Simon Honigmann

Sensor Orientation

October 12, 2018

Lab 4: Inertial Navigation in 2D / Nominal Signal

# Simulated Inertial Navigation in 2D:

In this lab exercise, the goal was to develop simulated IMU sensor readings for a simplified 2D inertial system and use the simulation to highlight the limitations of integrated localization techniques. Sensors were assumed to be ideal in this case, so that at any given point in time, the exact rotational velocities and accelerations in the vehicle’s frame are known. A simple circular path was selected to simplify the equations.

The navigation was resolved for two different sensor sampling frequencies, 10Hz and 100Hz, and two different integration techniques, rectangular and trapezoidal. The trajectories and errors were plotted for each combination below.

It can be noted that the errors for 100 Hz are approximately an order of magnitude smaller than those at 10 Hz. The Trapezoidal integration method has a smaller error by a factor of approximately 2 compared to the Rectangular method. Because there is never any correction of integrated error, the position localization error would continue to grow as the vehicle performed more rotations of the circle, steadily spiraling away from the original circle. This is demonstrated in Figure 5.

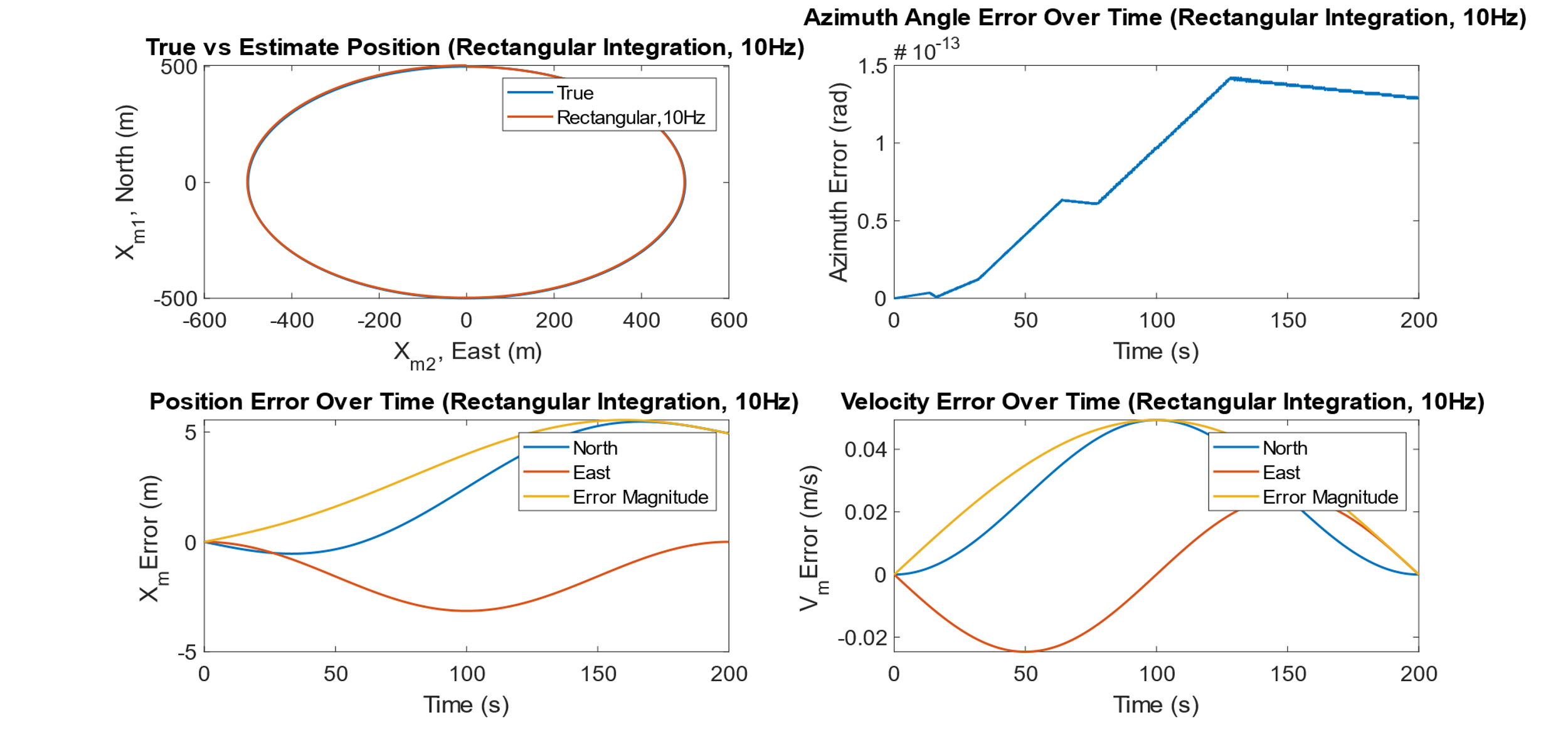


Figure 1: Trajectory and Error Plots Over One Rotation for Rectangular Integration at 10 Hz sampling frequency

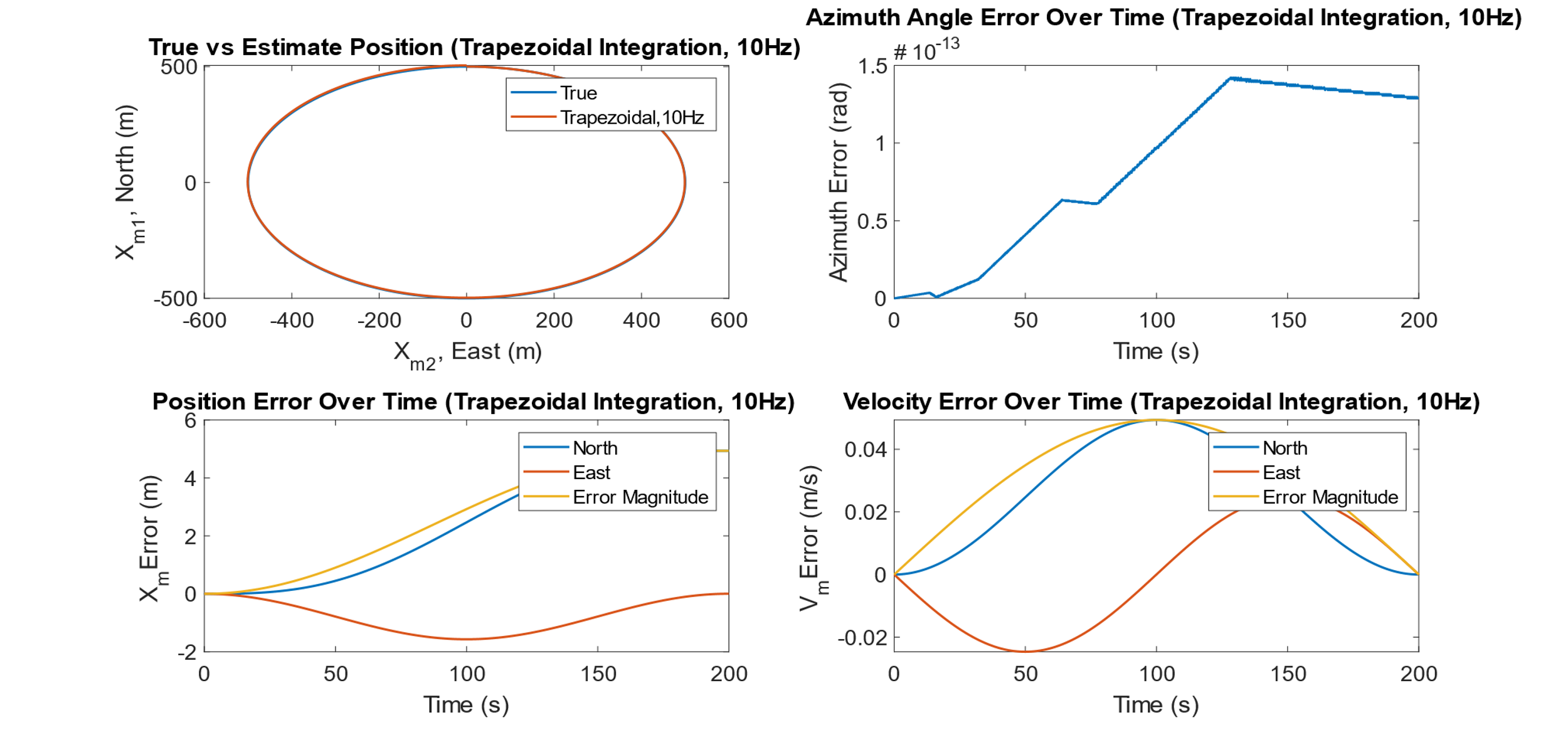


Figure 2: Trajectory and Error Plots Over One Rotation for Trapezoidal Integration at 10 Hz sampling frequency

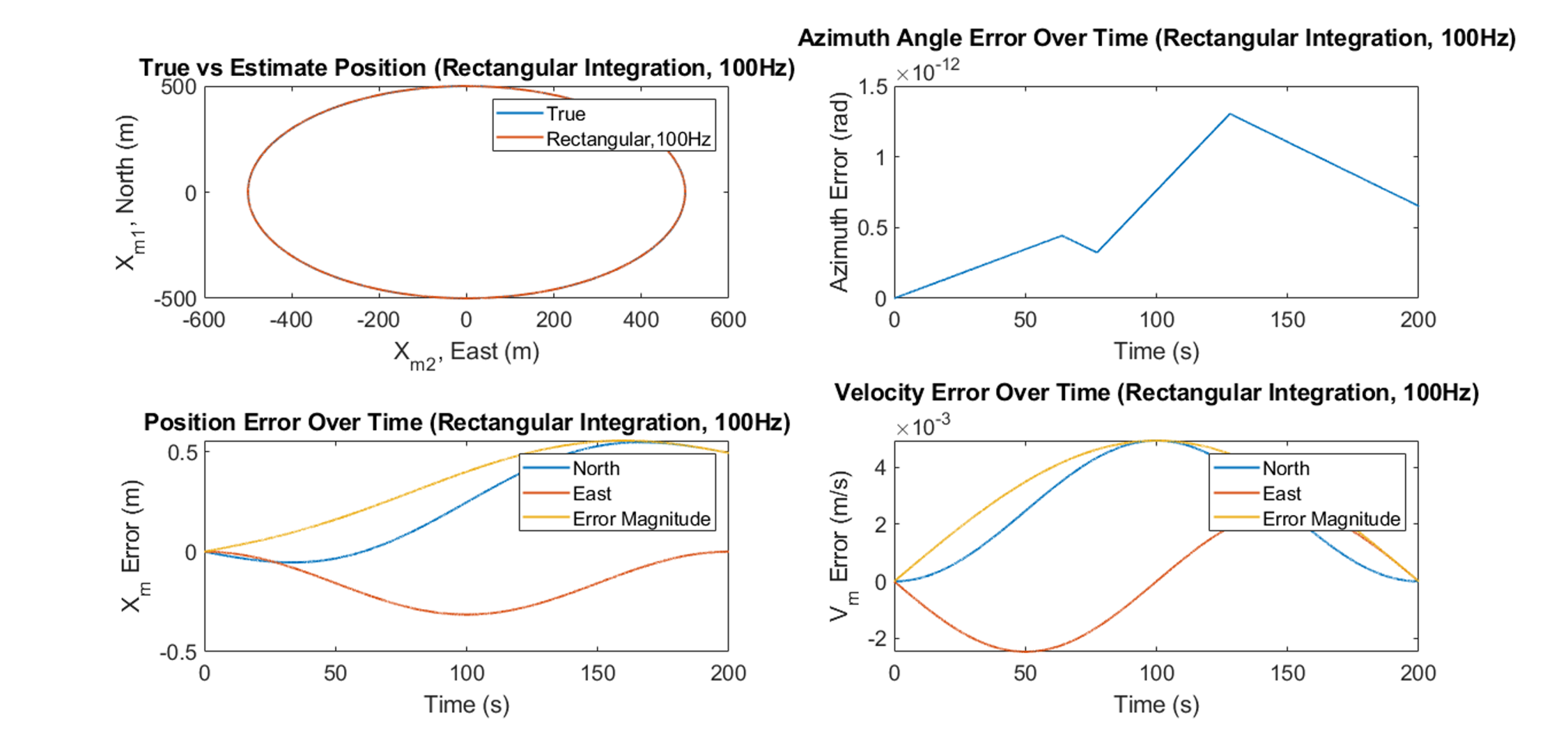


Figure 3: Trajectory and Error Plots Over One Rotation for Rectangular Integration at 100 Hz sampling frequency

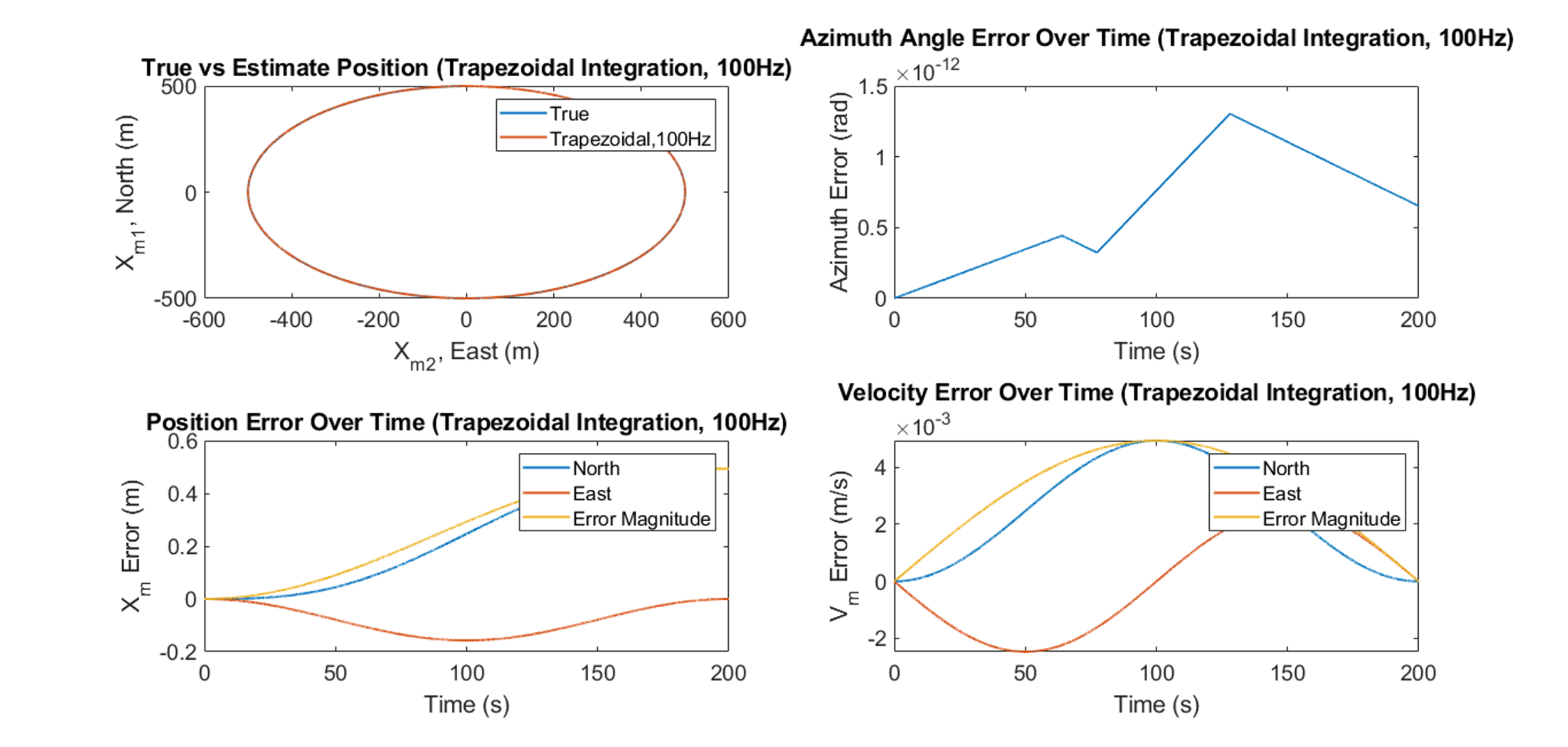


Figure 4: Trajectory and Error Plots Over One Rotation for Trapezoidal Integration at 100 Hz sampling frequency

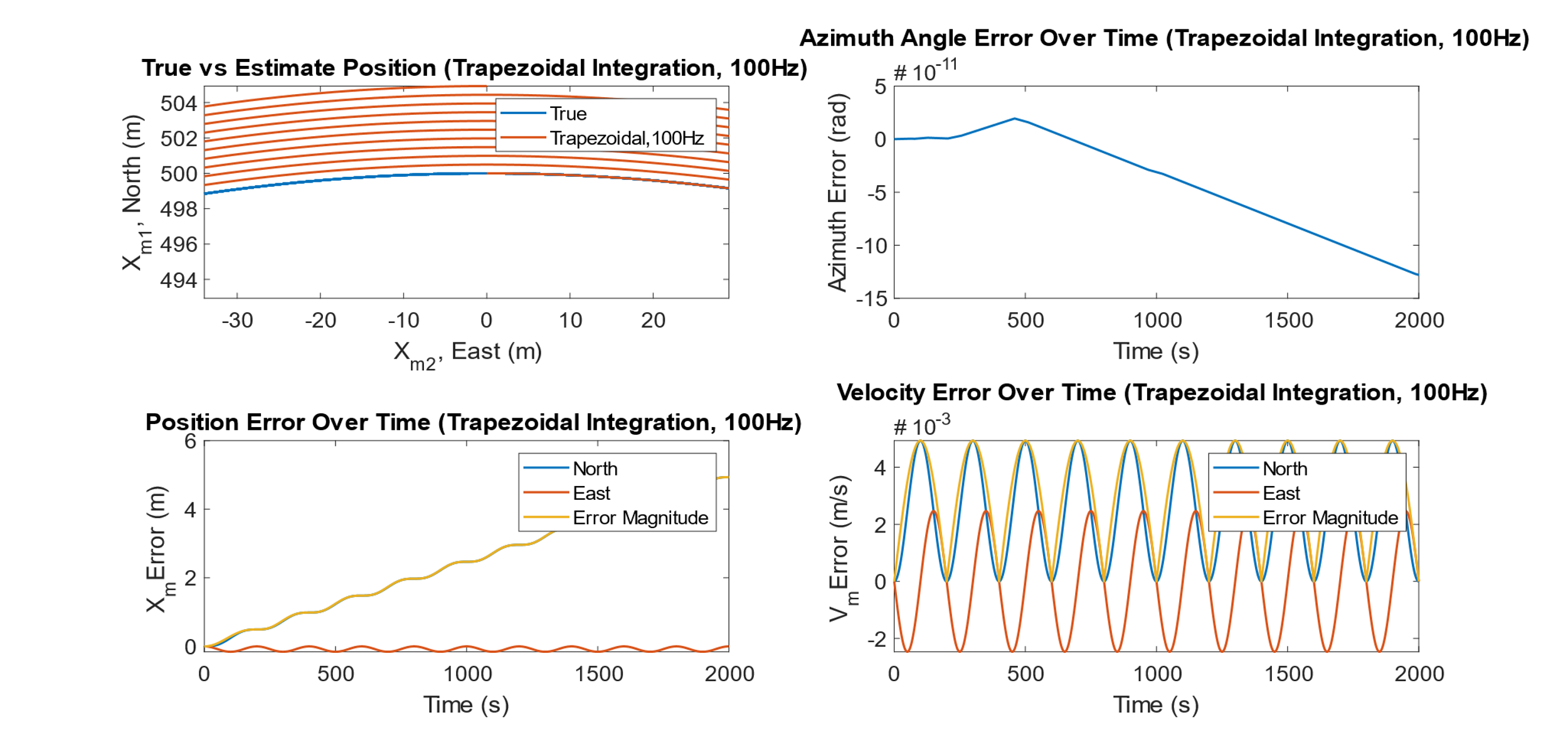


Figure 5: Trajectory and Error Plots Over 10 Rotations for Trapezoidal Integration at 100 Hz

# Questions:

1. What are the maximum committed errors in x & y coordinates after one revolution for:
   1. 10 Hz and 1st order integration
   2. 10 Hz and 2nd order integration
   3. 100 Hz and 1st order integration
   4. 100 Hz and 2nd order integration

The table below highlights the maximum and average errors for each tested combination.

Table 1: Average and Maximum Errors for Different Localization Methods

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Sampling Frequency (Hz)** | **Integration Order** | **East (X) Error [m]** | | **North (Y) Error [m]** | |
| **Average** | **Maximum** | **Average** | **Maximum** |
| 10 | 1st | 1.57 | 3.14 | 2.68 | 5.47 |
| 10 | 2nd | 0.78 | 1.57 | 2.47 | 4.93 |
| 100 | 1st | 0.16 | 0.31 | 0.27 | 0.55 |
| 100 | 2nd | 0.08 | 0.16 | 0.25 | 0.49 |

1. Which integration method would you recommend to use?

Typically, if your controller can handle a slightly increased computational and memory load, trapezoidal integration is preferred from a performance standpoint. Only a small increase in memory is required and would only take a few additional clock cycles assuming the variable sizes are within the processor word size. In general, physical system dynamics have a bandwidth considerably smaller than that the sampling interval used so processing is rarely a limiting factor. And so, for a given sampling rate (often, but not always, limited by sensors and not the microcontroller), a higher order integration method resulting in a more accurate estimate is advantageous.

# Appendix A: MATLAB Code

%% Simon Honigmann

% Sensor Orientation

% Lab 4: Inertial Navigation in 2D / Nominal Signal

% 10/26/18

%% Cleanup

clc;

%close all;

%% Lab Formatting:

set(groot,'DefaultAxesFontSize',14);

set(groot,'DefaultLineLineWidth',1.5);

%% Definitions:

% m = 2D mapping frame (non-accelerating, non-rotating)

% x\_m1 = north

% x\_m2 = east

% m frame polar coordinates

% psi = polar angle

% r = radius

% b = body frame

% alpha = azimuth (yaw) angle

% x\_b1 = tangential direction

% x\_b2 = radial direction (inwards)

%% Constants

r = 500; % Circle radius: 500 m, Angular speed ? = ?/100;

omega = pi/100; % angular speed = pi/100

%be clever and loop through all possible combinations here...

for k=1:4

if k==1

sampling\_rate = 10;

method = 'Rectangular';

elseif k==2

sampling\_rate = 10;

method = 'Trapezoidal';

elseif k==3

sampling\_rate = 100;

method = 'Rectangular';

elseif k==4

sampling\_rate = 100;

method = 'Trapezoidal';

end

sample\_rate\_text = num2str(sampling\_rate,3);

num\_rotations = 1; %number of time the vehicle goes around the circle. increase to better show divergence

%% Assumptions/Constraints

g = 0; %neglect gravity

dr = 0; %constant radius of motion in m frame

ddr = 0;

dpsi = omega; %constant angular velocity in m frame

ddpsi = 0;

%% Initial Conditions

psi\_0 = 0; % Initial position: on North axis

alpha\_0 = psi\_0 + pi/2;

x\_0 = [r,0];

dx\_0 = [0,omega\*r]; % Initial velocity: north-axis: 0, east-axis: ? · radius

% 1. Simulate the nominal (i.e. errorless) measurements for a gyro and 2 orthogonal

% accelerometers in a body-frame when subjected to uniform circular motion in a plane,

% which coordinate system is spanned by the North and East axes and represents a 2Dinertial-frame.

required\_time = 2\*num\_rotations\*pi/omega; %time for 1 full rotation [s]

t = (0:1/sampling\_rate:required\_time)'; %time vector, [s]

samples = (1:length(t))';

psi = (psi\_0:num\_rotations\*2\*pi/(length(t)-1):num\_rotations\*2\*pi)'; %polar angle, [rad]

alpha = psi + pi/2; %azimuth angle, Initial azimuth: 90? (towards x-accelerometer)

%use this line instead to have azimuth angle wrap at 2pi

%%alpha = mod(psi + pi/2,2\*pi); %azimuth angle, Initial azimuth: 90? (towards x-accelerometer)

x\_m = [r\*cos(psi),r\*sin(psi)]; % map frame, [north, east]

dx\_m = [dr.\*cos(psi)-r.\*sin(psi).\*dpsi , r.\*cos(psi).\*dpsi + dr.\*sin(psi)];

%(definitely could have simplified this...)

ddx\_m = [ddr.\*cos(psi) - dr.\*sin(psi).\*dpsi - (dr.\*sin(psi).\*dpsi + r.\*cos(psi).\*dpsi.^2 + r.\*sin(psi).\*ddpsi), ... %ddx\_m1

dr.\*cos(psi).\*dpsi - r.\*sin(psi).\*dpsi.^2 + r.\*cos(psi).\*ddpsi + ddr.\*sin(psi)+dr.\*cos(psi).\*dpsi ]; %ddx\_m2

dalpha = dpsi\*ones(length(t),1); %deriving alpha wrt time = deriving psi wrt time

v\_b = [ones(length(t),1)\*omega\*r,zeros(length(t),1)]; % got lazy... this works here because omega and r are constant. Would need to change this line if that was not the case

a\_b = [zeros(length(t),1),ones(length(t),1)\*omega^2\*r]; %got lazy... this works here because omega and r are constant. Would need to change this line if that was not the case

gyro = dalpha; %ideal gyroscope

accel = a\_b; %ideal accelerometer, f

% 2. Considering the initial conditions to be known, apply strapdown inertial navigation to

% calculate the trajectory parameters (i.e. the attitude (azimuth) and 2D velocity and

% position vectors, respectively).

%[Rmb,Rbm] = RotationMatrix(alpha); %define rotation matrices for all angles, alpha

%initialize variables with initial conditions

alpha\_sd = zeros(length(alpha),1); %strapdown azimuth

alpha\_sd(1) = alpha\_0;

v\_sd = zeros(length(t),2); %strapdown velocity, map frame

v\_sd(1,:) = dx\_0;

x\_sd = v\_sd;

x\_sd(1,:) = x\_0;

%doing this with a loop. we'll see if i regret it later

for i=2:length(t)

if strcmp(method,'Rectangular')

alpha\_sd(i) = alpha\_sd(i-1)+gyro(i)\*(t(i)-t(i-1)); %strapdown azimuth

[~,Rbm] = RotationMatrix(alpha\_sd(i));

Rbm = squeeze(Rbm);

v\_sd(i,:) = (v\_sd(i-1,:)'+Rbm\*accel(i,:)'\*(t(i)-t(i-1)))';

x\_sd(i,:) = (x\_sd(i-1,:)'+v\_sd(i,:)'\*(t(i)-t(i-1)))';

elseif strcmp(method, 'Trapezoidal')

alpha\_sd(i) = alpha\_sd(i-1)+1/2\*(gyro(i)+gyro(i-1))\*(t(i)-t(i-1));

[~,Rbm] = RotationMatrix(alpha\_sd(i));

Rbm = squeeze(Rbm);

v\_sd(i,:) = (v\_sd(i-1,:)'+1/2\*(Rbm\*accel(i,:)'+Rbm\*accel(i-1,:)')\*(t(i)-t(i-1)))';

x\_sd(i,:) = (x\_sd(i-1,:)'+1/2\*(v\_sd(i,:)+v\_sd(i-1,:))'\*(t(i)-t(i-1)))';

else

break %err. method invalid

end

end

% 3. Resolve the navigation (add 2) equations for two sampling rates (i.e. 10Hz, 100Hz) and

% using two integration methods (i.e. 1st order and higher order) for one revolution (i.e.

% complete circle).

% 4. For each solution compare the evolution of obtained trajectory parameters to the true

% values and plot the errors in a) Azimuth (unit: degree), b) Velocity (m/s) and c) Position

% (m). Evaluate the magnitude of maximum committed errors in coordinates to answer the

% questions below.

%% Plotting Trajectories

figure(k);

%position

subplot(2,2,1);

plot(x\_m(:,2),x\_m(:,1));

hold on;

xlabel('X\_m\_2, East (m)');

ylabel('X\_m\_1, North (m)');

title(['True vs Estimate Position (',method,' Integration, ',sample\_rate\_text,'Hz)']);

plot(x\_sd(:,2),x\_sd(:,1));

legend('True',strcat(method,', ',sample\_rate\_text,'Hz'));

%% Calculate Errors

err\_x = x\_sd-x\_m;

err\_alpha = alpha\_sd - alpha;

err\_v = v\_sd - dx\_m;

err\_x = [err\_x,sqrt(err\_x(:,1).^2+err\_x(:,2).^2)]; %add third column for position error magnitude

err\_v = [err\_v,sqrt(err\_v(:,1).^2+err\_v(:,2).^2)]; %add third column for position error magnitude

err\_table(k,:) = [mean(abs(err\_x(:,2))),max(abs(err\_x(:,2))),mean(abs(err\_x(:,1))),max(abs(err\_x(:,1)))];

%% Plotting Errors

%figure(2);

%position

subplot(2,2,3);

plot(t,err\_x(:,1));

hold on;

plot(t,err\_x(:,2));

plot(t,err\_x(:,3));

xlabel('Time (s)');

ylabel('X\_m Error (m)');

title(['Position Error Over Time (',method,' Integration, ',sample\_rate\_text,'Hz)']);

legend('North','East','Error Magnitude');

%azimuth

subplot(2,2,2);

plot(t,err\_alpha);

hold on;

xlabel('Time (s)');

ylabel('Azimuth Error (rad)');

title({['Azimuth Angle Error Over Time (',method,' Integration, ',sample\_rate\_text,'Hz)'],' '});

%velocity v1\_m

subplot(2,2,4);

plot(t,err\_v(:,1));

hold on;

plot(t,err\_v(:,2));

plot(t,err\_v(:,3));

xlabel('Time (s)');

ylabel('V\_m Error (m/s)');

if(sampling\_rate > 10)

title({['Velocity Error Over Time (',method,' Integration, ',sample\_rate\_text,'Hz)'],' '});

else

title({['Velocity Error Over Time (',method,' Integration, ',sample\_rate\_text,'Hz)']});

end

legend('North','East','Error Magnitude');

end

%% Functions

function [Rmb,Rbm] = RotationMatrix(alpha)

Rmb = zeros(length(alpha),2,2); %preallocate memory

Rbm = Rmb;

for i=1:length(alpha)

Rmb(i,:,:) = [cos(alpha(i)),sin(alpha(i)); -sin(alpha(i)),cos(alpha(i))];%transformation matrix R(m->b)

Rbm(i,:,:) = squeeze(Rmb(i,:,:))'; %transformation matrix R(b->m)

end

end