COMP0130-Coursework 1
Integrated Navigation for a Robotic Lawnmower

chujie zhang,19066782,chujie.zhang.19@ucl.ac.uk
yitao jin,19030781,yitao.jin.19@ucl.ac.uk

February 2020

1 Brief description

The first method We choose is GNSS. We calculated the positions and velocities by this method given the two csv files which are *Pseudo_ranges.csv* and *Pseudo_range_rates.csv*. The second method we choose is dead reckoning. We use this method to compute positions and velocities of Lawnmower given that the csv files which is *Dead_reckoning.csv*. The first two methods are using to compare to check out whether we make some large bugs. The third method is to use these two results to compute an integrated horizontal-only DR and GNSS solution using Kalman filter. And for the heading, we choose the method is Gyro-Magnetometer Integration, it is to use the magnetic heading to correct heading from gyroscope with a 2-state Kalman filter. And for the outlier detection, it was used during the GNSS solution, firstly we use this method in GNSS multi-epoch positioning and velocity to find all outliers in each epoch, then remove these outliers in GNSS Kalman Filter at all epochs.All the algorithms we use to compute these methods from lecture notes and workshops. The more details will be given in the next section.

2 Full description of the algorithms

2.1 Initial positions

The first algorithm is to compute the initial positions of lawnmower using least square estimation. This initial positions will be used for GNSS and dead reckoning. And this algorithm is from workshop 1 task 1b[1].
step a,set initial guess to current position $r_{ea}^{e-} = (0, 0, 0)$
step b, compute the cartesian ECEF positions of the satellites at time 0 using the matlab function *satellite_position_and_velocity.m*.
while error < 0.1

- step c, predict the ranges from the approximate user position to each satellite using equation (1)of [1] and compute the sagnac effect compensation matrix using equation(2) of [1] .where r_{aj} is the ranges from the approximate user position, r_{ej} is the Cartesian ECEF position of satellite j and r_{ea} is the initial guess position as I said in step a. $C_e^I(r_{aj})$ is the sagnac effect.W_{ie} is the Earth rotation rate compensation matrix and c is the speed of light.

- step d, compute the line-of-sight unit vector from the approximate user position to each satellite using equation(3) of [1]. where $u_{aj}^e$is the line-of-sight unit vector

- step e, formulate the predicted state vector x^-, measurement innovation vector dz, and measurement matrix H using the equation (4) of [1]. where p_a^j is the measured pseudo-range from satellite j to the user antenna and δp_c^{a-} is the predicted receiver clock offset.

- step f, compute the position and receiver clock offset using equation(5) of [1]. where $r_{ea}^{e+}$is the position and δp_c^{a+} is the receiver clock offset.

- finally, $error = abs(norm(r_{ea}^{e+}) - norm(r_{ea}^{e-}))$. and update variable $r_{ea}^{e-} = r_{ea}^{e+}$.

end while
This algorithm will stop when the changes of predicted positions is less 0.1, then it converges and we find a initial position of lawnmower.

2.2 GNSS Multi-epoch Positioning and velocity

The second algorithm is to use the same method from the first algorithm to compute the position and velocity at all of the epochs. Meanwhile, we will implement the outlier dection at each epoch and save the result in an array. And this algorithm is from task 2,task 3 and task 4 in workshop1[1].
step a,from the first algorithm, We could get an initial position r_{ea}^{e-}, and set initial velocity $v_{ea}^{e-} = (0, 0)$
Then implement the same method(least-square estimation) in the first algorithm from step b to step f to compute the positions without while loop. The difference is to compute velocity. Here are more details.

- Firstly is to compute the predicted range rates using equation(9) of [1]. where v_{ej}^- is the Cartesian ECEF velocity of satellite j, V_{ea}^{e-} is the predicted Cartesian ECEF user velocity and the skew symmetrix matrix Ω_{ie}^e is computed by using the equation (10) of[1].

- Secondly, The predicted state vector x^-, measurement innovation vector dz and measurement matrix H could be calculated by using the equation(11) of[1]. Where p_a^j is the measured pseudo-range from satellite j to hte user antenna and δp_c^{a-} is the predicted receiver clock offset.

- Finally, The velocity and receiver clock drift solution could be computed using the equation(12) of[1].

The algorithm to detect outlier will be given below in 2.4

2.3 GNSS Kalman Filter Multiple Epochs

This algorithm is from task2b in workshop 2[2] with a 8-state kalman filter.

- initialise the kalman filter state vector x_{est} and error covariance matrix p_{matrix} using the matlab function

Then for each epoch:

- read the outlier lists from GNSS Multi-epoch positioning and velocity and then remove the satellie at current epoch

- step 1, compute the transition matrix ϕ_{k-1} using the equation below

$$\Phi_{k-1} = \begin{bmatrix} I_3 & \tau_3 I_3 & 0_{3,1} & 0_{3,1} \\ O_3 & I_3 & 0_{3,1} & 0_{3,1} \\ 0_{1,3} & O_{1,3} & 1 & \tau_s \\ 0_{1,3} & O_{1,3} & 0 & 1 \end{bmatrix} \tag{1}$$

  where the propagation interval τ_s is 0.5.

- step 2, compute the system noise covariance matrix using the equation below.

$${}_{k-1} = \begin{bmatrix} 1/3 S_a \tau_s^3 I_3 & 1/2 S_a \tau_s^2 I_3 & 0_{3,1} & 0_{3,1} \\ 1/2 S_a \tau_s^2 I_3 & S_a \tau_s I_3 & 0_{3,1} & 0_{3,1} \\ 0_{1,3} & O_{1,3} & S_c^a \tau_s + 1/3 S_{cf}^a \tau_s^3 & 1/2 S_{cf}^a \tau_s^2 \\ 0_{1,3} & O_{1,3} & 1/2 S_{cf}^a \tau_s^2 & S_{cf}^a \tau_s \end{bmatrix} \tag{2}$$

  where the acceleration power spectral density(PSD) is S_a^e, the clock phase PSD is S_c^a and the clock frequency PSD is S_{cf}^a

- step 3, use the transition matrix to propagate the state estimates where could be computed by the equation below.

$$x_k^- = \Phi_{k-1} x_{k-1}^+ \tag{3}$$

- step 4, use the equation below to propagate the error covariance matrix.

$$P_k^- = \Phi_{k-1} P_{k-1}^+ \Phi_{k-1}^T + Q_{k-1} \tag{4}$$

- then for each satellites,compute the sagnac effect compensation matrix C_e, compute the predicted range r_{aj} and predicted range rate v_{aj}, then compute line-of-sight unit vector u_{aj} from the user position. it is the same as We used in the first two algorithms.

- step 5, compute the measurement matrix H using the equation below

$$H_k = \begin{bmatrix} -u^e_{a4,x} & -u^e_{a4,y} & -u^e_{a4,z} & 0 & 0 & 0 & 1 & 0 \\ -u^e_{a5,x} & -u^e_{a5,y} & -u^e_{a5,z} & 0 & 0 & 0 & 1 & 0 \\ ... & ... & ... & ... & ... & ... & ... & ... \\ -u^e_{a30,x} & -u^e_{a30,y} & -u^e_{a30,z} & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -u^e_{a4,x} & -u^e_{a4,y} & -u^e_{a4,z} & 0 & 1 \\ 0 & 0 & 0 & -u^e_{a5,x} & -u^e_{a5,y} & -u^e_{a5,z} & 0 & 1 \\ ... & ... & ... & ... & ... & ... & ... & ... \\ 0 & 0 & 0 & -u^e_{a30,x} & -u^e_{a30,y} & -u^e_{a30,z} & 0 & 1 \end{bmatrix} \tag{5}$$

  where u is the cartesian ECEF positions of the satellites.

- step 6 compute the measurement noise covariance matrix using the equation below.

$$R_k = \begin{bmatrix} \sigma_p^2 & 0 & ... & 0 & 0 & 0 & ... & 0 \\ 0 & \sigma_p^2 & ... & 0 & 0 & 0 & ... & 0 \\ ... & ... & ... & ... & ... & ... & ... & ... \\ 0 & 0 & ... & \sigma_p^2 & 0 & 0 & ... & 0 \\ 0 & 0 & ... & 0 & \sigma_r^2 & 0 & ... & 0 \\ 0 & 0 & ... & 0 & 0 & \sigma_r^2 & ... & 0 \\ ... & ... & ... & ... & ... & ... & ... & ... \\ 0 & 0 & ... & 0 & 0 & 0 & ... & \sigma_r^2 \end{bmatrix} \tag{6}$$

  where σ_p is error standard deviation of pseudo-range measurements and σ_r is error standard deviation of pseudo-range rate measurements.

- step 7 compute the kalman gain matrix using the equation below.

$$K_k = P_k^- H_k^T (H_k P_k^- H_k^T + R_k)^{-1} \tag{7}$$

- step 8 formulate the measurement innovation vector dz using the equation below.

$$R_k = \begin{bmatrix} p_a^4 - r_{a4}^- - \delta p_c^{a-} \\ p_a^5 - r_{a5}^- - \delta p_c^{a-} \\ ... \\ p_a^{30} - r_{a30}^- - \delta p_c^{a-} \\ \dot{p}_a^4 - \dot{r}_{a4}^- - \delta \dot{p}_c^{a-} \\ \dot{p}_a^5 - \dot{r}_{a5}^- - \delta \dot{p}_c^{a-} \\ ... \\ \dot{p}_a^{30} - \dot{r}_{a30}^- - \delta \dot{p}_c^{a-} \end{bmatrix} \tag{8}$$

  where p_a^j is the measured pseudo-range from satellite j to the user antenna, $\dot{p}_a^j$ is the measured pseudo-range rate from satellite j to the user antenna. δp_c^{a-} is the propagated receiver clock offset estimate and $\delta \dot{p}_c^{a-}$ is the propagated receiver clock drift estimate

- step 9 update the state estimates using the equation below

$$x_k^+ = x_k^- + K_k dz \tag{9}$$

- step 10, update the error covariance matrix using the equation below.

$$P_k^+ = (I - K_k Hk) P_k^- \tag{10}$$

end for

2.4 Outlier Detection

This algorithm is from task 3 of workshop 1[1]. It is to add residual-based outlier detection at each epoch.

- step a, compute the residuals vector$v$ using equation (6) of [1]. Where I_m is the m*m identity matrix, where m is the number of measurements
- step b, compute the residuals covariance matrix C_v using equation(7)of [1]. Where σp is the measurement error standard deviation, suitable value is 5m. Notes that this equation only applies to unweighted least-squares estimation.
- step c, Compute the normalised residuals v_j and compare each with a threshold using equation(8) of [1]. where measurement j is an outlier when the following condition is met.

2.5 Corrected gyro-derived heading solution with Kalman filter

This algorithm is implemented with a 2-state Kalman filter from lecture 6.

- set initial $h^- = (0, 0)$, $p^- = [\sigma_m^2, 0; 0, \sigma_{bias}^2]$,$\Phi = [1, \tau_s; 0, 1]$ and

$$Q = \begin{bmatrix} S_{rg}\tau + 1/3 S_{bgd}\tau^3 & 1/2 S_{bgd}\tau^2 \\ 1/2 S_{bgd}\tau^2 & S_{bgd}\tau \end{bmatrix} \tag{11}$$

where S_{rg} is the gyroscope random noise with PSD. S_{bfd} is gyroscopte bias variation. σ_m is the magnetic heading noise variance.

Then for each epoch:

- step 1 use transition matrix to propagate state estimate with equation below

$$x = \Phi h^- \tag{12}$$

- step 2 propagate error variance matrix using the equation below

$$P = \Phi P^- \Phi^T + Q \tag{13}$$

- step 3 compute measurement matrix $H_k = [-1, 0]$
- step 4 formula measurement innovation vector with equation below

$$dz = (\Psi^M - \Psi^G) - H_k x \tag{14}$$

where Ψ^M is the heading provided by the csv file and Ψ^G is the gyro heading

- step 5 compute measurement noise variance matrix $R = \sigma_m^2$
- step 6 compute kalman filter gain matrix using $K = PH_k^T(H_k PH_k^T + R)$
- step 7 update state estimate

$$x^+ = x + K * dzP^+ = (I_3 - K * H_k)P \tag{15}$$

- finally update variables for next epoch $h^- = x^+$ and $P^- = P^+$

2.6 Dead Reckoning navigation

This algorithm is used from the task 1 of workshop 3[3].

- the initial position x_0 could be got from the first algorithm $initialPositioning$ and initial velocity $V_{N,0} = V_0 \cos(\Psi)$,$V_{E,0} = V_0 \sin(\Psi)$
- for each epoch, compute the average velocity using the equation(1) from [3].
- compute latitude L_k and longitude λ_k using the equation (2) from [3]. where R_N is the meridian radius of curvature and R_E is the transverse radius of curvature.
- computed the damped instantaneous DR velocity at each epoch using the equation(3) from [3].

2.7 Dr/GNSS integration using a 4-state Kalman filter

This algorithm is implemented with a 4-state kalman filter from task 2 of workshop3[3].

- set the initial state vector to zero using the equation(4) of [3]. and the state estimation error covariance matrix P_0^+ using the equation (5) of [3].

Then implement a ten steps kalman filter:

- step 1, compute tthe transition matrix using the equation(6) of [3]. where the propagation interval τ_s is 0.5
- step 2, compute the system noise covariance matrix using the equation(7) of [3]. where the DR power spectral density is $P_D R$.

- step 3, Propagate the state estimates using the equation (8) of [3].
- step 4, Porpagate the error covariance matrix usin the equation(9) of [3].
- step 5, compute the measurement matrix H_k using the equation(10) of[3].
- step 6, compute the measurement noise covariance matrix R_k using the equation (11) of [3]. where $\sigma)Gr$ is the error standard deviation of GNSS position measurements. and $\sigma)Gv$ is the error standard deviation of GNSS velocity measurements.
- step 7, compute the kalman gain matrix using the equation (12) of [3].
- step 8, formula the measurement innovation vector using the equation(13) of [3]. where G denotes the GNSS-indicated solution obtained from $workshop3_GNSS_Pos_Vel.csv$ and D denotes the DR-indicated solution obtained from task 1.
- step 9,update the state estimates using the equation (14) of [3].
- step 10, update the error covariance matrix using the equation(15) of [3].
- Finally, use the kalman filter estimates to correct DR solution at each epoch using the equation (16) of [3]. where C denotes the DR solution.

3 Graphs

3.1 integration DR and GNSS

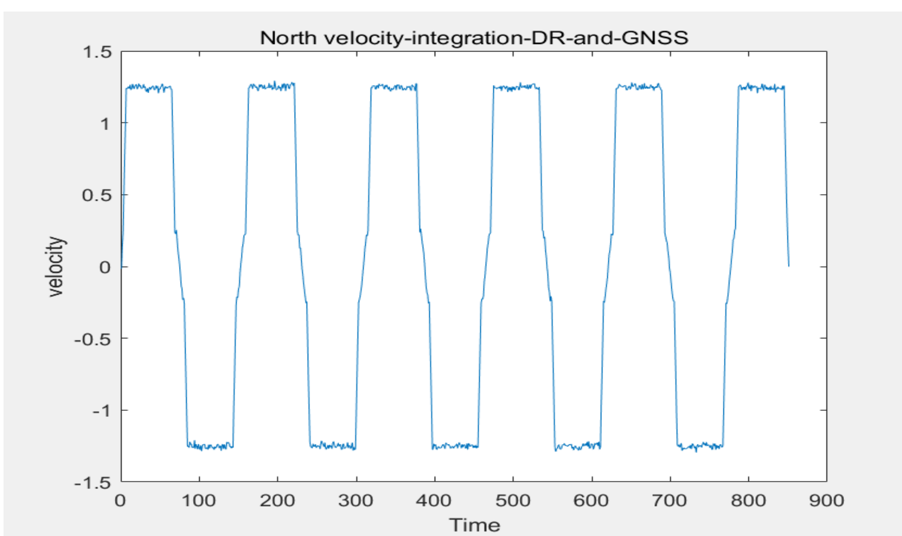

Figure 1: North velocity-integration-DR and GNSS

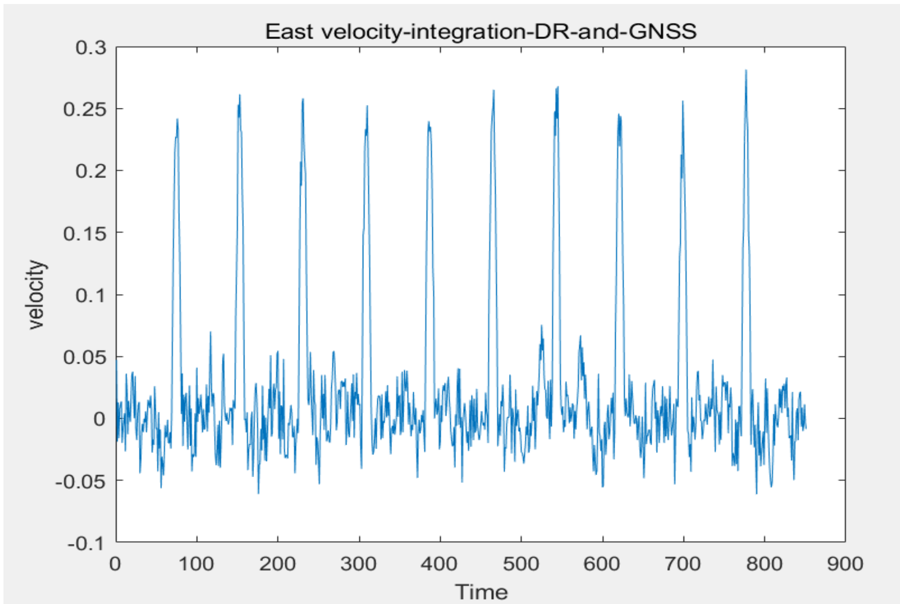

Figure 2: East velocity-integration-DR and GNSS

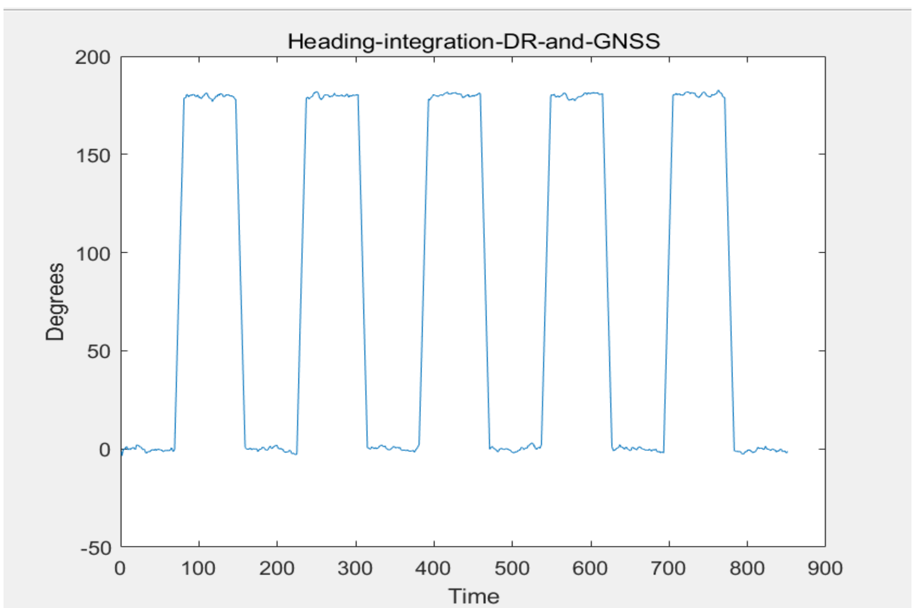

Figure 3: Heading-integration-DR and GNSS

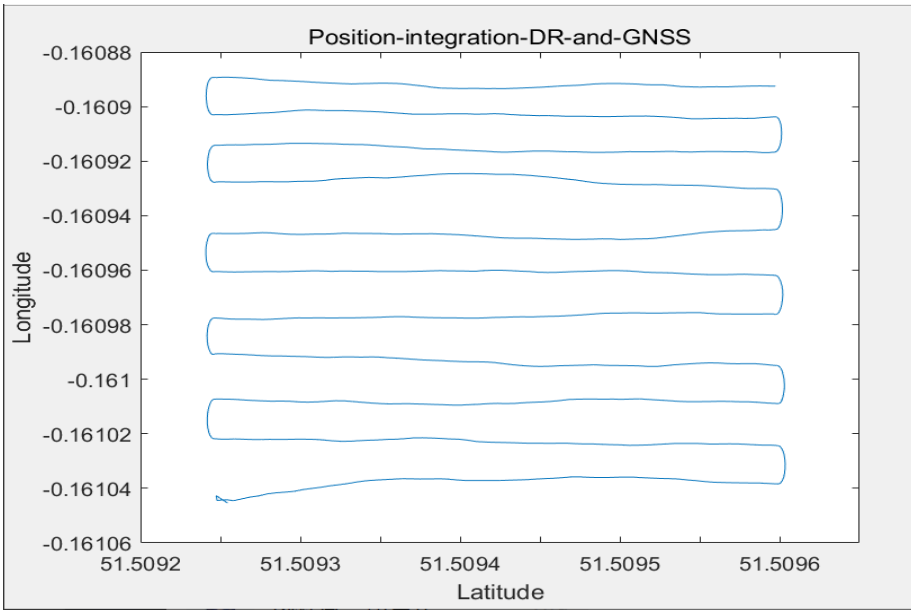

Figure 4: Position-integration-DR and GNSS

3.2 GNSS

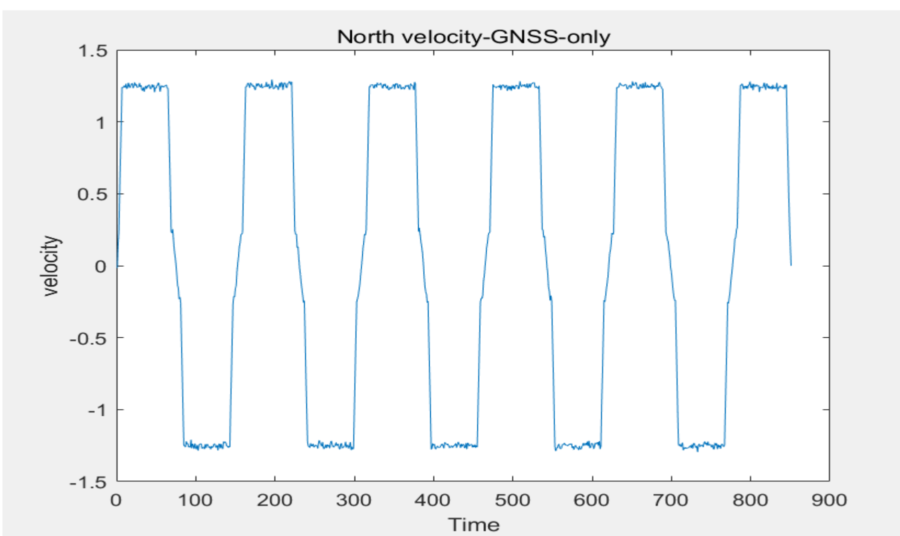

Figure 5: North velocity-GNSS only

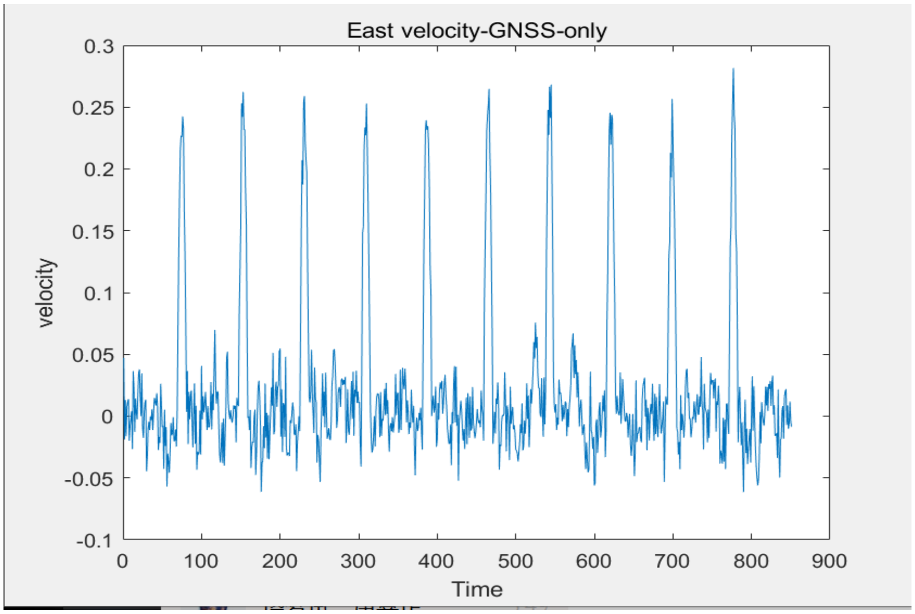

Figure 6: East velocity-GNSS only

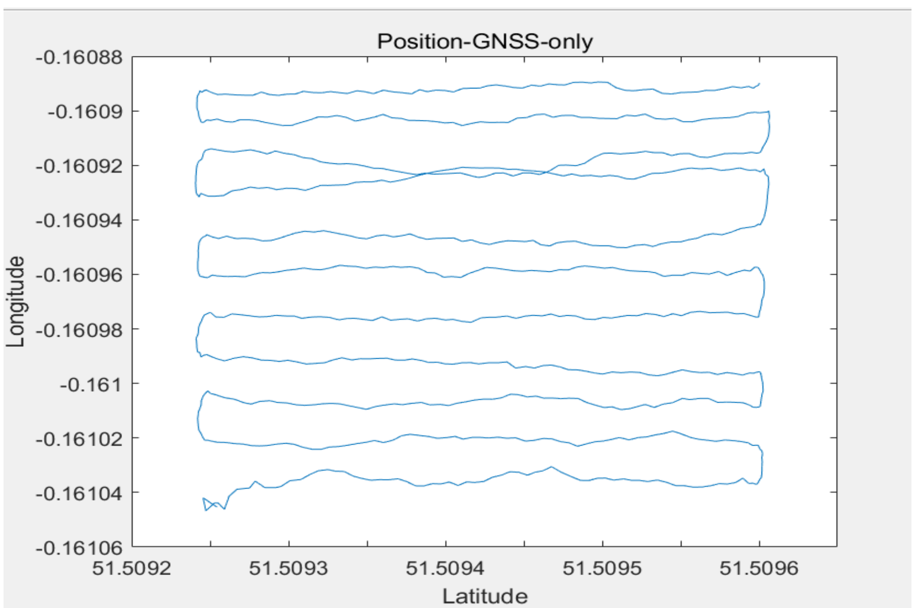

Figure 7: Position-GNSS only

3.3 Dead reckoning

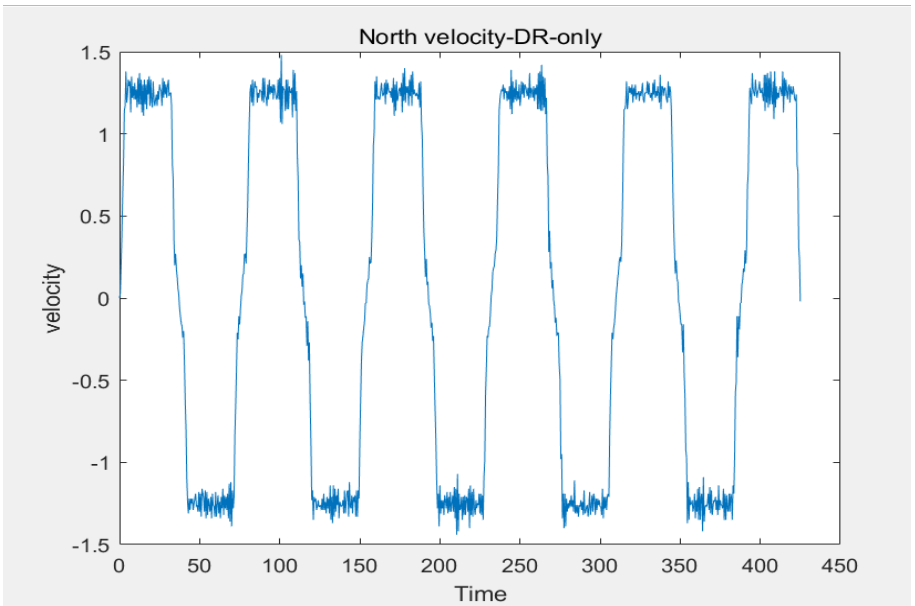

Figure 8: North velocity-DR only

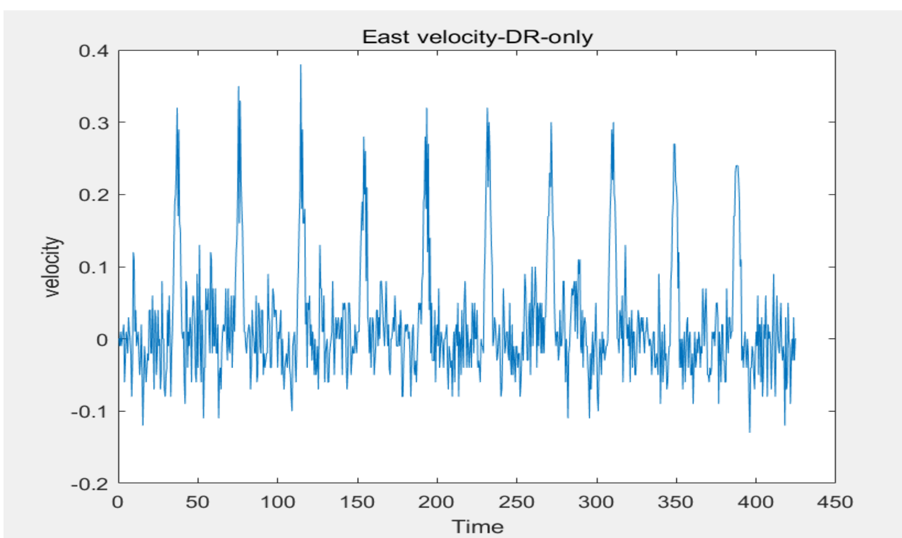

Figure 9: Eorth velocity-DR only

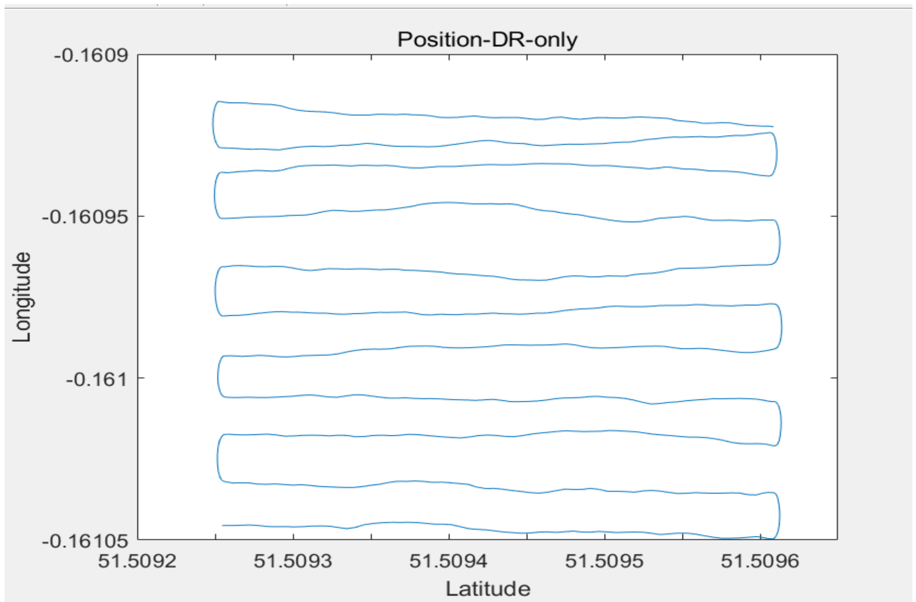

Figure 10: Position-DR only

4 A brief discussion of my results

The three methods I used will generate three csv files, which are *dead_reckon.csv*, *GNSS.csv*, and *integration_DR_and_GNSS.csv*. The first two files are used to compare and check out the result. However, the numerical differences in the files are very small, and it is difficult to analyze the results purely numerically. So the following analysis is mainly based on pictures.
First, by observing the trajectory map of the position obtained by the three methods, because this is easier to observe than velocity and heading, by comparing the results we can see that each point in the gnss trajectory map has a certain accuracy error, and even the two straight lines will overlap, this is still in the case of using the outlier. When the outlier is not used, there will be obvious errors at the end. I have not attached the picture. The error of DR seems to be relatively small. After integrating these two methods, it can be clearly seen that the result is more accurate. This is a lawnmower which is weeding on a rectangular grass.
It is difficult to compare the accuracy of these three methods from the speed image. But it is obvious that the lawnmower is doing repetitive acceleration and deceleration movements in a straight line, as well as turning action.
From the heading image, it is clear that this is a cyclical movement, and there will be a significant change every time, obviously this lawnmower is turning. The remaining heading will remain around 0 and 180.

5 Code

5.1 GNSS.m

```
function GNSS_results=GNSS
clear variables;
Define_Constants
%load data from csv file
pseudoRanges = csvread('Pseudo_ranges.csv');
pseudoRangeRates = csvread('Pseudo_range_rates.csv');

%store data into separate variables
time = pseudoRanges(2:end,1);
id = pseudoRanges(1,2:end);
pseudo_ranges = pseudoRanges(2:end,2:end);
pseudo_range_rates = pseudoRangeRates(2:end,2:end);

%step 1 because we dont know the initial position, so I use ...
    least-square to
%estimate the initial position until it dont converge
startingPosition = initialPositioning(time,id,pseudo_ranges);
%assume initial velocity is zero
startingVelocity = [0;0;0];

%step 2 implement the same method for all epochs
outlier_list = [0 0];
[positions,velocities,d_rho_c,dd_rho_c,outlier_solutions] = ...
    multipleEpochs(time,id,...
    pseudo_ranges,pseudo_range_rates,startingPosition,startingVelocity);
disp('LSE for all epochs done.')

% find the outlier satellites list
for k=1:size(time,1)
    [index,sat_id_number] = max(abs(outlier_solutions(k,:)));
    if index > 0
       outlier_list = [outlier_list;k,sat_id_number];
    end
end
%disp(outlier_list);

%step 3 implement kalman filter
GNSS_results = ...
    gnssKalmanFilter(time,id,pseudo_ranges,pseudo_range_rates, ...
    positions(:,1),velocities(:,1),d_rho_c,dd_rho_c,outlier_list);

%save result and write to a csv.file
gnss_result=zeros(size(time,1),5);
gnss_result(:,1)=time;
gnss_result(:,2:3)=GNSS_results(1:2,:)'*rad_to_deg;
gnss_result(:,4:5)=GNSS_results(4:5,:)';
writematrix(gnss_result,'GNSS.csv');

%draw
figure
plot(GNSS_results(1,:)'*rad_to_deg,GNSS_results(2,:)'*rad_to_deg);
```

```
xlabel('Latitude')
ylabel('Longitude')
title('Position-GNSS-only')

figure
plot(gnss_result(:,4))
xlabel('Time')
ylabel('velocity')
title('North velocity-GNSS-only')

figure
plot(gnss_result(:,5))
xlabel('Time')
ylabel('velocity')
title('East velocity-GNSS-only')
end


function ...
    [positions,velocities,clockOffset,clockOffset2,outlier_list] ...
    = ...
    multipleEpochs(time,sat_id,pseudo_ranges,pseudo_range_rates,...
    initial_positions,initial_velocirt)

Define_Constants
%step a. convert latitude, longititude and height to cartesian ...
    ECEF position
latitude = initial_positions(1,1);
longitude = initial_positions(2,1);
height = initial_positions(3,1);

%change ned to ecef using the function given by workshop
[r_eb_e,v_eb_e] = ...
    pv_NED_to_ECEF(latitude,longitude,height,initial_velocirt);

%inital clock offset estimation
clockOffset = 0;
clockOffset2 = 0;

%skew symmetric matrix
omegaE = [0,-omega_ie,0;
         omega_ie,0,0;
         0,0,0];

%define variables for using later
positions = zeros(3,size(time,1));
velocities = zeros(3,size(time,1));
outlier_list = zeros(size(time,1),size(pseudo_ranges,2));

for i=1:size(time,1)

    %define variables for using later
    total_sat_r_es_e = zeros(3,size(sat_id,2));
    total_sat_v_es_e = zeros(3,size(sat_id,2));
    r_aj = zeros(1,size(sat_id,2));
    u_aj = zeros(3,size(sat_id,2));
    v_aj = zeros(1,size(sat_id,2));
```

```
    dz = zeros(size(sat_id,2),1);
    d_z = zeros(size(sat_id,2),1);
    H = zeros(size(sat_id,2),4);

    for j=1:size(sat_id,2)
        %get value for satellite
        %step b cartesian ecef positions of satellites at time 0
        [sat_r_es_e,sat_v_es_e] = ...
            Satellite_position_and_velocity(time(i),sat_id(j));
        total_sat_r_es_e(:,j) = sat_r_es_e';
        total_sat_v_es_e(:,j) = sat_v_es_e';

        %step c Predict range from the approximate user position
        temp=eye(3,3)*total_sat_r_es_e(:,j) - r_eb_e;
        r_a=sqrt(temp'*temp);

        %Sagnac effect compensation matrix
        C_e = [1,omega_ie*r_a/c,0;
            -omega_ie*r_a/c,1,0;
            0,0,1];

        %recalcuate r_aj
        [C_e,r_a] = Raj(r_eb_e,total_sat_r_es_e(:,j)');
        r_aj(:,j) = r_a;

        %step d compute line-of-sight unit vector for satellite
        u_a = (C_e*total_sat_r_es_e(:,j) - r_eb_e) / r_aj(:,j);
        u_aj(:,j) = u_a;

        %get velocity
        v_a = u_a'*(C_e*(total_sat_v_es_e(:,j) + ...
            omegaE*total_sat_r_es_e(:,j)) - (v_eb_e + ...
                omegaE*r_eb_e));
        v_aj(:,j) = v_a;

        %step e Formulate the predicted state vector,
        %measurement innovation vector and
        %measurement matrix
        x_minus = [r_eb_e;clockOffset];
        %measurement innovation vector
        dz(j,1) = pseudo_ranges(i,j) - r_aj(1,j) - clockOffset;

        %measurement matrix
        H(j,:) = [-u_aj(:,j)' 1];

        %using the same method for velocity
        %predicted state vector
        x_minus_v = [v_eb_e;clockOffset2];
        %measurement innovation vector
        d_z(j,1) = pseudo_range_rates(i,j) - v_aj(1,j) - ...
            clockOffset2;
    end

    %check for outliers
    outlier_list(i,:) = findOutliers(H,dz);

    %f. Compute position and reciever clock offset using unweighted
```

```
    %least-squares
    x_new = x_minus + pinv(H'*H)*H'*dz;
    r_eb_e = x_new(1:3,1);
    clockOffset = x_new(4,1);

    %f. Compute velocity and reciever clock offset using unweighted
    %least-squares
    x_plus = x_minus_v + (H'*H)\H'*d_z;
    v_eb_e = x_plus(1:3,1);
    clockOffset2 = x_plus(4,1);

    %return in ecef format
    positions(:,i) = r_eb_e;
    velocities(:,i) = v_eb_e;
end
end

function gnss_solutions = ...
    gnssKalmanFilter(time,sat_id,pseudo_ranges,pseudo_range_rates, ...
    ...
    r_eb_e,v_eb_e,clockOffset,clockOffset2,outlier_list)

%workshop2 task 2b:  GNSS Kalman Filter Multiple Epochs with ...
    8-state kalman
%filter

Define_Constants

%initalize matrices
solutions = zeros(6,size(time,1));
total_sat_r_es_e = zeros(3,size(sat_id,2));
total_sat_v_es_e = zeros(3,size(sat_id,2));
r_aj = zeros(1,size(sat_id,2));
r_aj_dot = zeros(1,size(sat_id,2));
d_z = zeros(2*size(sat_id,2),1);

%initalize kalman filter state vector
[x_est,P_matrix] = ...
    Initialise_GNSS_KF(r_eb_e,v_eb_e,clockOffset,clockOffset2);

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


%compute system noise covariance matrix
Sa = 5;
Scphi = 0.01;
Scf = 0.04;
Q = ...
    [1/3*Sa*tau_s^3*eye(3,3) 1/2*Sa*tau_s^2*eye(3,3) zeros(3,1) ...
        zeros(3,1);
    1/2*Sa*tau_s^2*eye(3,3) Sa*tau_s*eye(3,3) zeros(3,1) zeros(3,1);
```

```
    zeros(1,3) zeros(1,3) Scphi*tau_s + 1/3*Scf*tau_s^3 ...
        1/2*Scf*tau_s^2;
    zeros(1,3) zeros(1,3) 1/2*Scf*tau_s^2 Scf*tau_s];
tempPseudo_ranges=pseudo_ranges;
tempPseudo_ranges_rates=pseudo_range_rates;
tempId=sat_id;
for epoch=1:size(time,1)
    %use transition matrix to propogate state estimate
    x_k = phi*x_est;
    for i=2:size(outlier_list,1)
        if epoch==outlier_list(i,1)
            pseudo_ranges(:,outlier_list(i,2)) = [];
            pseudo_range_rates(:,outlier_list(i,2)) = [];
            sat_id(outlier_list(i,2)) = [];
        end
    end
    %propogate state covariance matrix
    P = phi*P_matrix*phi' + Q;

    %compute line of sight vectors
    clear u_a_all;
    clear d_z;
    for j=1:size(sat_id,2)

        r_eb_e = x_k(1:3,1);
        v_eb_e = x_k(4:6,1);
        %step b get value for satellite
        [sat_r_es_e,sat_v_es_e] = ...
            Satellite_position_and_velocity(time(epoch),sat_id(j));
        total_sat_r_es_e(:,j) = sat_r_es_e';
        total_sat_v_es_e(:,j) = sat_v_es_e';

        %step c. Predict range from the approximate user position
        r_a = sqrt((eye(3,3)*total_sat_r_es_e(:,j) - r_eb_e)' * ...
            (eye(3,3)*total_sat_r_es_e(:,j) - r_eb_e));
        %Sagnac effect compensation matrix
        C_e = [1,omega_ie*r_a/c,0;
            -omega_ie*r_a/c,1,0;
            0,0,1];
        %recalcuate r_aj
        [C_e,r_a] = Raj(r_eb_e,total_sat_r_es_e(:,j)');
        r_aj(:,j) = r_a;

        %compute line of sight vector
        u_a = (C_e*total_sat_r_es_e(:,j) - r_eb_e) / r_aj(:,j);
        u_a_all(:,j) = u_a;

        %calculate range rates for each satellite
        r_a_dot = u_a'*(C_e*(total_sat_v_es_e(:,j) + ...
            Omega_ie*total_sat_r_es_e(:,j)) - (v_eb_e + ...
                Omega_ie*r_eb_e));
        r_aj_dot(:,j) = r_a_dot;

        %formulate measurement innovation vector
        d_z(j,1) = pseudo_ranges(epoch,j) - r_aj(1,j) - x_k(7,1);
        d_z(j+size(sat_id,2),1) = pseudo_range_rates(epoch,j) ...
            - r_aj_dot(1,j) - x_k(8,1);
```

```
    end

    %compute measurement matrix
    R_k = zeros(2*size(sat_id,2),2*size(sat_id,2));
    for r = 1:size(sat_id,2)
        R_k(r,r) = 10^2;
        R_k(r+size(sat_id,2),r+size(sat_id,2)) = 0.05^2;
    end
    %H_k = zeros(2*size(sat_id,2),2*size(u_a_all,1)+2);
    clear H_k
    for k=1:size(u_a_all,2)
        H_k(k,:) = [-u_a_all(:,k).',zeros(1,3),1,0];
        H_k(k+size(sat_id,2),:) = [zeros(1,3),-u_a_all(:,k).',0,1];

    end

    %disp(size(R_k));
    %disp(size(H_k));
    %Compute Kalman Gain matrix
    K = P*H_k'/(H_k*P*H_k' + R_k);

    %update state estimates
    x_plus = x_k + K*d_z;
    P_plus = (eye(size(P,1)) - K*H_k)*P;

    %append solutions
    [L_b,lambda_b,h_b,v_eb_n] = ...
        pv_ECEF_to_NED(x_plus(1:3),x_plus(4:6));
    solutions(:,epoch) = [L_b;lambda_b;h_b;v_eb_n];

    %update variables
    x_est = x_plus;
    P_matrix = P_plus;
    pseudo_ranges=tempPseudo_ranges;
    pseudo_range_rates=tempPseudo_ranges_rates;
    sat_id=tempId;
end
gnss_solutions = solutions;
end

function outlier_index = findOutliers(H,d_z)
%define variables
listss = size(H,1);
%step a compute residuals vector
v = (H*pinv(H'*H)*H' - eye(listss))*d_z;

%step b compute residual covariance
C_v = (eye(listss) - H*inv(H'*H)*H')*5^2;

%step c compute normalized residuals and compare to threshold
outlier_index = zeros(1,listss);
for i=1:size(H,1)
    if norm(v(i)) > sqrt(C_v(i,i))*6
        outlier_index(i) = v(i);
    end
end
```

```
end

function [C,r_aj] = Raj(rea,rej)
Define_Constants;
r_aj = 0;
temp = inf;
%recursion to find r_aj when it converges
while r_aj≠temp
temp = r_aj;
C = [1,omega_ie*r_aj/c,0;
    -omega_ie*r_aj/c,1,0;
    0,0,1];
temp2=C*rej'-rea;
r_aj=sqrt(temp2'*temp2);
end
end
```

5.2 initialPositioning.m

```
function [initial_pos] = initialPositioning(time,id,pseudo_ranges)
Define_Constants

%step b Compute the Cartesian ECEF positions of the satellites ...
    at time 0
total_sat_r_es_e = zeros(size(id,2),3);
for i=1:size(id,2)
    [sat_r_es_e,sat_v_es_e] = ...
        Satellite_position_and_velocity(time(1),id(i));
    total_sat_r_es_e(i,:) = sat_r_es_e;
end

%set initial data
r_ea = [0;0;0];

clockOffset = 0;
last_r_ea = [0;0;0];
thre = 0.10;
error = inf;
while (error > thre)
    %step c Predict the ranges from the approximate user position
    %to each satellite
    r_aj = zeros(size(id,2),1);
    for i=1:size(id,2)
        %implement recursion
        %initial range computation
        temp=eye(3,3)*total_sat_r_es_e(i,:)' - r_ea;
        r_a=sqrt( temp.' * temp);

        %Sagnac effect compensation matrix
        C_e = [1,omega_ie*r_a/c,0;
              -omega_ie*r_a/c,1,0;
               0,0,1];
        %recalcuate r_aj using C_e
        r_a = sqrt((C_e*total_sat_r_es_e(i,:)' - r_ea)' * ...
```

```
            (C_e*total_sat_r_es_e(i,:)' - r_ea));

        r_aj(i,:) = r_a;
    end

    %step d Compute the line-of-sight unit vector from the
    %approximate user position to each satellite
    u_aj = zeros(3,size(id,2));
    for i=1:size(id,2)
        u_a = (C_e*total_sat_r_es_e(i,:)' - r_ea) / r_aj(i);
        u_aj(:,i) = u_a;
    end

    %step e Formulate the predicted state vector,
    %measurement innovation vector
    % and measurement matrix

    % predicted state vector
    x_minus = [r_ea;clockOffset];

    %measurement innovation vector
    dz = zeros(size(id,2),1);
    for i=1:size(id,2)
        dz(i,1) = pseudo_ranges(1,i) - r_aj(i,1) - clockOffset;
    end

    %measurement matrix
    H = zeros(size(id,2),4);
    for i=1:size(id,2)
        H(i,:) = [-u_aj(:,i)' 1];
    end

    %step f Compute the position and receiver clock offset
    %using unweighted least-squares
    x_plus = x_minus + pinv((H'*H))*H'*dz;
    %set current result
    r_ea = x_plus(1:3,1);
    clockOffset = x_plus(4,1);
    error = abs(norm(r_ea) - norm(last_r_ea));
    last_r_ea = r_ea;
end
%step g Convert this Cartesian ECEF position solution
%to latitude, longitude and height
[latitude,longitude,height,¬] = pv_ECEF_to_NED(r_ea,clockOffset);
initial_pos = [latitude;longitude;height];
end
```

5.3 deadReckoning.m

```
function dr_result=deadReckoning
clear variables
Define_Constants

%load data from csv.file
```

```
pseudoRanges = csvread('Pseudo_ranges.csv');
%save into separate variables
time = pseudoRanges(2:end,1);
id = pseudoRanges(1,2:end);
pseudo_ranges = pseudoRanges(2:end,2:end);
initialPosition = initialPositioning(time,id,pseudo_ranges);

%load data from csv.file and save into separate variables
file = csvread('Dead_reckoning.csv');
time = file(:,1);
left_front = file(:,2);
right_front = file(:,3);
left_back = file(:,4);
right_back = file(:,5);
gyro = file(:,6);
heading = file(:,7)*deg_to_rad;
forward_speed=((right_front+right_back)/2+(left_front+left_back)/2)/2;

%forward_speed = file(:,2);
%heading = file(:,3)*deg_to_rad;
h=initialPosition(3);
position = zeros(size(time,1),2);
position(1,:) = initialPosition(1:2);
%set initial data
average_velocity = zeros(size(time,1)-1,2);
ins_dr_velocity = zeros(size(time,1)-1,2);
ins_dr_velocity(1,1) = forward_speed(1)*cos(heading(1));
ins_dr_velocity(1,2) = forward_speed(1)*sin(heading(1));


%-----------------------------task 1------------------------------%
for i=2:size(time,1)
    %compute the average velocity in north and east
    average_velocity(i,1) = ...
        0.5*(cos(heading(i))+cos(heading(i-1)))...
        *forward_speed(i); %average_V_N
    average_velocity(i,2) = ...
        0.5*(sin(heading(i))+sin(heading(i-1)))...
        *forward_speed(i);%averge_V_E

    %RN is the meridian radius of curvature and
    %RE is the transverse radius of curvature
    [R_N,R_E] = Radii_of_curvature(position(i-1,1));

    %compute latitude and longitude from their counterparts
    position(i,1)=position(i-1,1)+(average_velocity(i,1)...
        *(time(i)-time(i-1)))/(R_N+h);
    position(i,2)=position(i-1,2)+(average_velocity(i,2)...
        *(time(i)-time(i-1)))/((R_E+h)*cos(position(i,1)));

    % compute the damped instantaneous DR velocity at each epoch
    ins_dr_velocity(i,1)=1.7*average_velocity(i,1)-0.7...
        *ins_dr_velocity(i-1,1);%V_N
    ins_dr_velocity(i,2)=1.7*average_velocity(i,2)-0.7...
        *ins_dr_velocity(i-1,2);%V_E
end
position=position*rad_to_deg;
```

```matlab
ins_dr_velocity=roundn(ins_dr_velocity,-2);
%disp(position);
%disp(ins_dr_velocity);

%implement Gyro-Magnetometer Integration
gyro_heading = zeros(1,size(time,1));
gyro_heading(1) = heading(1);
for epoch=2:size(time,1)
   gyro_heading(epoch) = gyro_heading(epoch-1) + gyro(epoch)*0.5;
end
heading_solutions = ...
    gyroMagnetometerIntegration(time,heading,gyro_heading);
heading=heading_solutions';


%save result and write to a csv.file
dr_result=zeros(size(time,1),6);
dr_result(:,1)=time;
dr_result(:,2:3)=position;
dr_result(:,4:5)=ins_dr_velocity;
dr_result(:,6)=heading*rad_to_deg;
writematrix(dr_result,'dead_reckon.csv');

%draw
figure
plot(position(:,1),position(:,2));
xlabel('Latitude')
ylabel('Longitude')
title('Position-DR-only')

figure
plot(time,heading*rad_to_deg)
xlabel('Time')
ylabel('Degrees')
title('Heading-DR-only')

figure
plot(time,ins_dr_velocity(:,1))
xlabel('Time')
ylabel('velocity')
title('North velocity-DR-only')

figure
plot(time,ins_dr_velocity(:,2))
xlabel('Time')
ylabel('velocity')
title('East velocity-DR-only')
end
```

5.4 gyroMagnetometerIntegration.m

```matlab
function headingResult = ...
    gyroMagnetometerIntegration(time,heading,gyro_heading)
Define_Constants
```

```
%apply two state kalman filter for Gyro-Magnetometer Integration

%initalize data
result = zeros(size(time,1),1);
h_minus = [0;0];
sigma_bias = 1;
sigma_heading = 4*deg_to_rad;
tau_s=0.5;
P_minus = [sigma_heading^2,0;0,sigma_bias^2];
%The heading error is the integral of the gyro bias
phi = [1,tau_s;
       0,1];

%Gyro random noise with power spectral density (PSD)
S_rg =  1*10^-4;
%Gyro bias variation with PSD
S_bgd = 3*10^-6;

Q = [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];

for i=1:size(time,1)
    %step 1 use transition matrix to propogate state estimate
    x = phi*h_minus;

    %step 2 propagate error covariance matrix
    P = phi*P_minus*phi' + Q;

    %step 3 compute measurement matrix
    H_k = [-1 0];

    %step 4 Formulate measurement innovation vector
    d_z = heading(i) - gyro_heading(i) - H_k*x;

    %step 5 compute measurement noise covaraince matrix
    sigma_m = 4*deg_to_rad;
    R = diag([sigma_m^2]);

    %step 6 Compute Kalman Gain matrix
    K = P*H_k'/(H_k*P*H_k' + R);

    % step 7 update state estimates
    x_plus = x + K*d_z;
    P_plus = (eye(size(P,1)) - K*H_k)*P;

    %store results
    result(i,:) = (gyro_heading(i) - x_plus(1))';

    %update variables
    h_minus = x_plus;
    P_minus = P_plus;
end
headingResult = result';
end
```

5.5 integrationDRandGNSS.m

```
clear variables;
Define_Constants
%load data from another two methods, GNSS and dead reckoning.
dr_result=deadReckoning;
gnss_result=GNSS;
%store data in separate variables
time2=dr_result(:,1);
position=dr_result(:,2:3);
ins_dr_velocity=dr_result(:,4:5);

geodetic_position=gnss_result(1:3,:)';
referenced_velocity=gnss_result(4:6,:)';

%-----------------task 2--------------------------------%

%define a 4 state kalman filter estimating north and east DR
%velocity error, DR latitude error and DR longitude error
%the state vector is thus
x=zeros(4,1);
newPosition = geodetic_position(1,1:2);
newVelocity = referenced_velocity(1,1:2);
sigma_v=0.1;
sigma_r=10;
[R_N,R_E] = Radii_of_curvature(geodetic_position(1,1));
%The state estimation error covariance matrix
%is therefore initialised at
P_plus=eye(4,4);
P_plus(1,1)=sigma_v^2;
P_plus(2,2)=sigma_v^2;
P_plus(3,3)=(sigma_r^2)/((geodetic_position(1,3)+R_N)^2);
P_plus(4,4)=(sigma_r^2)/((R_E+geodetic_position(1,3))...
^2*cos(geodetic_position(1,1))^2);

%ten steps of Kalman filter
for i=2:size(time2,1)
    [R_N,R_E] = Radii_of_curvature(geodetic_position(i,1));
    %step 1 Compute the transition matrix
    tau_s=0.5;
    phi=eye(4,4);
    phi(3,1)=tau_s/(R_N+geodetic_position(i-1,3));
    phi(4,2)=tau_s/((R_E+geodetic_position(i-1,3))*...
        cos(geodetic_position(i-1,1)));

    %step 2 Compute the system noise covariance matrix
    Q=zeros(4,4);
    S_DR=0.2;
    Q(1,1)=S_DR*tau_s;
    Q(1,3)=0.5*((S_DR*tau_s^2)/(R_N+geodetic_position(i-1,3)));
    Q(2,2)=S_DR*tau_s;
    Q(2,4)=0.5*((S_DR*tau_s^2)/((R_E+geodetic_position(i-1,3))*...
        cos(geodetic_position(i-1,1))));
    Q(3,1)=0.5*((S_DR*tau_s^2)/(R_N+geodetic_position(i-1,3)));
    Q(3,3)=(1/3)*((S_DR*tau_s^3)/(R_N+geodetic_position(i-1,3))^2);
    Q(4,2)=0.5*((S_DR*tau_s^2)/((R_E+geodetic_position(i-1,3))*...
```

```
        cos(geodetic_position(i-1,1))));
    Q(4,4)=(1/3)*((S_DR*tau_s^3)/((R_E+geodetic_position(i-1,3))^...
        2*cos(geodetic_position(i-1,1))^2));

    %step 3 Propagate the state estimates:
    x_minus=phi*x;

    %step 4 Propagate the error covariance matrix:
    P_minus=phi*P_plus*phi'+Q;

    %step 5 Compute the measurement matrix
    H=[0,0,-1,0;
        0,0,0,-1;
        -1,0,0,0;
        0,-1,0,0];

    %step 6 Compute the measurement noise covariance matrix
    positionError_std=5;
    velocityError_std=0.02;
    R=zeros(4,4);
    R(1,1)=positionError_std^2/(R_N+geodetic_position(i,3))^2;
    R(2,2)=positionError_std^2/((R_E+geodetic_position(i,3))^2....
    *cos(geodetic_position(i,1))^2);
    R(3,3)=velocityError_std^2;
    R(4,4)=velocityError_std^2;

    %step 7 Compute the Kalman gain matrix
    K=P_minus*H'* pinv(H*P_minus*H'+R);

    %step 8 Formulate the measurement innovation vector
    dz=[geodetic_position(i,1)-position(i,1)*deg_to_rad;
        geodetic_position(i,2)-position(i,2)*deg_to_rad;
        referenced_velocity(i,1)-ins_dr_velocity(i,1);
        referenced_velocity(i,2)-ins_dr_velocity(i,2)]-H*x_minus;

    %step 9 Update the state estimates
    x_plus=x_minus+K*dz;

    %step 10 Update the error covariance matrix
    P_plus=(eye(4,4)-K*H)*P_minus;

    %Use the Kalman filter estimates to correct the DR solution ...
        at each epoch
    newPosition(i,1)=position(i,1)*deg_to_rad-x_plus(3);
    newPosition(i,2)=position(i,2)*deg_to_rad-x_plus(4);
    newVelocity(i,1)=ins_dr_velocity(i,1)-x_plus(1);
    newVelocity(i,2)=ins_dr_velocity(i,2)-x_plus(2);

    %update for next epoch
    x=x_plus;
end
newPosition=[newPosition,geodetic_position(:,3)];
newVelocity=[newVelocity,referenced_velocity(:,3)];
%disp(newPosition*rad_to_deg);
%disp(roundn(newVelocity,-2));

%save result and write to a csv.file
```

```
integration_result=zeros(size(time2,1),6);
integration_result(:,1)=time2;
integration_result(:,2:3)=newPosition(:,1:2)*rad_to_deg;
integration_result(:,4:5)=newVelocity(:,1:2);
integration_result(:,6)=dr_result(:,6);
writematrix(integration_result,'integration_DR_and_GNSS.csv');

%draw
figure
plot(integration_result(:,2),integration_result(:,3));
xlabel('Latitude')
ylabel('Longitude')
title('Position-integration-DR-and-GNSS')

figure
plot(integration_result(:,6))
xlabel('Time')
ylabel('Degrees')
title('Heading-integration-DR-and-GNSS')

figure
plot(integration_result(:,4))
xlabel('Time')
ylabel('velocity')
title('North velocity-integration-DR-and-GNSS')

figure
plot(integration_result(:,5))
xlabel('Time')
ylabel('velocity')
title('East velocity-integration-DR-and-GNSS')
```

5.6 Initialise_GNSS_KF.m

```
function [x_est,P_matrix] = ...
    Initialise_GNSS_KF(r_eb_e,v_eb_e,d_rho_c,dd_rho_c)
%Initialise_Integration_KF - Initializes the Integration KF ...
    state estimates
% and error covariance matrix for Workshop 2
%
% This function created 30/11/2016 by Paul Groves
%
% Outputs:
%   x_est                 Kalman filter estimates:

%   P_matrix              state estimation error covariance matrix

% Copyright 2016, Paul Groves
% License: BSD; see license.txt for details

% Begins

% Initialise state estimates
x_est = [r_eb_e;v_eb_e;d_rho_c;dd_rho_c];
```

```

% Initialise error covariance matrix
P_matrix =  zeros(8);
for i=1:3
    P_matrix(i,i) = 10^2;
end
for i=4:6
    P_matrix(i,i) = 0.05^2;
end
P_matrix(7,7) = 100000^2;
P_matrix(8,8) = 200^2;

% Ends
```

6 Reference

[1]P.D. Groves(2020),COMP0130:ROBOT VISION AND NAVIGATION, Workshop 1: Mobile GNSS Positioning using Least-Squares Estimation.[Accessed 5,Feb,2020].
[2]P.D. Groves(2020), COMP0130:ROBOT VISION AND NAVIGATION, Workshop 2: Aircraft Navigation using GNSS and Kalman Filtering.[Accessed 7,Feb,2020].
[3]P.D. Groves(2020), COMP0130:ROBOT VISION AND NAVIGATION, Workshop 3: Multisensor Navigation.[Accessed 7,Feb,2020].
[4]P.D. Groves(2020), COMP0130:ROBOT VISION AND NAVIGATION, Lecture 6: Multisensor Integrated Navigation.[Accessed 8,Feb,2020].