COMP0130: ROBOT VISION AND NAVIGATION

Coursework 1:

Integrated Navigation for a Robotic Lawnmower

Jian Zhou
21082500
ucabj42@ucl.ac.uk

Xianjian Bai
21135608
ucabxb1@ucl.ac.uk

Hanpeng Li
22072479
ucabhl9@ucl.ac.uk

February 7, 2023

1. Introduction

A lawnmower equipped with a GNSS receiver, wheel speed sensors, magnetic compass and a low-cost MEMS gyroscope is working in somewhere in London. This coursework requires students to select a set of suitable data processing approaches and error mitigation approaches to determine an efficient route of the lawnmower. In general, our group firstly used integrated GNSS + Kalman filter to determines the position and velocity directly basing on the pseudo ranges and pseudo range rates. And then a locally dead reckoning estimation was made to make comparation. In the end, the position and velocity information, generated from GNSS Kalman and dead reckoning, were fused together to obtain more accurate position, velocity and heading information. At the end, the error from GNSS measurement and Dead Reckoning measurement would be analyzed.

2. Methodology

2.1 Integrated GNSS Kalman filter

There are at least two approaches that we can apply to handle GNSS data, which are 'Basic Kalman Filter' and 'Integrated GNSS Kalman Filter' [1]. Basic Kalman Filter takes the pre-calculated position as measurement inputs. The system state of it is the position and velocity in each direction of the robot. However, it cannot count in the error we meet in GNSS. It can only fuse all errors' distribution covariance into one simple measurement noise variance matrix R, and the determination of the noise covariance can be not so accurate. Therefore, we choose to use integrated GNSS Kalman filter, which takes the pseudo-range and pseudo range rate as inputs directly. After the transformation of measurement matrix, the pseudo-ranges and rates can be converted to system states and then we can calculate innovation for updating. The details of the GNSS algorithm and implemented equations can be found in workshop 2 instructions[1].

2.2 Dead Reckoning

Dead Reckoning can be used to obtain the speed and heading data. The speed of the four wheels can be obtained directly from the sensors to calculate the overall speed. For heading, it is necessary to smooth through the formula, and sum the weighted Gyroscopic heading and Magnetic heading. Details of the formula are in reference [2][3]. The longitude and latitude can be obtained after calculation through speed and heading [2]. However, only using DR was less effective, because multiple internal sensors will produce different systematic errors, and the accuracy will decrease with the increase of time and iterations. There will also be cumulative errors due to navigation equations approximations. It can be seen from Figure 2 that when DR is used alone, though we have corrected and compensated the calculation results through known error sources, but the results still have a certain gap with GNSS and integration method.

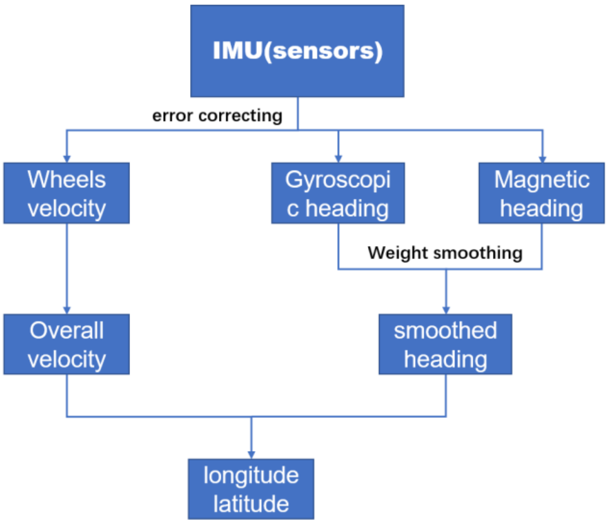

Figure 1: Dead Reckoning flow chart [2]

2.3 Basic Loosely-coupled Open-looped Integration

In this coursework, basic loosely-coupled open-loop integration is used for calculation of lawnmower's position and velocity. In current scenarios, there are two main reasons for choosing to integrate. Firstly, integrate method is more dependable than just using satellite data or just using sensor data, because it uses more data under the same conditions. Besides, since GNSS offer better long-term accuracy and inertial navigation provide better short-term accuracy, integrate method is expected to obtain better accuracy the others.

As shown in following figure, when applying this method, two solutions based on GNSS navigation and DR navigation should be first obtained. After that, Kalman filter would be applied to deal with the difference between two solutions and obtain the result would be obtained by correcting DR solutions with Kalman filter outputs. The full details of the algorithms can be found in workshops 3 instructions [4]. The equations (4) to equation (16) in the instruction was implemented in the code as appendix[4].

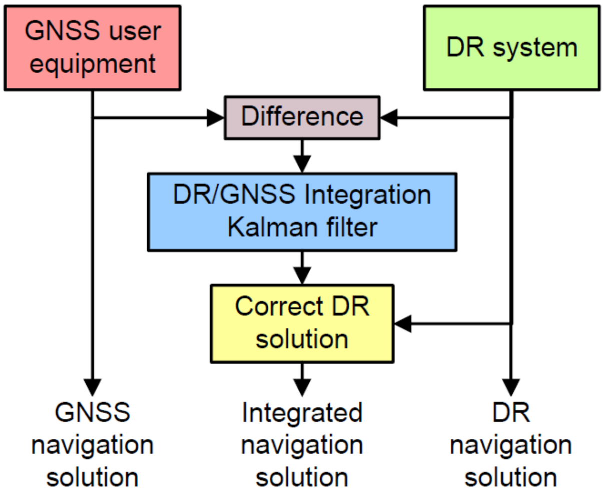

Figure 2: basic loosely coupled open-loop integration flow chart.[5]

3. Error analysis:

3.1 Integrated GNSS Kalman filter

GNSS measurement errors have five main parts, which are signal in space error, residual ionosphere and troposphere error, code tracking and multipath error and range rate tracking and multipath error [6]. We didn't use the GNSS measurement error directly because we don't have the knowledge of how these errors would interact with each other and the method to fuse them together. Therefore, we adapt the method in coursework sheet that consider them where the total standard deviation is 10m on all pseudo-range and 0.05m/s on all range rate measurements.

As for the error on properties for GNSS receiver clock. All the errors are taken into consideration directly.

In the GNSS Kalman filter part, the only parameter that is set under estimation is the acceleration power spectral density. We tried a lot of possible values and chose 0.001 m^2s^{-3} basing on the result comparison between results of GNSS and dead reckoning.

3.2 Dead Reckoning

In Dead Reckoning, the result of the sensor is equal to the sum of the actual value and all errors .

When processing velocity data, scale factor error (3%), normal noise (0.05m/s) and quantized error (0.02m/s) are considered. After judging the speed direction, the error corresponding to the speed at each moment is calculated and compensated. The gyroscope error considers bias (1 deg), scale factor error (1%), cross-coupling error (0.1%), random noise (10^{-4} rad/s). and quantization (2×10^{-4} rad/s) . Also, after judging the direction of rotation, the error corresponding to the gyroscope at each moment is calculated for compensation. Considering the earth rotation and the difference between the body coordinate system and the earth coordinate system, the modified gyroscope cannot be directly used as the heading. We obtain the final smoothed heading by a weighted sum of the Magnetic heading and the Gyroscopic heading. In order to obtain the corresponding specific gravity, the Gyro angular rate error standard deviation should be calculated. The overall angular rate error standard deviation is the root sum of squares of the contributions. After correction, the speed and heading curves are significantly smoother, and are closer to the trajectory curves of the other two methods.[2]

4. Solution & Discussion

In Figure 3, we can see the three trajectory maps obtained by applying GNSS navigation, DR method and integration method respectively, where the starting point is in the lower left corner and the ending point is in the upper right corner. It is easy to find that the target movement patterns obtained by all three methods are similar, i.e., meandering from left to right. Comparing the paths obtained by GNSS and DR, both of them have similar paths but differences in their exact positions at the same time. After combining the information of both, the integrated method gives the corrected DR route, which seems to be a reasonable route for the lawnmower to work.

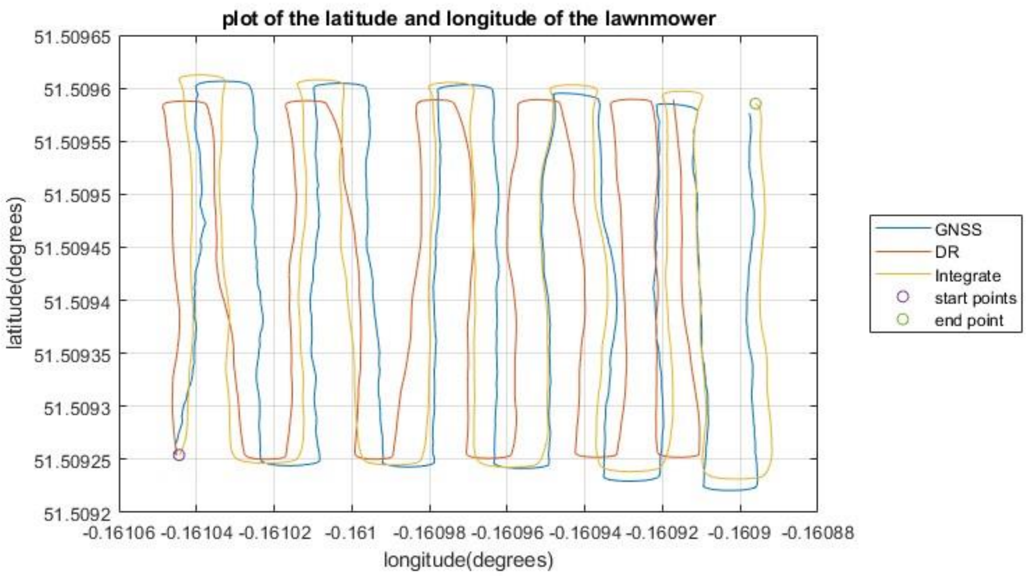

Figure 3: plot of the latitude and longitude of the lawnmower

On this basis, the velocity profiles in Figure 4 and Figure 5 are observed. The northward velocity recorded in Fig. 3 switches back and forth between a pair of values of close magnitude and opposite direction, corresponding to the lawnmower's movement over the same longitude. Meanwhile, the eastward velocity fluctuates around 0 overall, except for the periodically occurring peaks, coinciding with the steering movement of the lawnmower. In short, the changes of velocity in both directions coincide with the moving trajectory of the mower, and there is no significant difference in the velocity solution obtained by the three calculation methods.

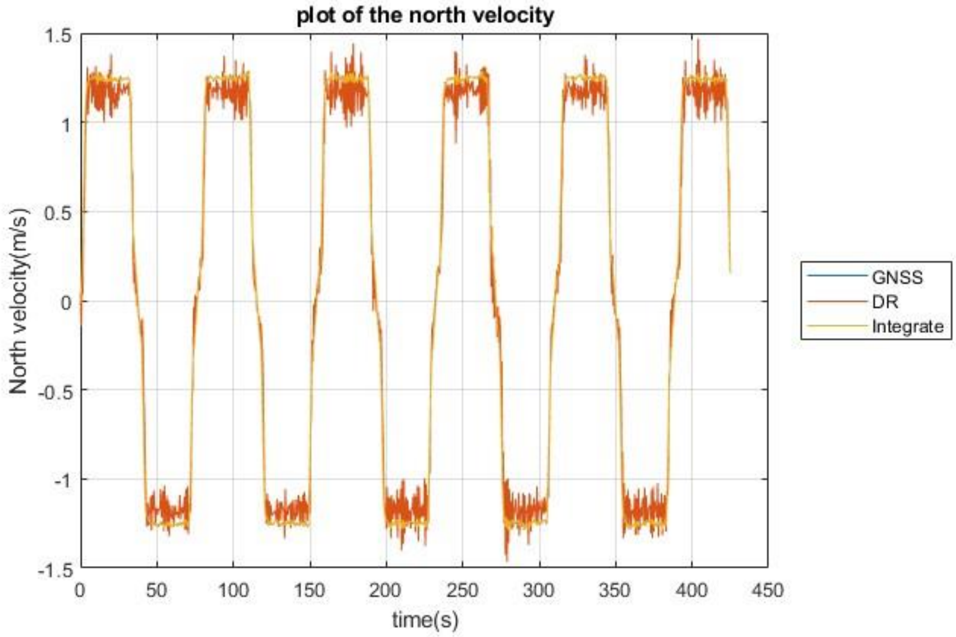

Figure 4: plot of north velocity

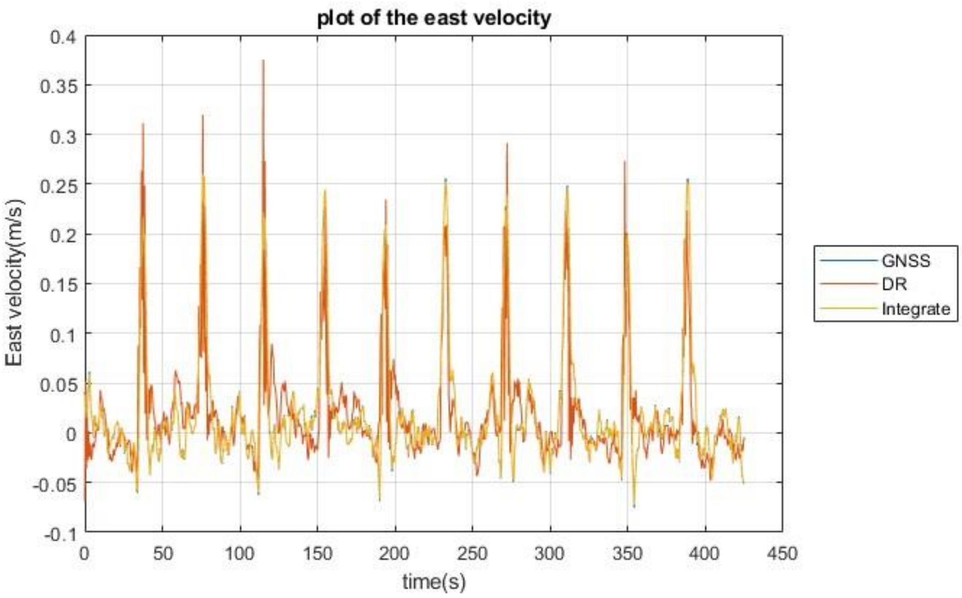

Figure 5: plot of the east velocity

Figure 6 documents the change in orientation, and the trend is generally consistent with the north velocity in Figure 4. During the operation of the lawnmower, the orientation of the lawnmower switches periodically between near 0 degrees and near 180 degrees

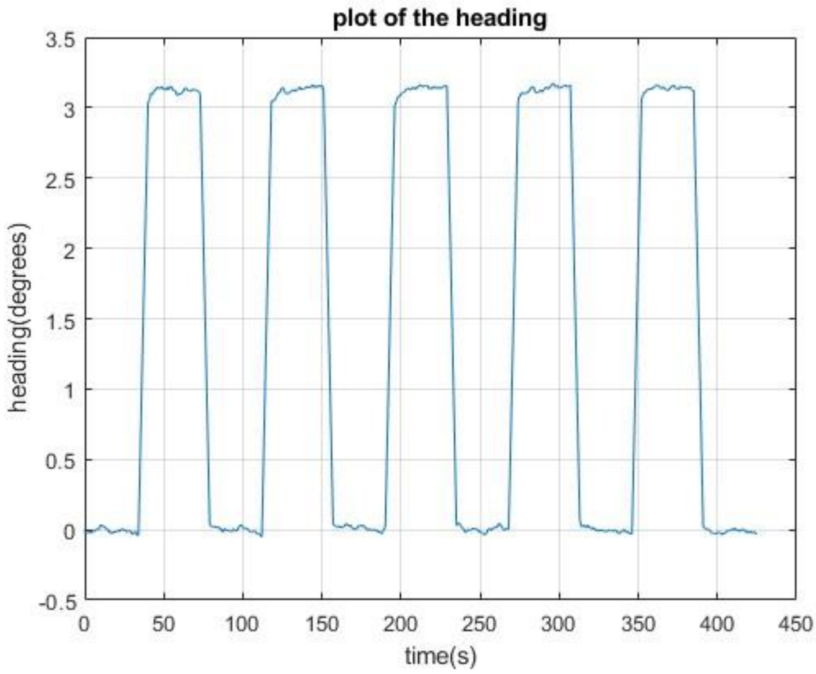

Figure 6: plot of the heading

Reference

[1] P. D. Groves, COMP130: Robot Vision and Navigation, Workshop 2: Aircraft Navigation using GNSS and Kalman Filtering.

[2] P. D. Groves, COMP130: Robot Vision and Navigation, Lecture 3A Slides.

[3] P. D. Groves, Principles of GNSS, Inertial, and Multisensor Integrated Navigation Systems (2nd Edition)

[4] P. D. Groves, COMP130: Robot Vision and Navigation, Workshop 3: Multisensor Navigation.

[5] P. D. Groves, COMP130: Robot Vision and Navigation, Lecture 3B Slides.

[6] P. D. Groves, COMP130: Robot Vision and Navigation, Coursework 1: Integrated Navigation for a Robotic Lawnmower.

Appendix

1 Integrated GNSS Kalman filter

1.1 GNSS.m

```
clc;
clear all;

% Define some constant and initialize some variables.
GNSS_ini;


% Define two kalman filter's parameters that don't change during ...
    iteration
F=[eye(3),T*eye(3),zeros(3,1),zeros(3,1); % Transition matrix
zeros(3),eye(3),zeros(3,1),zeros(3,1);
zeros(1,3),zeros(1,3),1,T;
zeros(1,3),zeros(1,3),0,1];

Q=[Sa*T^3/3*eye(3),Sa*T^2/2*eye(3),zeros(3,1),zeros(3,1); % ...
    System noise
Sa*T^2/2*eye(3),Sa*T*eye(3),zeros(3,1),zeros(3,1);
zeros(1,3),zeros(1,3), Scphi*T+Scf*T^3/3, Scf*T^2/2;
zeros(1,3),zeros(1,3), Scf*T^2/2, Scf*T];


% Assign initial values to system state
X_old=X0;
P_old=P0;
X_first=GNSS_get_EFEC_From_GNSS(data1(1:2,:)); % using ...
    least-square to calculate the initial position and clock offset

X_old(1:3)=X_first(1:3);
X_old(7)=X_first(4);


% Define some containers or variables in processing
r_ej=zeros(3,n);
u_aj=zeros(3,n);
r_aj=zeros(3,n);
r_aj_norm=zeros(1,n);
```

```
r_aj_estimated=zeros(3,n);
r_aj_estimated_norm=zeros(n,1);
r_aj_estimated_norm_dot=zeros(n,1);
r_ea=zeros(3,1);

% Record
record=zeros(m,7);

for t=1:1:m

% estimate the system state and error covariance matrix for this
% iteration
X_estimated=F*X_old;
P_estimated=F*P_old*F.'+Q;

for i=1:1:n % Calculate the unit vector between user and each ...
    satellites
j=sid(i);
time=tid(t);

r_ea_m=X_estimated(1:3); % we use estimated position and velocity
v_ea_m=X_estimated(4:6); % to be the current p and v for vector ...
    calculation

[r_ej(:,i),V_ej] = Satellite_position_and_velocity(time,j); % ...
    range vector of earth to satellite
r_aj(:,i)=r_ej(:,i)-r_ea_m; % vector between user and each ...
    satellites
r_aj_norm(1,i)=norm(r_aj(:,i),2); % distance between user and ...
    each staellites

C=[1,                              omega_ie*r_aj_norm(1,i)/c,  0;
-omega_ie*r_aj_norm(1,i)/c,  1,                            0;
0,                           0,                            1];

r_aj_estimated(:,i)=C*r_ej(:,i)-r_ea_m; % Earth displacement ...
    corrected distance vector
r_aj_estimated_norm(i)=sqrt(r_aj_estimated(:,i).'* ...
    r_aj_estimated(:,i));

%line of sight unit vector. In other words, unit vector between ...
    user and equlivent satellite.
u_aj(:,i)=r_aj_estimated(:,i)/r_aj_estimated_norm(i);

V_ej=V_ej.';
r_aj_estimated_norm_dot(i)=u_aj(:,i).'*(C*(V_ej+Omega_ie*r_ej(:,i))-(v_ea_m+Omega
end

% Measurement matrix
H=[-u_aj.',zeros(n,3),ones(n,1),zeros(n,1);
zeros(n,3),-u_aj.',zeros(n,1),ones(n,1)];
% measurement error covariance
Rk=[10^2*eye(n),zeros(n);
zeros(n),0.05^2*eye(n)];
% Kalman Gain
```

```
    K=P_estimated*H.'/(H*P_estimated*H.'+Rk);

    % calculate the innovation
    this_pesudo_range = data1(t+1,2:end).';
    this_pesudo_range_rate = data2(t+1,2:end).';
    clock_offset=X_estimated(7);
    clock_drift=X_estimated(8);

    innovation=[this_pesudo_range - r_aj_estimated_norm - clock_offset;
    this_pesudo_range_rate - r_aj_estimated_norm_dot - clock_drift];

    % Update system state
    X_new=X_estimated+K*innovation;
    P_new=(eye(8)-K*H)*P_estimated;

    % prepare for next iteration
    X_old=X_new;
    P_old=P_new;

    % Result data collection
    [L_b,lambda_b,h_b,v_eb_n] = pv_ECEF_to_NED(X_new(1:3),X_new(4:6));
    Latitude_estimate=L_b*rad_to_deg;
    Longitude_estimate=lambda_b*rad_to_deg;

    record(t,1) = (t-1)*0.5;
    record(t,2:4)=[Latitude_estimate,Longitude_estimate,h_b];
    record(t,5:7)=v_eb_n.';

    end
    % csvwrite('GNSSresult.csv',record);
    writematrix(record,'GNSS_solution.csv');
```

1.2 GNSS_ini.m

```
% Define some constants
GNSS_Define_Constants;


% Read data from files
data1 = readmatrix('Pseudo_ranges.csv');
data2 = readmatrix('Pseudo_range_rates.csv');
[m,n]=size(data1);
m=m-1;
n=n-1;
sid=data1(1,2:end).'; % Satellites ID
tid=data1(2:end,1);    % Frames ID


% Initialise system state and error covariance matrix
X0 = zeros(8,1);
P0 =  zeros(8);

```


CONTINUED

```
P0(1,1) = 1^2;
P0(2,2) = 1^2;
P0(3,3) = 1^2;
P0(4,4) = 0.1^2;
P0(5,5) = 0.1^2;
P0(6,6) = 0.1^2;
P0(7,7) = 1000000^2;
P0(8,8) = 200^2;
```

1.3 GNSS_get_EFEC_From_GNSS.m

```
function r_ea_all=GNSS_get_EFEC_From_GNSS(data)
% input data must be a 2xn matrix with n representing the number of
% satellite, the first row is satellites' id and first column is the
% time

%% Variables definition
% r_ej: distance vector between earth and jth satellite
% r_eu: distance vector between earth and user
% r_aj: distance vector between user and jth satellite
% r_aj_norm: % absolute distance between user and jth satellite

% Data stripping, storage to different variables
[¬,n]=size(data);
n=n-1;
satellites_id=data(1,2:end).';
time_id=data(2:end,1);

Pseudo_Range=data(2,2:end).';


% Define Some constants and container variables.
c = 299792458; % Speed of light in m/s
omega_ie = 7.292115E-5;  % Earth rotation rate in rad/s
r_eu=zeros(3,1);
r_ej=zeros(3,n);
u_aj=zeros(3,n);
r_aj=zeros(3,n);
r_aj_norm=zeros(1,n);
r_aj_estimate_norm=zeros(n,1);
X_new=zeros(4,1);
X_old=zeros(4,1);

% Main iterations
for iterations=1:1:1000
for i=1:1:n
j=satellites_id(i);
time=time_id(1);

[r_ej(:,i),¬] = Satellite_position_and_velocity(time,j); % range ...
    vector of earth to satellite
r_aj(:,i)=r_ej(:,i)-r_eu;
```

```
        r_aj_norm(1,i)=norm(r_aj(:,i),2);
        C=[1,                    omega_ie*r_aj_norm(1,i)/c,   0;
        -omega_ie*r_aj_norm(1,i)/c,  1,                  0;
        0,                  0,                  1];
        media=C*r_ej(:,i)-r_eu;
        r_aj_estimate_norm(i)=sqrt(media.'*media);      % predict the ...
            ranges from the approximate user position to each satellite.
        media=C*r_ej(:,i)-r_eu;
        u_aj(:,i)=media/r_aj_estimate_norm(i);          %line of sight ...
            unit vector. In other words, unit vector between user and ...
            equlivent satellite.
    end


    ΔZ=Pseudo_Range-r_aj_estimate_norm-X_old(4,1); % calculate ...
        innovation
    H=[-u_aj.',ones(n,1)];                              % measurement ...
        matrix
    X_new=X_old+inv(H.'*H)*H.'*ΔZ;
    r_eu=X_new(1:3);
    X_old=X_new;
end
r_ea_all=X_new;
end
```

1.4 GNSS_Define_Constants.m

```
%Define_Constants
%SCRIPT This defines a number of constants for your use
% Edited 07/02/23 by Jian Zhou
% Created 16/11/16 by Paul Groves
% Copyright 2016, Paul Groves
% License: BSD; see license.txt for details

% Constants
deg_to_rad = 0.01745329252; % Degrees to radians conversion factor
rad_to_deg = 1/deg_to_rad; % Radians to degrees conversion factor
c = 299792458; % Speed of light in m/s
omega_ie = 7.292115E-5;  % Earth rotation rate in rad/s
Omega_ie = Skew_symmetric([0,0,omega_ie]);
R_0 = 6378137; %WGS84 Equatorial radius in meters
e = 0.0818191908425; %WGS84 eccentricity

Sa=0.001;   % acceleration PSD
Scphi=0.01; % Clock phase PSD
Scf=0.04;   % Clock frequency PSD
T=0.5;      % Time interval
% Ends
```


CONTINUED

2 Dead Reckoning

2.1 DR.m

```
clear;
Define_Constants;
data =  readmatrix('Dead_reckoning.csv');
% seconds fowrward_speed heading_in_degrees
%update interval
t_s = 0.5;
%height at 0
h = 39.2043;
%longitude and latitude at 0
L_ini = 51.5092543897043*deg_to_rad;
lambda_ini = -0.161045151548226*deg_to_rad;
%correct the speed with known errors
v_cf = [];
for i = 1:851
if data(i,4)≥0 && data(i,5)≥0
v_cf(i,1) = (data(i,4)-(0.05 + 0.02 + ...
    data(i,4)*0.03)+data(i,5)-(0.05 + 0.02 + data(i,5)*0.03))/2;
elseif data(i,4)≥0 && data(i,5)<0
v_cf(i,1) = (data(i,4)-(0.05 + 0.02 + ...
    data(i,4)*0.03)+data(i,5)+(0.05 + 0.02 - data(i,5)*0.03))/2;
elseif data(i,4)<0 && data(i,5)<0
v_cf(i,1) = (data(i,4)+(0.05 + 0.02 - ...
    data(i,4)*0.03)+data(i,5)+(0.05 + 0.02 - data(i,5)*0.03))/2;
else
v_cf(i,1) = (data(i,4)+(0.05 + 0.02 - ...
    data(i,4)*0.03)+data(i,5)-(0.05 + 0.02 + data(i,5)*0.03))/2;
end

end

%calculate the statement at time 0
v_forward = v_cf(1,1)/2;
v_n_ini   = v_forward*cos(data(1,7)*deg_to_rad);
v_e_ini   = v_forward*sin(data(1,7)*deg_to_rad);
v_ini     = [v_n_ini+(0.05 + 0.02 - v_n_ini*0.03);v_e_ini-(0.05 ...
    + 0.02 + v_e_ini*0.03)];

L_p = L_ini;
lambda_p = lambda_ini;
v_p = v_ini;

record = zeros(851,5);
record(1,1)= 0 ;
record(1,2)=51.5092543897043;
record(1,3)=-0.161045151548226;
record(1,4:5)=v_ini.';


psi_p = data(1,7)*deg_to_rad; %previous heading
```

```

for i = 1:850


%   psi_c = data(i+1,7)*deg_to_rad; %current heading

% Correct the gyroscope with known errors
if data(i+1,6)≥0
omega = data(i+1,6) - (1*deg_to_rad + (0.01 + 0.001)*data(i+1,6) ...
    + 0.0002);
else
omega = data(i+1,6) + (1*deg_to_rad - (0.01 + 0.001)*data(i+1,6) ...
    + 0.0002);
end
%smooth heading
w = (1 + (0.01 + 0.001)*data(i+1,6)*rad_to_deg + ...
    0.0002*rad_to_deg)*0.5/4;%compute weight
psi_c = w*data(i+1,7)*deg_to_rad + (1-w)*(psi_p+t_s*omega);
%compute the overall speed
v_ned = 0.5*[cos(psi_c)+cos(psi_p);sin(psi_c)+sin(psi_p)]*v_cf(i,1);

[R_N,R_E]= Radii_of_curvature(L_p);
%longitude and latitude
L_c = L_p+(v_ned(1)*t_s)/(R_N+h);
lambda_c = lambda_p+(v_ned(2)*t_s)/((R_E+h)*cos(L_c));


%instantaneous velocity
v = 1.7*v_ned-0.7*v_p;

record(i+1,1)= i*t_s ;                 %time
record(i+1,2)=L_c*rad_to_deg;          %latitude
record(i+1,3)=lambda_c*rad_to_deg;     %longitude
record(i+1,4:5)=v.';                   %velocity in North and East
record(i+1,6)= psi_c ;                 %Heading
L_p = L_c;
lambda_p = lambda_c;
v_p = v;
psi_p = psi_c;
end

writematrix(record,'DR_solution.csv');
%graph of longitude and latitude
%x =  record(:,3);
%y =  record(:,2);
%figure;
%set(gcf,'Color',[0.9 0.9 0.9]);
%plot(x,y);
%xlabel('longitude');
%ylabel('latitude');
%title('plot of the latitude and longitude of the lawnmower');
%
%
% %graph of velocity
% x  =  record(:,1);
```

```
% y1 =  record(:,4);
% y2 =  record(:,5);
% figure;
% set(gcf,'Color',[0.9 0.9 0.9]);
% plot(x,y1,x,y2,'--');
% legend('velocity in North','velocity in ...
   East','Location','northeast','Orientation','vertical');
% xlabel('time(s)');
% ylabel('velocity(m/s)');
% title('plot of the velocity');
%
%
% %graph of heading
% x  =  record(:,1);
% y  =  record(:,6);
%
% figure;
% set(gcf,'Color',[0.9 0.9 0.9]);
% plot(x,y);
% xlabel('time(s)');
% ylabel('heading(deg)');
% title('plot of the heading');
```

3 Basic Loosely-coupled Open-looped Integration

3.1 Integration.m

```
% update GNSS solution
GNSS
clear
%update DR solution
DR
clear
%read DR solution and GNSS solution
DR_solution = readmatrix('DR_solution.csv');
GNSS_solution = readmatrix('GNSS_solution.csv');
%define cdonstant
Define_Constants
%define initial states
x = zeros(4,1); % v_n v_e l lamda
% define initial error covariance
sigma_v = 0.1;% velocity uncertainty
sigma_r = 1;% position uncertainty
L_b = GNSS_solution(1,2)*deg_to_rad;
lambda_b = GNSS_solution(1,3)*deg_to_rad;
h_b = GNSS_solution(1,4);
[R_N,R_E]= Radii_of_curvature(L_b);
P = diag([sigma_v^2 sigma_v^2 ...
sigma_r^2/(R_N+h_b)^2 sigma_r^2/(R_E+h_b)^2*cos(L_b)^2]);

result = zeros(851,5);
```

```
for i = 1:851
%Apply Kalman filter
%obtain position information
L_b = GNSS_solution(i,2)*deg_to_rad;
lambda_b = GNSS_solution(i,3)*deg_to_rad;
h_b = GNSS_solution(i,4);
[R_N,R_E]= Radii_of_curvature(L_b);

% calculate transition matrix
t_s = 0.5;
phi = eye(4);
phi(3,1) = t_s/(R_N+h_b);
phi(4,2) = t_s/((R_E+h_b)*cos(L_b));

% calculate noise covariance matrix
s_dr = 0.01;%DR velocity error power spectral density (PSD),;
Q = zeros(4);
Q(1,1) = s_dr*t_s;
Q(2,2) = Q(1,1);
Q(3,3) = (1/3)*(s_dr*t_s^3)/(R_N+h_b)^2;
Q(4,4) = (1/3)*(s_dr*t_s^3)/((R_E+h_b)^2*cos(L_b)^2);
Q(1,3) = (1/2)*(s_dr*t_s^2)/(R_N+h_b);
Q(3,1) = Q(1,3);
Q(2,4) = (1/2)*(s_dr*t_s^2)/((R_E+h_b)*cos(L_b));
Q(4,2) = Q(2,4);
%propagate state
x_n = phi*x;
%propagate error covariance
P_n = phi*P*phi.' + Q;
%compute the error covariance matrix
H = [0 0 -1 0;
0 0 0 -1;
-1 0 0 0;
0 -1 0 0];

% GNSS measurment noise covariance matrix
sigma_gr = 5;% position measuremnet error standard deviation
sigma_gv = 0.02;% velocity measuremnet error standard deviation
h_c = GNSS_solution(i,4);
L_c = GNSS_solution(i,2)*deg_to_rad;
lambda_c = GNSS_solution(i,3)*deg_to_rad;
[R_N,R_E]= Radii_of_curvature(L_c);
R = diag([sigma_gr^2/(R_N+h_c)^2 ...
    sigma_gr^2/((R_E+h_c)^2*cos(L_c)^2) ...
sigma_gv^2 sigma_gv^2]);

%Compute the Kalman gain matrix
K = P_n*H.'*inv(H*P_n*H.'+R);
%measurement innovation vector
%collect the measuremnt value
v_n_G = GNSS_solution(i,5);
v_e_G = GNSS_solution(i,6);
DR_record = DR_solution(i,2:5).';
DR_record(1) = DR_record(1)*deg_to_rad;
DR_record(2) = DR_record(2)*deg_to_rad;
```


CONTINUED

```matlab
    %calculate innvation vector
    Δ_z = [L_c;lambda_c;v_n_G;v_e_G]-DR_record-H*x_n;
    %update the state estimate
    x_p = x_n + K*Δ_z;
    %update the error covariance matrix
    P_p = (eye(4)-K*H)*P_n;
    %update DR solution with update state estimate
    x_p_record = x_p;
    %translate rad to degree for state value
    x_p_record(3) = x_p_record(3)*rad_to_deg;
    x_p_record(4) = x_p_record(4)*rad_to_deg;
    %record the corrected DR solution
    result(i,1) = (i-1)*t_s;
    result(i,2:5) = DR_solution(i,2:5).' + H*x_p_record;
    %update state and error covariance mateix for next epoch
    x = x_p;
    P = P_p;

    end
    %store output in csv file
    %add heading solution from original DR solution
    Motion_profile = [result DR_solution(:,6)];
    %store the results in csv file
    writematrix(Motion_profile,'Motion_Profile.csv');

    %posotion
    figure;
    plot(GNSS_solution(:,3),GNSS_solution(:,2));%
    hold on
    %graph of longitude and latitude
    plot(DR_solution(:,3),DR_solution(:,2));
    hold on
    plot(result(:,3),result(:,2));%
    hold on
    plot(result(1,3),result(1,2),'o');
    hold on
    plot(result(851,3),result(851,2),'o');
    title('plot of the latitude and longitude of the lawnmower');
    xlabel('longitude(degrees)');
    ylabel('latitude(degrees)');
    legend('GNSS','DR','Integrate','start points','end point')
    grid on;
    hold off;
    %north velocity
    figure;
    t = GNSS_solution(:,1);
    plot(t,GNSS_solution(:,5));%
    hold on
    %graph of longitude and latitude
    plot(t,DR_solution(:,4));
    hold on
    plot(t,result(:,4));%
    title('plot of the north velocity');
    xlabel('time(s)');
    ylabel('North velocity(m/s)');
```

```
legend('GNSS','DR','Integrate')
grid on;
% East velocity
figure;
t = GNSS_solution(:,1);
plot(t,GNSS_solution(:,6));%
hold on
%graph of longitude and latitude
plot(t,DR_solution(:,5));
hold on
plot(t,result(:,5));%
title('plot of the east velocity');
xlabel('time(s)');
ylabel('East velocity(m/s)');
legend('GNSS','DR','Integrate')
grid on;
%heading
figure;
t = GNSS_solution(:,1);
plot(t,DR_solution(:,6));
title('plot of the heading');
xlabel('time(s)');
ylabel('heading(degrees)');
grid on;
```