))/3  (S_e_a * tau_s^2 * eye(3))/2                          zeros(3, 1)
zeros(3, 1);...
(S_e_a * tau_s^2 * eye(3))/2      (S_e_a * tau_s * eye(3))                           zeros(3, 1)
zeros(3, 1);...
zeros(1, 3)                  zeros(1, 3)  (S_a_cphi * tau_s + (S_a_cf * tau_s^3)/3)  (S_a_cf *
tau_s^2)/2;...
zeros(1, 3)                   zeros(1, 3)                      (S_a_cf * tau_s^2)/2        S_a_cf * tau_s];

%% Kalman Filter Step 3 %%
% Propagate the state estimate
x_minus_k = transition_m * state_vector_est_x;

%% Kalman Filter Step 4 %%
% Propagate the error covariance matrix
P_minus_k =  transition_m * e_covariance_m_P * transition_m.' + Q_k;

% After finishing the computation of the state estimate vector and the error covariance matrix,
% then need to compute the line-of-sight unit vector from the approximate user position to each satellite
% (Following the Workshop 2 Task 2A - f) and g))
num_of_satellite = size(New_satellite_number, 2);
Pos_Vel_s = zeros(3, num_of_satellite, 2);
% Compute the position and velocity of each satellite at the current epoch
for k = 1:num_of_satellite
current_satellite = New_satellite_number(1, k); % Define the current satellite
%   position_satellite       ECEF satellite position (m) vector
%   sat_v_es_e               ECEF satellite velocity (m/s) vector
[position_satellite, sat_v_es_e] = Satellite_position_and_velocity(times, current_satellite);
Pos_Vel_s(:, k, 1) = position_satellite;
Pos_Vel_s(:, k, 2) = sat_v_es_e;
end

% Compute the predicted range at the current epoch
predict_ranges = zeros(1, num_of_satellite);
for k = 1:num_of_satellite
% Initialised the predicted range
current_position = Pos_Vel_s(:, k, 1);
distance = current_position - x_minus_k(1:3, :);
r_aj = sqrt(distance.' * distance);

% Compute the Sagnac effect compensation matrix
C_I_e = [                  1      omega_ie * r_aj/c    0;...
-omega_ie * r_aj/c                     1    0;...
0                      0    1];
% Re-compute the predicted range
new_distance = C_I_e * current_position - x_minus_k(1:3, :);
predict_ranges(:, k) = sqrt(new_distance.' * new_distance);
end
% From now, compute the line of sight unit vector from the approximate user position to each satellite
u_e_aj = (Pos_Vel_s(:, :, 1) - repmat(x_minus_k(1:3, :), [1, num_of_satellite]))./repmat(predict_ranges,
[3, 1]);

%% Kalman Filter Step 5 %%
% Compute the measurement matrix
H_k = [                 -u_e_aj.'  zeros(num_of_satellite, 3)   ones(num_of_satellite, 1)
zeros(num_of_satellite, 1);...
zeros(num_of_satellite, 3)                   -u_e_aj.'  zeros(num_of_satellite, 1)   ones(num_of_satellite,
1)];

%% Kalman Filter Step 6 %%
% Compute the measurement noise covariance matrix
sigma_p = 2; % Code tracking and multipath error standard deviation for pseudo range measurement (m)
sigma_r = 0.02; % Range rate tracking and multipath error standard deviation for pseudo range rate
measurement (m/s)
R_k = [sigma_p^2 * eye((num_of_satellite),(num_of_satellite))                 zeros(num_of_satellite,
num_of_satellite);...
zeros(num_of_satellite, num_of_satellite)     sigma_r^2 * eye(num_of_satellite, num_of_satellite)];
R_k = R_k + [zeros(num_of_satellite, 2 * (num_of_satellite));...
zeros(num_of_satellite,  2 * (num_of_satellite) - 1)   ones(num_of_satellite, 1)];
R_k(2 * (num_of_satellite), 2 * (num_of_satellite)) = 0;%R_k(2 * (num_of_satellite), 2 *
(num_of_satellite)) - 1;
% This method for computing the measurement error covariance follows
% the COMP0130 Least Sqaure Step by Step.pdf

%% Kalman Filter Step 7 %%
% Compute the Kalman gain matrix
K_k = P_minus_k * H_k.' * (H_k * P_minus_k * H_k.' + R_k)^-1;

% Compute the predict range rate for the velocity
r_aj_dot = zeros(1, num_of_satellite);
for k = 1:num_of_satellite
% Define the position and velocity for each satellite for using the Satellite_position_and_velocity.m file
position_satellite = Pos_Vel_s(:, k, 1);
velocity_satellite = Pos_Vel_s(:, k, 2);

% Define the propagated state vector for the position and velocity for each satellite
r_e_ea = x_minus_k(1:3, :); % Predicted Cartesian ECEF user position
v_e_ea = x_minus_k(4:6, :); % Predicted Cartesian ECEF user velocity

% Predict the range rates from the approximate user position to each satellite
r_aj = predict_ranges(1, k);

% Define the line-of-sight unit vector from the approximate user position to each satellite
element_1 = velocity_satellite + Omega_ie * position_satellite;
```

```
element_2 = v_e_ea + Omega_ie * r_e_ea;
LoS = u_e_aj(:, k).'; % LoS means the line-of-sight
C_I_e = [                   1  omega_ie * r_aj / c    0;...
-omega_ie * r_aj / c                    1    0;...
0                    0    1];
r_aj_dot(1, k) = LoS * (C_I_e * element_1 - element_2);
end

%% Kalman Filter Step 8 %%
% Formulate the measurement innovation vector
delta_z = zeros(1, 2 * (num_of_satellite)); % Make an array size of the measurement innovation vector
delta_rho_a_c_dot = ones(1, num_of_satellite) * x_minus_k(end - 1);
delta_rho_a_c_dot_again = ones(1, num_of_satellite) * x_minus_k(end);
delta_z(1, 1:num_of_satellite) = New_pseudo_ranges_data - predict_ranges - delta_rho_a_c_dot;
delta_z(1, num_of_satellite + 1:end) = New_pseudo_range_rates_data - r_aj_dot - delta_rho_a_c_dot_again;

%% Kalman Filter Step 9 %%
% Update the state estimates
x_plus_k = x_minus_k + K_k * delta_z.';

%% Kalman Filter Step 10 %%
% Update the error covariance matrix
P_plus_k = (eye(size(K_k * H_k))- K_k * H_k) * P_minus_k;

%% Finished the computation of Kalman Filter at the epoch %%
%% Check the residual based outlier detection for the position at the epoch %%
% Define inputs: measurement matrix (H_e_G_outlier), measurement innovation matrix (delta_z_outlier),...
%                ionosphere error (sigma_iono), troposhpere error (sigma_tropo),
%                outlier detection threshold (T)
sigma_iono = 2; % Define the residual ionosphere error standard deviation at zenith
sigma_tropo = 0.2; % Define the residual troposhpere error standard deviation at zenith
T = 6; % Define the outlier detection threshold, which is based on the workshop 1 Task 3 - (c)
H_e_G_outlier = [H_k(1:num_of_satellite , 1:3), ones(num_of_satellite, 1)] ;
delta_z_outlier = delta_z(1, 1:num_of_satellite) ;

%% Outlier Detection Step 1 %%
% Compute the residuals vector
v = (H_e_G_outlier * ((H_e_G_outlier.' * H_e_G_outlier)^-1) * H_e_G_outlier.' - eye(num_of_satellite)) *
delta_z_outlier.';

% Consider for the position (x,y,z) in line-of-sight unit vector, which means latitude and longtitude,
respectively, so we need to measure both error at Zenith
% The angle between the user and satellite can compute the sine function sin(z_e_as, r_as)
SD_zenith = repmat((sigma_iono + sigma_tropo), 1, num_of_satellite); % Total residual error for the
position is sum with ionosphere and troposphere
Angle = u_e_aj(3, :)./r_aj;
SD_measurement_error = SD_zenith./sin(Angle); % Reference by Lecture 1D Slide pg 21
% Compute the measurement error variance = (Measurement error of standard deviation)^2
e_variance = (SD_measurement_error).^2;

%% Outlier Detection Step 2 %%
% Compute the residuals covariance matrix
C_v = eye(num_of_satellite) - H_e_G_outlier * (H_e_G_outlier.' * H_e_G_outlier)^-1 * H_e_G_outlier.';

%% Outlier Detection Step 3 %%
% Compute the normalised residuals and compare each with a threshold
C_v_jj = diag(C_v)' .* e_variance; % Define the diagonal element of the residuals covariance matrix
w_i = v' ./ sqrt(C_v_jj); % Normalised the residual, based on Least Sqaure Step-by-Step.pdf (Section:
Measurement error hypothesis testing)

%% Outlier Detection Step 4 %%
% Check the condition when the measurement j is an outlier with Threshold
check_condition = abs(w_i) > T;  % same as abs(v) > sqrt(C_v_jj).* T;
outlier = abs(w_i) - T;
if outlier > 0 % If the condition of the outlier detection is satisfied,
Outlier_Detection = check_condition; % Save the values and keep going to compute for the next epoch
else                           % If not, the outlier has contaminated the position solution and have the
largest residual, so
Outlier_Detection = []; % Remove the measurement and repeat the emasurement to find that all measurements
end
end

%% Handle the outlier detection %%
%% After Computing the Kalman Filter, if the measurement had the largest residual, then make empty array to
recalculate the position at the epoch %%
function [update_r, update_r_r, update_satellite] = Outlier_Detections(previous_r, previous_r_r,
previous_satellite, Check_Outlier)
% Errors in individual measurements or their measurement models can be identified from the residuals
% Define the predicted ranges
update_satellite = previous_satellite;
update_r = previous_r;
update_r_r = previous_r_r;
num_of_satellites = size(previous_satellite, 2);

occur_outlier = 0; % Define the initial Outlier detection
% Removing the measurement from the outlier detection of the satellite
% and changes the range data to compute for solving right results
for k = 1:num_of_satellites
```

```
if Check_Outlier(k) == 1 % If the outlier detection occurs, then use this loop
occur_outlier = occur_outlier + 1; % Comfirm the detection outliers on some of the other measurements
update_satellite(k - occur_outlier) = []; % Found the outlier detection of the satellite number, which has
the largest residual
update_r_r(k - occur_outlier) = []; % Save new range rate values and remove the measurement and repeat the
emasurement to find a new range rate
update_r(k - occur_outlier) = []; % Remove the measurement and repeat the mesurement to find a new range
end
end

end
```

A.2 Calculation Dead Reckoning

```
%%% Computation of Dead Reckonging and Integration of Gyro-Magnetometer %%%
%%% Define the position at all epochs from all data %%%
function Dead_Reckoning_Result = Calculation_Dead_Reckoning
Define_Constants  % Import 'Define Constants m file and this is useful to calculate the results
Dead_reckoning = csvread('Data_File\Dead_reckoning.csv');  % Import data from the Dead reckoning csv file
[i, ~] = size(Dead_reckoning); % i means epoch
% Make array format to save the results after finishing the computation

% Explain the format Dead_Reckoning_Result
% rows = epoch, which means time step
% 1st column = Time (s)
% 2nd column = Latitude (degrees)
% 3rd column = Longitude (degrees)
% 4th column = Velocity_x (North) (m/s)
% 5th column = Velocity_y (East) (m/s)
% 6th column = Heading (degrees)
Dead_Reckoning_Result = zeros(i, 6); % row = each epoch (times) and column = same format of the example
output profile

% Let's define the results profile for time(s) because it's same as the data
Dead_Reckoning_Result(:, 1) = Dead_reckoning(:, 1); % 1st column = Time(s) from Dead reckoning data
time = Dead_reckoning(:, 1);
% Define the initial parameters from the Calculation_GNSS function
initial_para = Calculation_GNSS_with_Outlier_detection;
%% Define several useful parameters

% Define driving speed for the robotic lawnmower
% Since it has four wheeels and the rear wheels are the driving wheels,
% we denote the average speed of rear wheels as the driving speed
driving_speed = 0.5*(Dead_reckoning(:,4)+Dead_reckoning(:, 5));

% Define gyroscope angular rate
gyroscope_angular_rate = Dead_reckoning(:,6);

% Define heading measurements in degrees from the magnetic compass
magnetic_heading = Dead_reckoning(:,7);

%% Integrate Gyro-Magnetometer to get the corrected heading
% First, we calculate gyro-heading from angular rate

% Initalize gyro-heading
gyro_heading = zeros(i,1);
% Define initial gyro-heading: when we look at the Dead reckoning csv file we
% find that the initial gyroscope angular rate is 0. Thus, the initial
% gyro-heading is 0
gyro_heading(1) = 0;
% Find total gyro-heading on each time epoch: compute the integral of
% angular rate in the horizontal plane.
for j = 2:i
    % time interval is 0.5 based on the Dead reckoning csv file
    gyro_heading(j) = gyro_heading(j-1)+0.5*gyroscope_angular_rate(j);
end

% Then we find the integration of Gyro-Magnetometer
% First, define several parameters
% Convert deg to rad or rad to deg using Define_Constants
bias_std = 1*deg_to_rad; % A bias standard deviation of 1 degree per second
magnet_error_std = 4*deg_to_rad; % a noise-like error with a standard deviation of 4 degrees
magnetic_heading = magnetic_heading*deg_to_rad; % convert to radian
noise_std = 10^-4;% A noise standard deviation of 10-4 radians per second
S_rg = 3*10^-6; % Gyro random noise with power spectral density
S_bgd = 3*10^-6; % Gyro bias variation with PSD
tau_s = 0.5; % propagation interval = 0.5
P_k_minus1 = [noise_std^2 0; 0 bias_std^2];
x_k_minus1 = [0;0];
% initialize our result corrected heading
corrected_heading = zeros(i,1);
% Since there is no initial gyro-heading, we store the first magnetic
% heading as our first corrected heading.
corrected_heading(1) = magnetic_heading(1);
% Then follow the ten steps of the Kalman filter as follows:
for k=2:i
    % Compute the transition matrix
    phi_k_minus1 = [1 tau_s; 0 1];
    % Compute the system noise covariance matrix
    Q_k_minus1 = [S_rg*tau_s+1/3*S_bgd*tau_s^3 1/2*S_bgd*tau_s^2;...
        1/2*S_bgd*tau_s^2 S_bgd*tau_s];
    % Propagate the state estimates
    x_k_minus = phi_k_minus1*x_k_minus1;
    % Propagate the error covariance matrix
    P_k_minus = phi_k_minus1*P_k_minus1*phi_k_minus1'+Q_k_minus1;
    % Compute the measurement matrix
    H_k = [-1 0];
    % Compute the measurement noise covariance matrix
    R_k = [magnet_error_std^2];
    % Compute the Kalman gain matrix
    K_k = P_k_minus*H_k'/(H_k*P_k_minus*H_k'+R_k);
    % Formulate the measurement innovation vector
    delta_z_k_minus = [magnetic_heading(k) - gyro_heading(k)] - H_k*x_k_minus;
    % Update the state estimates
    x_k_plus = x_k_minus+K_k*delta_z_k_minus;
    % Update the error covariance matrix
    P_k_plus = (eye(2) - K_k*H_k)*P_k_minus;

    % store the value in corrected_heading
    corrected_heading(k) = gyro_heading(k)-x_k_plus(1);

    % update parameters
    P_k_minus1 = P_k_plus;
    x_k_minus1 = x_k_plus;
end

% store corrected heading in degrees to Dead_Reckoning_Result
Dead_Reckoning_Result(:,6) = corrected_heading*rad_to_deg;
%% Use corrected heading to find the dead Reckoning solution
% Define geodetic height for dead reckoning
height = initial_para(:,7);
% Define initial position from the Calculation_GNSS function
Dead_Reckoning_Result(1,2) = initial_para(1,2); % initial Geodetic latitude
Dead_Reckoning_Result(1,3) = initial_para(1,3); % initial Geodetic longitude
% Define initial velocity: when we look at the Dead reckoning csv file we
% find that the initial speed is 0. Thus, the initial velocity is 0.
Dead_Reckoning_Result(1,4) = 0; % initial North velocity
Dead_Reckoning_Result(1,5) = 0; % initial East velocity
% Find geodetic latitude, longitude and North and East velocity
for k = 2:i
    % The average velocity between epochs k?1 and k
    v_N_k = 1/2*(cos(corrected_heading(k))+cos(corrected_heading(k-1)))*...
        driving_speed(k);
    v_E_k = 1/2*(sin(corrected_heading(k))+sin(corrected_heading(k-1)))*...
        driving_speed(k);

    % The radii of curvature may be computed from the latitude using the Matlab
    % function Radii_of_curvature
    [R_N,R_E]= Radii_of_curvature(Dead_Reckoning_Result(k-1,2));
    % Update the latitude, L_k(Dead_Reckoning_Result(:,2)),
    % and tue longitude, lambda_k(Dead_Reckoning_Result(:,3))
    Dead_Reckoning_Result(k,2) = Dead_Reckoning_Result(k-1,2) + (v_N_k*0.5)...
       /(R_N+height(k));
    Dead_Reckoning_Result(k,3) = Dead_Reckoning_Result(k-1,3) + (v_E_k*0.5)...
       /((R_E+height(k))*cos(Dead_Reckoning_Result(k,2)));
    % Update the North velocity(Dead_Reckoning_Result(:,4)) and
    % East velocity(Dead_Reckoning_Result(:,5)) without damping
    Dead_Reckoning_Result(k,4) = 2*v_N_k - Dead_Reckoning_Result(k-1,4);
    Dead_Reckoning_Result(k,5) = 2*v_E_k - Dead_Reckoning_Result(k-1,5);
end
% convert latitude and longitude to degree
csvwrite('Dead_Reckoning_computation.csv',Dead_Reckoning_Result)

end
```

A.3 Calculation Integration Navigation

```
%%% Computation of Integration of Dead Reckoning and GNSS %%%
%%% Define the position at all epochs from all data %%%
function Integration_Result = Calculation_Integration
Define_Constants  % Import 'Define Constants m file and this is useful to calculate the results
Dead_reckoning = csvread('Data_File\Dead_reckoning.csv');  % Import data from the Dead reckoning csv file
[i, ~] = size(Dead_reckoning); % i means epoch

% Explain the format Integration_Result
% rows = epoch, which means time step
% 1st column = Time (s)
% 2nd column = Latitude (degrees)
% 3rd column = Longitude (degrees)
% 4th column = Velocity_x (North) (m/s)
% 5th column = Velocity_y (East) (m/s)
% 6th column = Heading (degrees)
Integration_Result = zeros(i, 6); % row = each epoch (times) and column = same format of the example output
profile

% Define the initial parameters from the Calculation_GNSS function
initial_para_GNSS = Calculation_GNSS_with_Outlier_detection;

% Define the initial parameters from the Calculation_Dead_Reckoning function
initial_para_Dead_Reckoning = Calculation_Dead_Reckoning;

% Store time in column 1
Integration_Result(:, 1) = initial_para_GNSS(:, 1);
Integration_Result(:, 6) = initial_para_Dead_Reckoning(:, 6);
% First, define several parameters
S_DR = 0.01; % velocity error variance with PSD
tau_s = 0.5; % propagation interval = 0.5
sigma_v = 0.1; % initial velocity uncertainty is 0.1m/s in each direction
sigma_r = 10; % initial position uncertainty is 10m per direction
sigma_Gr = 10; % GNSS poistion noise std on all pseudo-range measurements
sigma_Gv = 0.05; % GNSS velocity noise std on all on all pseudo-range rate measurements
height = initial_para_GNSS(:,7); % height computed from GNSS
Latitude = initial_para_Dead_Reckoning(:,2)*deg_to_rad; % latitude computed from Dead reckoning
% The radii of curvature may be computed from the latitude using the Matlab
% function Radii_of_curvature
[R_N_initial,R_E_initial]= Radii_of_curvature(Latitude(1));
P_k_minus1 = [sigma_v^2 0 0 0;...
             0 sigma_v^2 0 0;...
             0 0 sigma_r^2/(R_N_initial+height(1))^2 0;...
             0 0 0 sigma_r^2/((R_E_initial+height(1))^2)*cos(Latitude(1))^2)];
% implement a 4-state Kalman filter estimating north and east DR velocity error,
% DR latitude error and DR longitude error
x_k_minus1 = [0;0;0;0];
% Initialize the integration result with Dead Reckoning result
Integration_Result(1,:) = initial_para_Dead_Reckoning(1,:);
% Then follow the ten steps of the Kalman filter as follows:
for k=2:i
    % Compute the transition matrix
    [R_N,R_E]= Radii_of_curvature(Latitude(k));
    phi_k_minus1 = [1 0 0 0;...
                    0 1 0 0;...
                    tau_s/(R_N+height(k-1)) 0 1 0;...
                    0 tau_s/((R_E+height(k-1))*cos(Latitude(k-1))) 0 1];
    % Compute the system noise covariance matrix
    Q_k_minus1 = [S_DR*tau_s 0 1/2*S_DR*tau_s^2/(R_N+height(k-1)) 0;...
                  0 S_DR*tau_s 0 1/2*S_DR*tau_s^2/((R_E+height(k-1))*cos(Latitude(k-1)));...
                  1/2*S_DR*tau_s^2/(R_N+height(k-1)) 0 1/3*S_DR*tau_s^3/(R_N+height(k-1))^2 0;...
                  0 1/2*S_DR*tau_s^2/((R_E+height(k-1))*cos(Latitude(k-1))) 0 ...
                  1/3*S_DR*tau_s^3/((R_E+height(k-1))^2*cos(Latitude(k-1))^2)];
    % Propagate the state estimates
    x_k_minus = phi_k_minus1*x_k_minus1;
    % Propagate the error covariance matrix
    P_k_minus = phi_k_minus1*P_k_minus1*phi_k_minus1'+Q_k_minus1;
    % Compute the measurement matrix
    H_k = [0 0 -1 0;...
           0 0 0 -1;...
           -1 0 0 0;...
           0 -1 0 0];
    % Compute the measurement noise covariance matrix
    R_k = [sigma_Gr^2/(R_N+height(k))^2 0 0 0;...
           0 sigma_Gr^2/((R_E+height(k))^2*cos(Latitude(k))^2) 0 0;...
           0 0 sigma_Gv^2 0;...
           0 0 0 sigma_Gv];
    % Compute the Kalman gain matrix
    K_k = P_k_minus*H_k'/(H_k*P_k_minus*H_k'+R_k);
    % Formulate the measurement innovation vector
    delta_z_k_minus = [initial_para_GNSS(k,2)*deg_to_rad - initial_para_Dead_Reckoning(k,2)*deg_to_rad;...
                       initial_para_GNSS(k,3)*deg_to_rad - initial_para_Dead_Reckoning(k,3)*deg_to_rad;...
                       initial_para_GNSS(k,4) - initial_para_Dead_Reckoning(k,4);...
                       initial_para_GNSS(k,5) - initial_para_Dead_Reckoning(k,5)]...
                     - H_k*x_k_minus;
    % Update the state estimates
    x_k_plus = x_k_minus+K_k*delta_z_k_minus;
    % Update the error covariance matrix
    P_k_plus = (eye(4,4) - K_k*H_k)*P_k_minus;

    % store the value in integration result
    % Geodetic latitude
    Integration_Result(k,2) = initial_para_Dead_Reckoning(k,2)*deg_to_rad - x_k_plus(3);
    % Geodetic longitude
    Integration_Result(k,3) = initial_para_Dead_Reckoning(k,3)*deg_to_rad - x_k_plus(4);
    % North velocity
    Integration_Result(k,4) = initial_para_Dead_Reckoning(k,4) - x_k_plus(1);
    % East velocity
    Integration_Result(k,5) = initial_para_Dead_Reckoning(k,5) - x_k_plus(2);

    % update parameters
    P_k_minus1 = P_k_plus;
    x_k_minus1 = x_k_plus;
end
Integration_Result(2:end,2) = Integration_Result(2:end,2)*rad_to_deg;
Integration_Result(2:end,3) = Integration_Result(2:end,3)*rad_to_deg;
csvwrite('Integration_computation.csv',Integration_Result)
end
```