Final Project

A final project submitted in partial fulfillment of
the requirements for the
degree Bachelor of Science (B.Sc.)
at the Ariel University of Samaria
The Department of Mathematics

By
Ariel David
Eliyahu Abraham
Ilan Kitzin

Under the supervision of
Dr. Oleg Kupervasser
Dr. Roman Yavich

January 2021

אוניברסיטת אריאל בשומרון
הפקולטה למדעי הטבע
המחלקה למתמטיקה

פרויקט גמר

פרויקט גמר שהוגש במילוי חלקי של של הדרישות לתואר ראשון במדעים
באוניברסיטת אריאל שבשומרון

נכתב ע"י
אריאל דוד
אליהו אברהם
אילן קיציו

בהנחיית
ד"ר קופרווסר
ד"ר רומן יביץ

ינואר 2021

Ariel University of Samaria
The Faculty of Natural Sciences
The Department of Mathematics

Final Project

Ariel David
Eliyahu Abraham
Ilan Kitzin

A final project submitted in partial fulfillment of the requirements for the degree Bachelor of Science (B.Sc.)

Under the supervision of
Dr. Oleg Kupervasser
Dr. Roman Yavich

Signature of student: _____ Date: _____
Signature of supervisor: _____ Date: _____
Signature of chairperson of the
committee for graduate studies: _____ Date: _____

January 2021

תקציר

ראיה ממוחשבת היא ענף מחקר מרכזי של מדעי המחשב,
העוסק בעיבוד אוטומטי של תמונות המבוססות על העולם האמיתי, במטרה לחלץ ולפרש מידע חזותי הטמון בהם. מערכת ניווט אינרציאלית־מערכת שכוללת אוסף של חיישנים ובעיקר מדי תאוצה וגירוסקופ.
במערכת אינרציאלית אנו מודדים תאוצה ומהירות זוויתית ועל בסיס זה מודדים שינוי של קואורדינטות וגם סיבוב ביחס למצב הקודם.
זאת מערכת אינרציאלית עם שישה צירים ויכולים להוסיף מד גובה ומד שדה מגנטי אז זה יהיה עשרה צירים.
אלגוריתם חיפוש צילום זהו אלגוריתם הפועל על ידי שיטות במתמטיקה (גאומטריה פרואקטיבית ומצלמת חריר) למציאת צילום של הקרקע.

Abstract

We present a project based on computer vision based navigation.
In this project we create a specific drone trajectory in six dimensions - three coordinates and three angles.
By simulation we get these six parameters with a drag error from a complex inertial system From six axes, a 3D accelerometer and a 3D gyroscope.
After a response we find the order of filming of the film from the camera that is on the drone that gets a kind of orthophoto of different ground based on a constant brightness assumption.

Contents

1 Introduction

Over the past decade there has been a rapid development in the understanding and modelling of the geometry of multiple views in computer vision. The theory and practice have now reached a level of maturity where excellent results can be achieved for problems that were certainly unsolved a decade ago, and often thought unsolvable. These tasks and algorithms include:
1. Given two images, and no other information, compute matches between the images, and the 3D position of the points that generate these matches and the cameras that generate the images.
2. Given three images, and no other information, similarly compute the matches between images of points and lines, and the position in 3D of these points and lines and the cameras.
3. Compute the internal calibration of a camera from a sequence of images of natural scenes (i.e. calibration "on the fly"). More practically, it is often not possible to calibrate cameras once-and-for-all; for instance where cameras are moved (on a mobile vehicle) or internal parameters are changed (a surveillance camera with zoom). Furthermore, calibration information is simply not available in some circumstances. Imagine computing the motion of a camera from a video sequence, or building a virtual reality model from archive film footage where both motion and internal calibration information are unknown. The achievements in multiple view geometry have been possible because of developments in our theoretical understanding, but also because of improvements in estimating mathematical objects from images. The first improvement has been an attention to the error that should be minimized in over-determined systems – whether it be algebraic, geometric or statistical. The second improvement has been the use of robust estimation algorithms (such as RANSAC), so that the estimate is unaffected by "outliers" in the data. Also these techniques have generated powerful search and matching algorithms. Many of the problems of reconstruction have now reached a level where we may claim that

they are solved. Such problems include:
(i) Estimation of the multifocal tensors from image point correspondences, particularly the fundamental matrix and trifocal tensors (the quadrifocal tensor having not received so much attention).
(ii) Extraction of the camera matrices from these tensors, and subsequent projective reconstruction from two, three and four views.
Other significant successes have been achieved, though there may be more to learn about these problems. Examples include:
(i) Application of bundle adjustment to solve more general reconstruction problems.
(ii) Metric (Euclidean) reconstruction given minimal assumptions on the camera matrices.
(iii) Automatic detection of correspondences in image sequences, and elimination of outliers and false matches using the multifocal tensor relationships

2 Theoretical Background

Computer Vision - is a major research branch of computer science, which deals with the automatic processing of images based on the real world, in order to extract and interpret visual information contained in them.
Inertial navigation system - a system that includes a collection of sensors and especially accelerometers and gyroscopes.
In an inertial system we measure acceleration and angular velocity and on this basis we measure a change of coordinates and also a rotation with respect to the previous state.
This is an inertial system with six axes and can add an altimeter and a magnetic field meter so it will be ten axes.
Photo Search Algorithm - An algorithm that works by methods in mathematics (proactive geometry and a pinhole camera) to find a photograph of the ground.
Selection * Orthophoto of the land - The orthophoto is a photo map that can be used as a background for planning and geographical applications in urban MMG systems, for monitoring and monitoring construction anomalies, for updating maps and geographical information layers and more.
Choosing a true route - We plan to develop a program that solves existing problems in identifying the position of a body in an image in computer vision.
With the help of computer vision and by a system of axes X, Y, Z we will get a griscope where we will get the angular velocity.
We will choose three angles: rotation angle - rotation angle, Alrod - angle of ascent and descent and roll.
Projective model of the camera-eyelet camera: A camera that creates a two-dimensional image from a three-dimensional object.
Optically, the principle of operation of the device is similar to the principle of operation of the camera, The camera obscura has been used since its invention as an aid used for research in optics and for understanding the mechanism of action of the eye.

The early version of the camera obscura consists of a closed cell that has a tiny hole in one of its walls.
Light rays that are reflected from the bone penetrate through the hole and create an inverted image in the opposite wall.
The tiny hole ensures that the source of one point in the figure is in the light reflected from one point on the surface of the bone.
Without the puncture each figure point would receive light from the reflection of many points on the bone and no clear figure would be obtained.

Goals:
Search:
A) Measurements of sensors from an inertial system of 10 axes per trajectory.
B) Images by using a pinhole camera, trajectory and maps in a photo search algorithm.
Accessories: Tablecloth, input, route and maps.

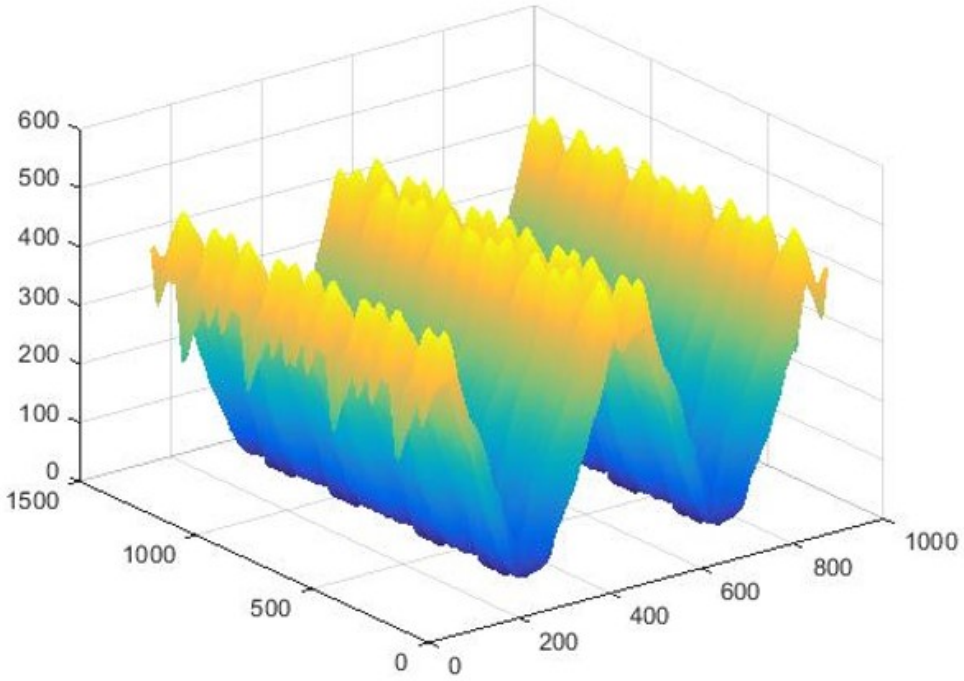
3 Project implementation

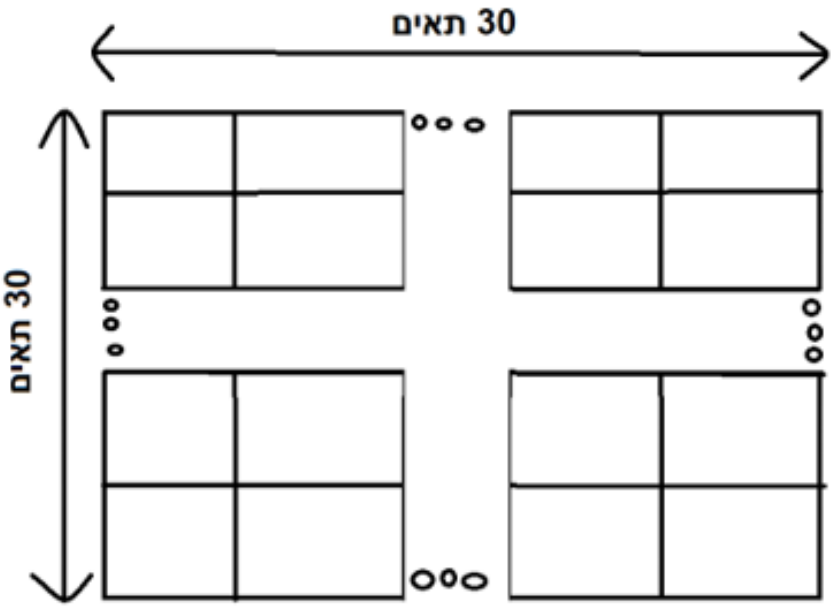
Step one - need to choose an orthophoto:
Elevation map and brightness maps (pictures) of the ground are defined by an array of points.
Step Two - To solve the problem we create a certain trajectory in six dimensions (three coordinates and three angles).
Define the trajectory by a set of points with a time interval Δt.
Step Three - Looking for measurements of sensors with 10-axis inertial system with time interval Δt:
Three-dimensional accelerometer, three-dimensional gyroscopes (angular velocity), three-dimensional Earth Earth magnetic field meter, altitude meter (air pressure).
Fourth stage - based on the six-dimensional route in the Harir camera (which creates a projective model) looking for photographs of space with time gain $\Delta t1$ ¿ Δt.

4 Methods

Input of the program 1: DTM-digital terrain map

Elevation map:
We divide space into 900 cells -30 rows and 30 columns.
Height map of each cell the same and looks like in the right part.
Height map of one cell:



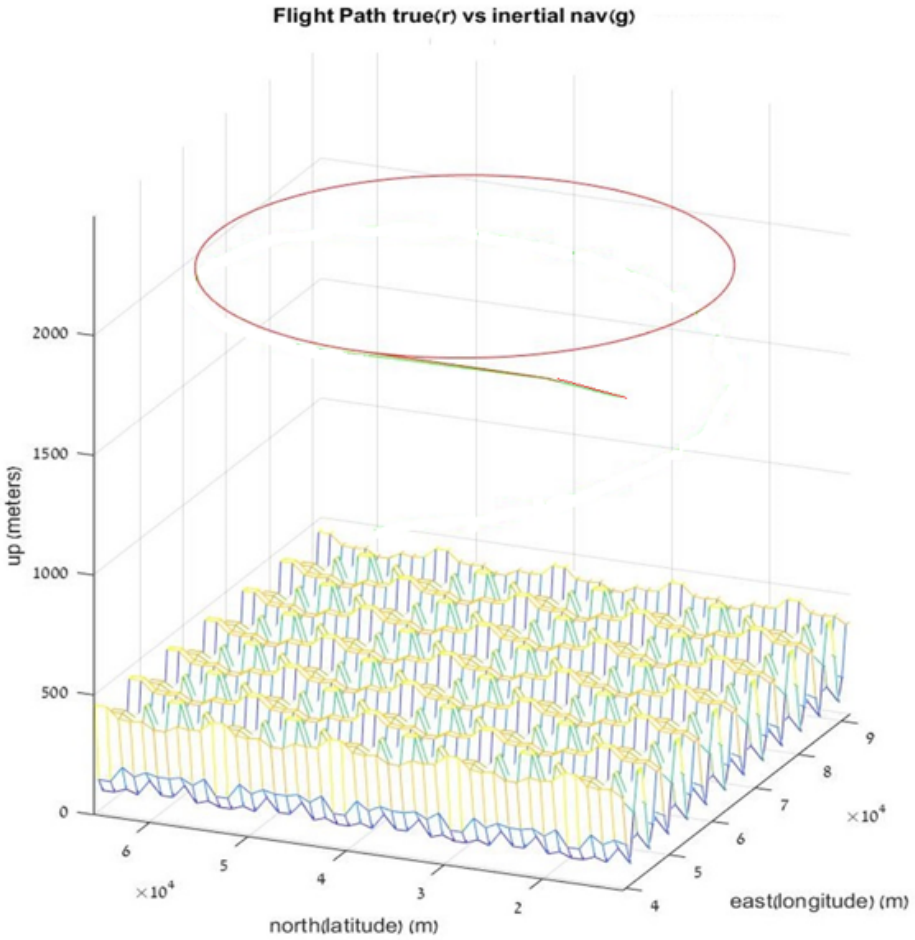


Input of the program 2: Brightness map

Brightness map (satellite image): We divide space into 900 cells -30 rows and 30 columns.
Brightness map of each cell the same and looks like in the right part.
Height map of each cell the same and looks like this:

Input of the 3: Flight path creation program

We use the **"Fly Profile Generator.m"** function to generate a route:
Every tenth of a second (Δt) we get velocity, x, y, z coordinates and an R rotation matrix.
The real route we chose is painted in red.
Below the route you can see the land of suitable suitable built-up land.



```
%%%%%%%%%%%%%%%%%gener%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%   Fly_Profile_Generator.m
%
% Generation of fly trajectory from driver's signals
%   program to show the use of PROGEN.M
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Block 1 s %%%%%%%%%%%%%%%%%%%%%%%%
grad2rad=pi/180;
C = [0 1 0; 1 0 0; 0 0 -1];
% %%%%%%%%%%%%%% INPUT %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% earthflg =1;
%
% lat=32*grad2rad;
% lon=35*grad2rad;
% hint=2000;
%
% initvel = [0 200 0];   % initial velocity
% initacc = [0 0 0];   % initial acceleration
%
% phi=0*pi/180;       % initial roll angle is zero
% theta=0*pi/180;     % initial pitch angle is zero
% psi=0*pi/180;       % initial yaw angle is zero (thus aircraft is pointed north)
%pitch_angle=asin(1000/((200+10)/2)/100)*180/pi;
pitch_angle=0.26;
accs=(200-10)/100/gravity(0,0);
```

```
%% 0.1 and 0.01 - time steps
dt=.1;
% segparam = [1 5 NaN NaN  NaN NaN NaN NaN 0.1; % 1 - Level constant velosity for 5 seconds
% % %             4 NaN NaN NaN -999 0  pitch_angle 1 0.01;    % 2 - pitch up transition
% % %             2  100 accs NaN NaN NaN NaN NaN 0.1;    % 3 - 100 second climb acceleration
% % %             4 NaN NaN NaN -999 0    0 1 0.01;    % 4 - level off with
% %           1 300 NaN NaN  NaN NaN NaN NaN 0.1; % 5 - Level constant velosity for 300 seconds
% % %             4 NaN NaN NaN  15 3 -999 0 0.01;    % 6 -  15 grad roll into a turn
% % %             3 NaN NaN 360 NaN NaN NaN NaN 0.1;    % 7 - 360 degree turn
% % %             4 NaN NaN NaN    0 3 -999 0 0.01;    % 8 - roll back to straight and level
% %           1  100 NaN NaN NaN NaN NaN NaN 0.1];  % 9 - 100 second straight segment to the North
% % %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
global   segparam % described in modified progen.m (progen_new.m)
segparam = [1 5 NaN NaN  NaN NaN NaN NaN dt; % 1 - Level constant velosity for 5 seconds, 0.1 sec -time step; 1 -
            4 NaN NaN NaN -999 0  pitch_angle 1 0.01;    % 2 - pitch up transition (?? time to transition not clear; "1" probably
            2 30 accs NaN NaN NaN NaN NaN dt;   % 3 - 30 second climb acceleration
            4 NaN NaN NaN -999 0    0 1 0.01;    % 4 - level off
            1 70 NaN NaN  NaN NaN NaN NaN dt; % 5 - Level constant velosity for 70 seconds
            4 NaN NaN NaN  15 3 -999 0 0.01;    % 6 -  15 grad roll into a turn
            3 NaN NaN 360 NaN NaN NaN NaN dt;    % 7 - 360 degree turn
            4 NaN NaN NaN   0 3 -999 0 0.01];    % 8 - roll back to straight and level
              % 9 - 100 second straight segment to the North
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
input_fly;
drawnow
jstop=0;
```

```
 83 -        yaw(k)=eulv(3);
 84 -        end
 85 -        yaw(k)=yaw(k)*180/pi;
 86          %%% Ts_EE_to_Base_true clculation %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
 87 -        deltaenu=[profile(k,1) profile(k,2) profile(k,3)]';
 88 -        Ts_EE_to_Base_true{k}=[C*dcmbn deltaenu ;[0 0 0 1]];
 89          %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
 90 -    end
 91
 92 -        zz1=[lat lon hint];  %initial position
 93          %new initial position calculation
 94 -        xyz1=llh2xyz(zz1);
 95 -        deltaenu=[profile(1,1) profile(1,2) profile(1,3)-hint];
 96 -        xyz=enu2xyz_corrected(deltaenu,xyz1);
 97 -        llh=xyz2llh(xyz);
 98 -        lat=llh(1);
 99 -        lon=llh(2);
100 -        hint=llh(3);
101
102 -        zz1=[lat lon hint];  %new initial position
103
104 -        profile1(1,1:3)=[zz1(2) zz1(1) hint];
105 -        xyz1=llh2xyz(zz1);
106 -        for ii = 2:size(profile,1)
107 -        deltaenu=[profile(ii,1)-profile(ii-1,1) profile(ii,2)-profile(ii-1,2) profile(ii,3)-profile(ii-1,3)];
108 -        xyz=enu2xyz_corrected(deltaenu,xyz1);
109 -        llh=xyz2llh(xyz);
110 -        xyz1=xyz;
111 -        profile1(ii,1:3)=[llh(2) llh(1) llh(3)];
112 -        end
113 -        profile(:,1:2)=profile1(:,1:2)/grad2rad;
114 -        profile(:,3)=profile1(:,3);
```

5 Results

Output of the program 1

Accelerometer measurements are obtained through the **"deltaV deltaTheta Calculation program ()"**

X (i) = X ((i +1) * delta(t)) - X (i * delta(t))

Y (i) = Y ((i +1) * delta(t)) - Y (i * delta(t))

Z (i) = Z ((i +1) * delta(t)) - Z (i * delta(t))

vx(i) = delta(X)(i)/delta(t)

vy(i) = delta(Y)(i)/delta(t)

vz(i) = delta(Z)(i)/delta(t)

vx(i) = vx((i +1) * delta(t))- vx(i *delta(t))

vy(i) = vy((i +1) * delta(t))- vy(i *delta(t))

vz(i) = vz((i +1) * delta(t))- vz(i *delta(t))

```
function  deltaV_deltaTheta_Calculation(...
profile,...  %profile information about motion, obtained by motion generator progen
vxbias,...        % 40 micro-g x-accel bias
vybias,...         % 50 micro-g y-accel bias
vzbias,...
vxsferr,...       % 0 accel scale-factor errors
vysferr,...
vzsferr,...
vxstdev,...         % 0 accel white noise standard deviation
vystdev,...
vzstdev,...
thxbias,...        % 0.01 deg/hr x-gyro bias
thybias,...       % 0.015 deg/hr y-gyro bias
thzbias,...
thxsferr,...         % 0 gyro scale-factor errors
thysferr,...
thzsferr,...
thxstdev,...      % 0.009 deg/root-hour gyro white noise standard deviation
thystdev,...     % 0.005 deg/root-hour gyro white noise standard deviation
thzstdev,...     % 0.0005 deg/root-hour gyro white noise standard deviation
earthflg,... Earth type
rnd_seed_theta,...
rnd_seed_vel) %random generator initialization

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%
%
%      Spheroidal, rotating earth effects (and gravity) included
%
%      ERRORS ADDED TO THE MEASUREMENTS

% clear all
% close all
```

```
 fprintf(1,' . . . . . deltaV deltaTheta search\n')
%ms2nmh=3600*3.2808/6076;
nmph2mps = 1.6878*0.3048;
%g = gravity(0,0);



dvparam = [vxbias vybias vzbias;       % Set up parameter matrix for
   vxsferr vysferr vzsferr;            % GENDVERR
   vxstdev vystdev vzstdev];

dthparam = [thxbias thybias thzbias;   % Set up parameter matrix for
   thxsferr thysferr thzsferr;         % GENTHERR
   thxstdev thystdev thzstdev];



time = profile(:,19);       % time vector
totvel_prof = sqrt(sum(profile(:,4:5)'.^2))'/nmph2mps;
totvelx_prof = profile(:,4)/nmph2mps ;
totvely_prof = profile(:,5)/nmph2mps;
totvelz_prof = profile(:,6)/nmph2mps;
tc_prof=atan2(totvelx_prof,totvely_prof);
height_prof = profile(:,3);
lat_prof=profile(:,2)*pi/180;
lon_prof=profile(:,1)*pi/180;
DCMnb_prof(:,1:3) = profile(:,10:12);
DCMnb_prof(:,4:6) = profile(:,13:15);
DCMnb_prof(:,7:9) = profile(:,16:18);
DCMel_prof = dcmelgen(lat_prof, lon_prof, tc_prof);
vel_prof_L(:,1:3) = profile(:,4:6);   % Generate velocity profile


fprintf(1,' . . . . . \n')
```

```
71 -  dthetbody = gendthet_corrected(DCMnb_prof);       % component of delta-theta associated
72    %                                     % with body motion
73
74 -  deltaer = earthrot(time,DCMel_prof,DCMnb_prof);   % component of delta-theta
75    %                                                 % associated with earth-rate
76    %                                   % generate the component of delta-theta
77    %                                   % associated with craft rate
78 -  deltacr = gendelcr(lat_prof,tc_prof,totvel_prof,...
79       height_prof,time,DCMnb_prof,DCMel_prof,earthflg);
80
81 -  deltheta = dthetbody + deltaer + deltacr;   % ideal (error-free) gyro output
82
83 -  dtherr = gentherr_corr(deltheta,time,dthparam,rnd_seed_theta);  % generate delta-theta errors
84
85 -  est_dtheta = deltheta + dtherr;   % form profiles of 'measured' delta-theta's
86
87 -  deltav_b = gendv(vel_prof_L,DCMnb_prof);    % Generate the component of delta-V
88    %                                           % associated with body motion relative
89    %                                           % to the earth
90
91    %                                           % Generate the Coriolis component
92    %                                           % of delta-V and Gravitation
93    %                                           % component g
94 -  dvcor = gendvcor_new(lat_prof,totvel_prof,totvelz_prof,tc_prof,height_prof,time,...
95       DCMnb_prof,DCMel_prof,earthflg);
96 -  dvtot = deltav_b + dvcor;
97
98 -  dverr = gendverr_corr(dvtot,time,dvparam,rnd_seed_vel);  % Generate delta-V errors
99
100 -  est_dv = dvtot + dverr;           % form profile of 'measured' delta-V's
101
102 -  dvtot=est_dv;
103 -  deltheta=est_dtheta;
104 -  save data dvtot deltheta
```

Output of the program 2

Gyroscope measurements are obtained through the
"deltaV deltaTheta Calculation program () "

$\varphi\,(i \cdot \Delta t),\ \theta\,(i \cdot \Delta t),\ \psi\,(i \cdot \Delta t) \longrightarrow R(i)$

$\varphi\,((i+1) \cdot \Delta t),\ \theta\,((i+1) \cdot \Delta t),\ \psi\,((i+1) \cdot \Delta t) \longrightarrow R(i+1)$

$\Delta R\,(i) = (R(i))^{-1} \cdot (R(i+1))$

$\Delta R\,(i) \longrightarrow \Delta\varphi\,(i),\ \Delta\theta\,(i),\ \Delta\psi\,(i)$

$$R_{\varphi} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos(\varphi) & sin(\varphi) \\ 0 & -\sin(\varphi) & \cos(\varphi) \end{pmatrix}$$

$$R_{\theta} = \begin{pmatrix} \cos(\theta) & 0 & -\sin(\theta) \\ 0 & 1 & 0 \\ \sin(\theta) & 0 & \cos(\theta) \end{pmatrix}$$

$$R_{\psi} = \begin{pmatrix} \cos(\psi) & \sin(\psi) & 0 \\ -\sin(\psi) & \cos(\psi) & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

$$\varphi = arctg\left(R_{3,2}/R_{3,3}\right)$$

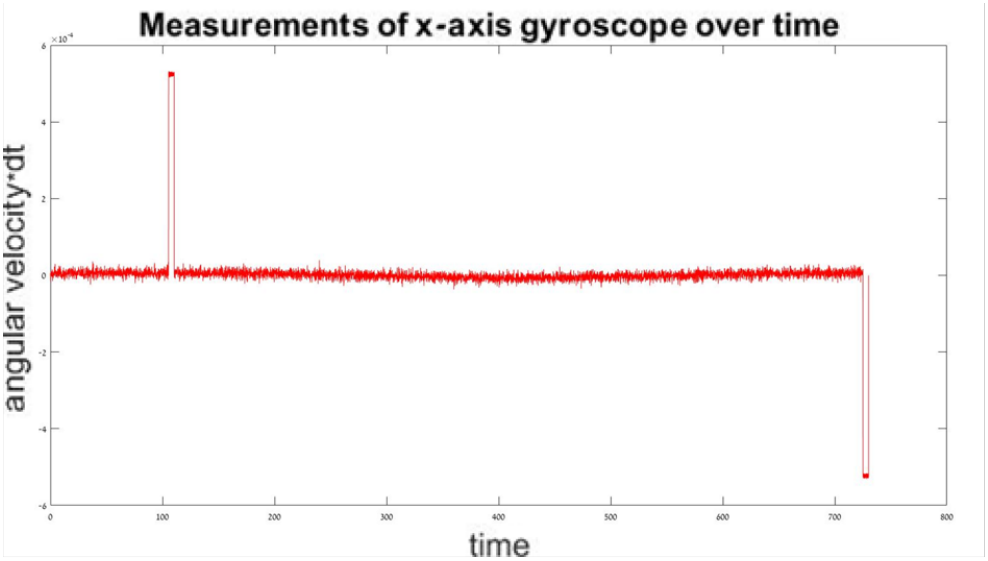
$$\theta = arcsin\left(-R_{3,1}\right)$$

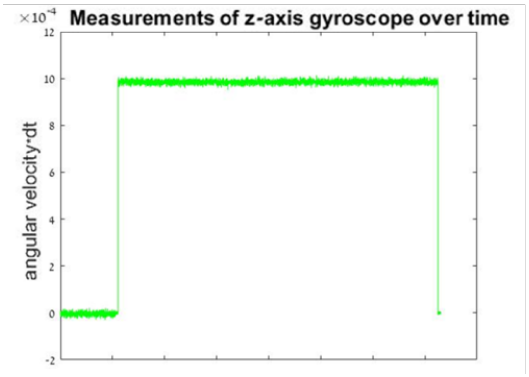
$$R = (R_{\psi} \cdot R_{\theta} \cdot R_{\varphi})^{\mathrm{T}}$$

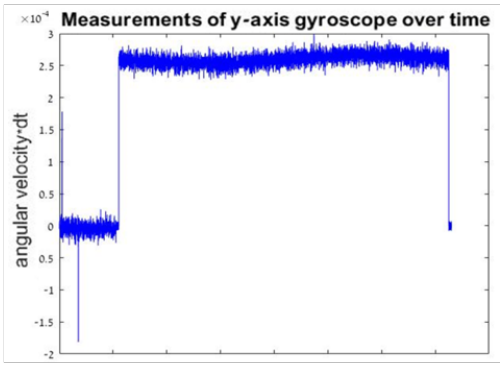
$$R_{1,1} > 0 \text{ אם } \psi = arctg\left(\frac{R_{2,1}}{R_{1,1}}\right))$$

$$R_{1,1} < 0\,;R_{2,1} > 0 \text{ אם } \psi = \pi + arctg\left(\frac{R_{2,1}}{R_{1,1}}\right)$$

$$R_{1,1} < 0\,;R_{2,1} < 0 \text{ אם } \psi = -\pi + arctg\left(\frac{R_{2,1}}{R_{1,1}}\right)$$

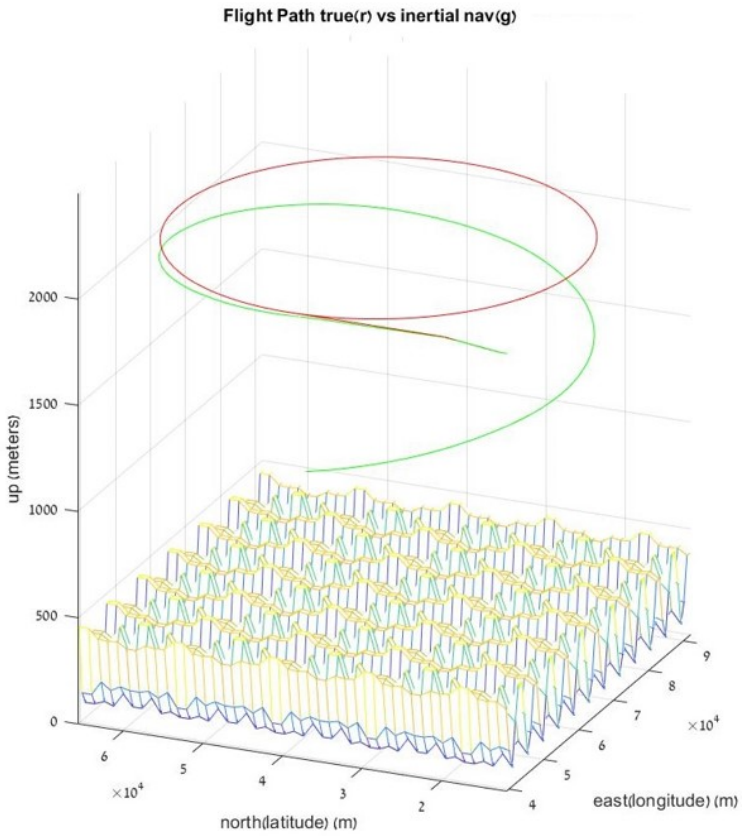

Measurements of x-axis gyroscope over time
angular velocity*dt
time


Measurements of z-axis gyroscope over time
angular velocity*dt


Measurements of y-axis gyroscope over time
angular velocity*dt

Restore the trajectory acceleration dimension and gyroscope

We use a function: **"Data Generator from deltaV deltaTheta ()"**
To restore a route:
Precise track red color.
Green color Reconstruction of a track based on an accelerometer and gyroscope.
See that there is a drag error in the recovery.



```
% Reconstruction of trajectory from sensor data, described in the article
function [Ts_EE_to_Base_drifted...
 tru_lat_new...
 tru_long_new...
 tru_alpha_new...
 tru_height_new...
 tru_roll_new...
 tru_pitch_new...
 tru_yaw_new...
 t...
 vx_1...
 vy_1...
 vz_1...
 dxenu_err...
 dyenu_err]=Data_Generator_from_deltaV_deltaTheta(...
Ts_EE_to_Base_true1,...
deltheta,...
dvtot,...
time ,...              % time vector
vxbias,...             % 40 micro-g x-accel bias
vybias,...              % 50 micro-g y-accel bias
vzbias,...
vxsferr,...       % 0 accel scale-factor errors
vysferr,...
vzsferr,...
vxstdev,...         % 0 accel white noise standard deviation
vystdev,...
vzstdev,...
thxbias,...         % 0.01 deg/hr x-gyro bias
thybias,...       % 0.015 deg/hr y-gyro bias
thzbias,...
thxsferr,...          % 0 gyro scale-factor errors
thysferr,...
thzsferr,...
```

```
35      thxstdev,...      % 0.009 deg/root-hour gyro white noise standard deviation
36      thystdev,...     % 0.005 deg/root-hour gyro white noise standard deviation
37      thzstdev,...     % 0.0005 deg/root-hour gyro white noise standard deviation
38      vertmech,...
39      tru_alpha,...
40      alphaerr,...
41      n_tilt ,...% 10 arc-second initial north tilt error
42      e_tilt ,...
43      u_tilt,...
44      n_init_vel_error ,... % 0.1 m/s north initial velocity error
45      e_init_vel_error ,...
46      u_init_vel_error ,...
47      laterr,...
48      longerr,...
49      height_err,...
50      dcmnb,...
51      tru_lat ,...
52      tru_long ,...
53      tru_height,...
54      vx1 ,...            % Incorporate initial velocity     % errors into initialization
55      vy1,...             % parameters
56      vz1,...
57      earthflg,... Earth type
58      rnd_seed_theta,...
59    rnd_seed_vel) %random generator initialization
60
61    %
62    %
63    %      Constant-altitude, constant velocity user
64    %      Spheroidal, rotating earth effects (and gravity) included
65    %
66    %      ERRORS ADDED TO THE MEASUREMENTS
```

```
70 -  C = [0 1 0; 1 0 0; 0 0 -1];    % conversion between NED and ENU
71 -  m2km=1/1000;
72 -  dph2rps = (pi/180)/3600;   % conversion constant from deg/hr to rad/sec
73 -  g = gravity(0,0);
74 -  nmph2mps = 1.6878*0.3048;
75
76 -  npts = length(deltheta)+1;
77
78
79 -  td12=time(2)-time(1);
80 -  tdint = time(2) - time(1);
81
82 -  dcmbn=dcmnb';
83 -  eulv=extractEulerAngles(dcmbn);
84 -  roll = eulv(1)*180/pi;
85 -  pitch = eulv(2)*180/pi;
86 -  yaw = eulv(3)*180/pi;
87    % phi=eulv(1) + n_tilt;            % North tilt corresponds to the
88    % theta=eulv(2) + e_tilt;          % aircraft roll axis in this case
89    % psi=eulv(3)+u_tilt;        % because aircraft is pointed north
90
91 -  DCMnb=createRfromAngles([eulv(1) eulv(2) eulv(3)])';
92 -  DCMbn = DCMnb';
93 -  DCMbn = bodupdat_corrected(DCMbn,DCMnb*C*[e_tilt, n_tilt, u_tilt]' );
94 -  eul_vect = extractEulerAngles(DCMbn);
95 -  phi=eul_vect(1); % roll angle in radians
96 -  theta=eul_vect(2); % pitch angle in radians
97 -  psi=eul_vect(3); % yaw angle in radians
98
99
100 -  tru_alpha =tru_alpha +alphaerr;
101
102 -  llh1=[tru_lat*pi/180 tru_long*pi/180 tru_height];
103 -  tru_lat = tru_lat*pi/180+laterr;
```

```
104 -   tru_long = tru_long*pi/180+longerr;
105 -   tru_height=tru_height+height_err;
106
107     %   Ts_EE_to_Base_drifted clculation %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
108 -   enu1=Ts_EE_to_Base_true1(1:3,4);
109
110 -   eul_vect1(1) = phi;   %roll angle in radians
111
112 -   eul_vect1(2) = theta; %pitch angle in radians
113
114 -   eul_vect1(3) = psi;   %yaw angle in radians
115
116 -   llh2=[tru_lat tru_long tru_height];
117
118 -   denu = xyz2enu(llh2xyz(llh2),llh2xyz(llh1));
119
120 -   dxenu_err=denu(1);
121 -   dyenu_err=denu(2);
122
123 -   llh1=llh2;
124
125 -   enu2=enu1+denu;
126 -   enu1=enu2;
127
128 -   Ts_EE_to_Base_drifted{1}=[C*createRfromAngles(eul_vect1) enu2 ;[0 0 0 1]];
129     %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
130
131
132 -   height2 = tru_height;
133 -   height1 = height2;
134 -   est_height(1) = height2;
135
136 -   vx1 = vx1+e_init_vel_error;             % Incorporate initial velocity
```

```
vx2 = vx1;                              % errors into initialization
vy1= vy1+n_init_vel_error;                % parameters
vy2 = vy1;
vz1= vz1+u_init_vel_error;
vz2=vz1;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% INITIALIZATION in this whole section

DCMnb=createRfromAngles([phi theta psi])';  DCMbn = DCMnb';

roll(1) = phi; pitch(1) = theta; yaw(1) = psi;

DCMel = llw2dcm([tru_lat tru_long tru_alpha]);

est_lat(1) = tru_lat;
est_lon(1) = tru_long;
est_alpha(1) = tru_alpha;

est_DCMbn = DCMnb';
est_DCMel = DCMel;

omega2_el_L = crafrate(tru_lat,vx1,vy1,height2,est_DCMel,earthflg,vertmech);
[DCMel_new, DCM_ll_E] = navupdat_corrected(omega2_el_L,omega2_el_L,td12,est_DCMel,1);
llw_vect = dcm2llw(DCMel_new);
lat_ex=llw_vect(1);
lat2 = tru_lat;  lat1 = lat2 - (lat_ex-lat2);
```

```
vel_l(1,:) = [vx2 vy2 vz2];
vel2 = [vx1 vy1 vz1];  vel1 = vel2;
cang = cos(-est_alpha(1));     % We are building up a rotation matrix
   sang = sin(-est_alpha(1));     % here to convert velocity from local-level
   Crotmat = [cang sang 0;        % coordinates to East-North-Up.
              -sang cang 0;
              0 0 1];
  vtmp=vel_l(1,:)';
  vel_ENU(1,:) = ( Crotmat*vtmp )';

dvparam = [vxbias vybias vzbias;       % Set up parameter matrix for
   vxsferr vysferr vzsferr;            % GENDVERR
   vxstdev vystdev vzstdev];


dthparam = [thxbias thybias thzbias;   % Set up parameter matrix for
   thxsferr thysferr thzsferr;         % GENTHERR
   thxstdev thystdev thzstdev];




dtherr = gentherr_corr(deltheta,time,dthparam,rnd_seed_theta);  % generate delta-theta errors

est_dtheta = deltheta - dtherr;   % form profiles of 'real' delta-theta's

dverr = gendverr_corr(dvtot,time,dvparam,rnd_seed_vel);  % Generate delta-V errors

est_dv = dvtot - dverr;           % form profile of 'real' delta-V's

fprintf(1,' . . . . . \n')
fprintf(1,' Starting nav computations \n')
h = waitbar(0,'nav computations');
```

```
206 - for i = 1:npts-1
207 -     if i>2
208 -    td12 = time(i) - time(i-1);
209 -     end
210
211 -    if i<npts
212 -    tdint = time(i+1) - time(i);
213 -    end
214
215 -    tdex = 0.5*tdint;
216
217      %    [DCM_ll_I, omega_el_L, ~] = lclevupd_corrected(lat1,lat2,vx1,vx2,vy1,vy2,...
218      %       height1,height2,td12,tdex,tdint,est_DCMel,vertmech,1,earthflg);
219      %
220      %   est_DCMbn = createRfromAngles(est_dtheta(i,1:3))';
221      %    est_DCMbn = C*(DCM_ll_I*(C*est_DCMbn));
222
223 -    est_DCMbn = bodupdat_corrected(est_DCMbn,est_dtheta(i,1:3));
224
225 -    [DCM_ll_I, omega_el_L, ~] = lclevupd_corrected(lat1,lat2,vx1,vx2,vy1,vy2,...
226         height1,height2,td12,tdex,tdint,est_DCMel,vertmech,1,earthflg);
227
228 -    est_DCMbn = C*(DCM_ll_I*(C*est_DCMbn));
229
230 -    eul_vect = extractEulerAngles(est_DCMbn);
231 -    roll(i) = eul_vect(1);
232 -    pitch(i) = eul_vect(2);
233 -    if i==1
234 -    yaw(1) = yawprog(eul_vect(3),yaw(1));
235 -    continue;
236 -    else
237 -    yaw(i) = yawprog(eul_vect(3),yaw(i-1));
238 -    end
```

```
240 -       est_delv_b = est_dv(i,1:3);       % extract delta-V for current point in time
241
242 -       del_Vl = C*(est_DCMbn*est_delv_b');
243
244 -       omega1_el_L = omega2_el_L;   omega2_el_L = omega_el_L;
245 -       [est_DCMel, ~] = navupdat_corrected(omega1_el_L,omega2_el_L,td12,est_DCMel,1);
246
247 -       h_extr = extrapol(height1,height2,td12,tdex);
248
249 -       lat_extr = extrapol(lat1,lat2,td12,tdex);
250
251 -       g_extr = gravity(lat_extr,h_extr);
252
253
254 -       vtmp = velupdat(vel2,vel1,td12,tdex,del_Vl,...
255            omega_el_L,est_DCMel,g_extr,0,tdint);
256
257 -       vel_l(i,:) = vtmp';
258
259
260     %    est_height(i) = height2+td12*vel_u_extr;
261     %    est_height(i) = tru_height;
262
263        %pos(i,1:3)= pos(i-1,1:3) + deltat*mean([vel_L(i,1:3); vel_L(i-1,1:3)]);
264
265 -       vx1 = vx2; vy1 = vy2;
266 -       vx2 = vel_l(i,1);  vy2 = vel_l(i,2);
267 -       vel1 = vel2;   vel2 = vel_l(i,:);
268 -       llw_vect = dcm2llw(est_DCMel);
269 -       est_lat(i) = llw_vect(1); est_lon(i) = llw_vect(2);  est_alpha(i) = llw_vect(3);
270 -       lat1 = lat2;  lat2 = est_lat(i);
```

```
274 -       cang = cos(-est_alpha(i));     % We are building up a rotation matrix
275 -      sang = sin(-est_alpha(i));    % here to convert velocity from local-level
276 -      Crotmat = [cang sang 0;        % coordinates to East-North-Up.
277                  -sang cang 0;
278                     0 0 1];
279 -    vel_ENU(i,:) = ( Crotmat*vtmp )';
280 -      vel_u_extr=(vel_ENU(i,:)+vel_ENU(i-1,:))/2;
281     %
282     %    zz1=[est_lat(i-1) est_lon(i-1) 0];
283     %     xyz1=llh2xyz(zz1);
284     %     xyz=enu2xyz(vel_u_extr*td12,xyz1);
285     %     zz2=[est_lat(i) est_lon(i) 0];
286     %     xyz2=llh2xyz(zz2);
287     %     delta=xyz2enu(xyz,xyz2);
288     %     dheight=delta(3);
289     %     est_height(i) = height2+dheight;
290     %
291     %    est_height(i) = height2+td12*vel_u_extr(3);
292     %    if(est_height(i)<0)est_height(i)=0; end
293     %    if(est_height(i)>100000000)est_height(i)=100000000; end
294
295 -       est_height(i) = est_height(i-1)+td12*vel_u_extr(3);
296 -       height1 = height2;  height2 = est_height(i);
297
298
299         %%% Ts_EE_to_Base_drifted clculation %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
300 -       phi=roll(i-1);
301 -       theta=pitch(i-1);
302 -       psi=yaw(i-1);
303
304 -       eul_vect1(1) = phi;   %roll angle in radians
305
306 -       eul_vect1(2) = theta; %pitch angle in radians
```

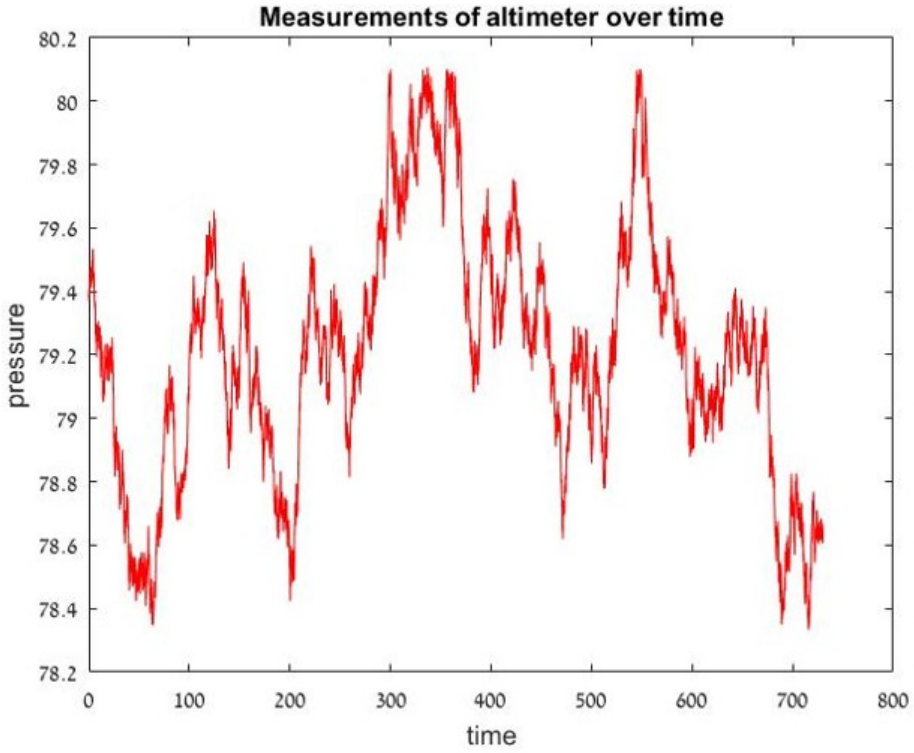
```
308 -       eul_vect1(3) = psi;   %yaw angle in radians
309 
310 -       llh2=[est_lat(i) est_lon(i) est_height(i)];
311 
312 -       denu = xyz2enu(llh2xyz(llh2),llh2xyz(llh1));
313 
314 -       denu(3)=llh2(3)-llh1(3); % height relative to sea level
315 -       llh1=llh2;
316 
317 -       enu2=enu1+denu;
318 -       enu1=enu2;
319 
320 -       Ts_EE_to_Base_drifted{i}=[C*createRfromAngles(eul_vect1) enu2 ;[0 0 0 1]];
321         %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
322 -       waitbar(i/(npts-1),h)
323 -   end
324 -   close(h)
325 -   t = time(1:npts-1);
326 -   tru_lat_new=est_lat;
327 -   tru_long_new=est_lon;
328 -   tru_height_new=est_height;
329 -   tru_roll_new=roll(1:length(t))*180/pi;
330 -   tru_pitch_new=pitch(1:length(t))*180/pi;
331 -   tru_yaw_new=yaw(1:length(t))*180/pi;
332 -   tru_alpha_new=est_alpha;
333 -   vx_1=vel_1(:,1);
334 -   vy_1=vel_1(:,2);
335 -   vz_1=vel_1(:,3);
```

Output of the program 3

Altimeter measurements are obtained through the function:
"altimeter gen ()"

$h(i) = Z(i^* \Delta t)$
$p(i) = F(h(i))$

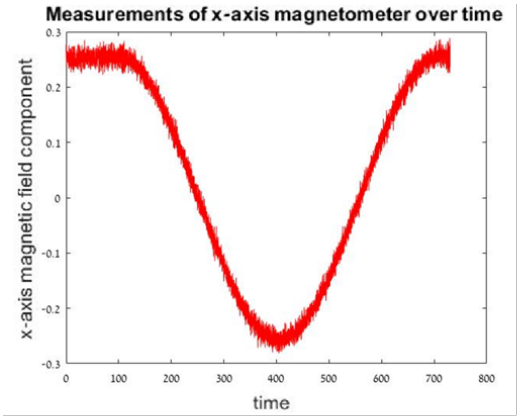
F - a function that gives a relationship between altitude and air pressure

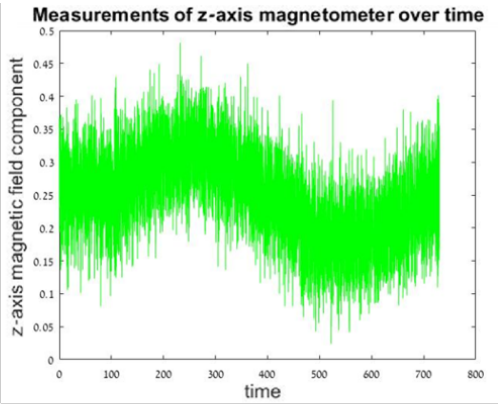


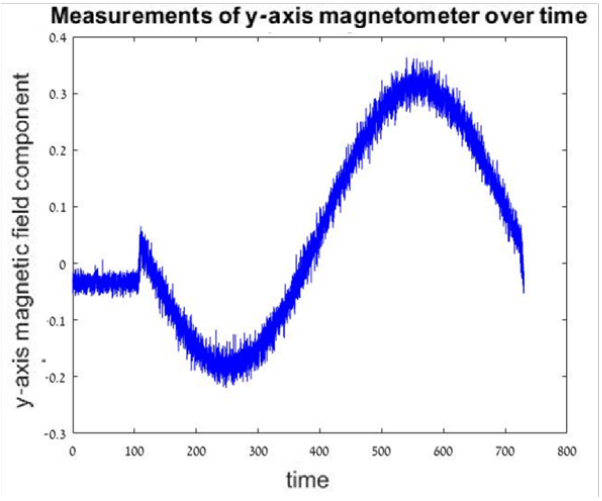
```
%%generate altimeter measurement as function of time
function [h_shift_init, press] = altimeter_gen(rnd_seed_h, Th_h_shift, Ts_EE_to_Base_true, timeInd, Err_h1, CONST_H, tru_height_new)
rng(rnd_seed_h);
rand1=2*(rand(1)-.5);
h_shift=Th_h_shift*rand1; % inital condition of drift of altimeter.
t_timeInd_init = Ts_EE_to_Base_true{1}(1:3,4);
h_1_new=zeros(1,timeInd);
press=h_1_new;
for i=1:timeInd % altimeter with noise

    ran1=randn(1);
    ran2=randn(1);

    t_timeInd_init0 = t_timeInd_init;
    t_timeInd_init = Ts_EE_to_Base_true{i}(1:3,4); %Ts_EE_to_Base_true 4x4 true matrix UAV to ENU
    h_1_new(i)=t_timeInd_init(3)+h_shift+Err_h1*ran1;%%%%%%% func1 = const - added in \prog_yes_INS\

    dx=t_timeInd_init-t_timeInd_init0;
    h_shift=h_shift+CONST_H*sqrt(dx(1)*dx(1)+dx(2)*dx(2))*ran2;%%%%% define CONST_H from
    if(abs(h_shift)>Th_h_shift)
    h_shift=h_shift-CONST_H*sqrt(dx(1)*dx(1)+dx(2)*dx(2))*ran2;
    end
    press(i)=inv_kPa2m(h_1_new(i));% we dont need entire array h_1_new(i)

end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
h_shift_init=kPa2m(press(1))-tru_height_new(1);%input for runExperiment
```

Output of the program 4

Measurements of magnetic field meter (H (i)) are obtained through the **"magnetometergen()"** function.







```
%%generate magnetometer measurement as function of time
function [HF, Ts_Base_to_EE, Err_HFx_enu]=magnetometer_gen(rnd_seed_HF, Err_HN, Err_H3, timeInd, HH, HN, Ts_EE_to_Base_true, C, D)
rng(rnd_seed_HF);
Err_HFx_enu=sqrt(2*Err_HN^2+Err_H3^2);
HF=zeros(3,timeInd);
for i=1:timeInd

    ran3=randn(1);
    ran4=randn(1);
    ran5=randn(1);

    % Magnetic field in NED (some typical values)
%    HH = 0.256;
%    HN = 0.256;
    HF(1,i)=HN*cos(D)+Err_HN*ran3;
    HF(2,i)=HN*sin(D)+Err_HN*ran4;
    HF(3,i)=HH+Err_H3*ran5;
    % Magnetic field in body coordinates
    Ts_Base_to_EE=invmy(Ts_EE_to_Base_true{i});
    hff=Ts_Base_to_EE(1:3,1:3)*C*[HF(1,i);HF(2,i);HF(3,i)]; % matrix C related to transform from ENU to NED and vice versa
    HF(1,i)=hff(1);
    HF(2,i)=hff(2);
    HF(3,i)=hff(3);
    % Examples for China, Hangzhou
end
```

Output of the program 5 - videos

Every half second we take a photo through the
"images and masks generation ()" function.

```
function images_and_masks_generation(Ts_EE_to_Base_in,CameraMatStruct,NpX,NpY,umax,vmax,nImage,thetaThresh)
%%%DTM definition and smoothing
global DTM;
%global smoothDTM;

% DTM data files source
% stri=num2str(DTMsize);
% if DTMsize<10
%     imname=['0',stri];
% else
%     imname=stri;
% end
% DTMmy=['..\..\experimentData\calibration\DTMdata\DTMdata',imname,imname,'.mat'];
% as1 = exist(DTMmy,'file');
% DTMmy=['..\..\experimentData\calibration\DTMdata\DTMdatasmooth',imname,imname,'.mat'];
% as2 = exist(DTMmy,'file');
imname=[];
stri=num2str(nImage(end));
if nImage(end)<10
    imname=['0000',stri];
elseif nImage(end)<100
    imname=['000',stri];
elseif nImage(end)<1000
    imname=['00',stri];
elseif nImage(end)<10000
    imname=['0',stri];
elseif nImage(end)<100000
    imname=stri;
end
imageFileName=['..\..\experimentData\photos\image_',imname,'.bmp'];
as4 = exist(imageFileName,'file');

```

```
% if as1==0
%     makeDTM;
% end
%
% % DTM parameters
% initDTM_for_integration(DTM_fileName,DTMdelta,DTMsize);

% smoothDTM = DTM;
% smoothDTM.DTM_fileName=[DTM.DTM_fileName,'smooth'];
% DTM_fileNamemy=[smoothDTM.DTM_fileName,'01','01'];

% if as2==0 || as1==0 %??? as1==0
%     %%%DTM  smoothing
%     smooth_DTM(smooth1,smooth2,DTMsize);
% end
%
% tmp=load(DTM_fileNamemy);
% DTMdata=tmp.DTMdata;
% smoothDTM.data = DTMdata;

if as4==0
    T_DTM_to_Base = CameraMatStruct.T_W_to_B; % shift of DTM system relative to ENU system
    T_Base_to_DTM = invmy(T_DTM_to_Base);

    %%%%%%%%%% Blok 3 f %%%%%%%%%%%%%%%%%%%%%%%%%%%%

    %%%%%%%%% Camera parameters
    cameraAp=[umax, vmax]; %

    T_EE_to_Cam=CameraMatStruct.T_EE_to_Cam; %camera orientation relative to body

    T_Cam_to_EE = invmy(T_EE_to_Cam); % should be unit matrix
```

```
    deltaX=DTM.delta;
    deltaY=DTM.delta;
    for i=1:length(nImage)
        T_Cam_to_Base=Ts_EE_to_Base_in{nImage(i)}*T_Cam_to_EE;
        Ts_Base_to_Cam=invmy(T_Cam_to_Base);
        cameraPos=T_Cam_to_Base(1:3,4);
        cameraRot=T_Cam_to_Base(1:3,1:3);
        % cameraPos, corners, center - all in the Base system (meters)
        % cameraRot - in Base system
        [corners, center, numOfRegCorners]=planeProjection_for_integration(cameraAp,cameraPos,cameraRot);
        center=[];
        [X, Y, Z, lightOut]=matCrop_for_integration(corners,numOfRegCorners,T_Base_to_DTM,deltaX,deltaY,cameraPos,thetaThresh,center);
        [D, ~, finf]=picture_from_position_for_integration(Ts_Base_to_Cam,X,Y,Z,lightOut,NpX,NpY,umax,vmax);

        strTmp=num2str(nImage(i));
        L=length(strTmp);
        if L<5
            L0='00000';
            L0=L0(1:(5-L));
            L=[L0 strTmp];
        end
        filename=['..\..\experimentData\photos\image_',L, '.bmp'];
        D_uint8 = uint8(255 * (D));
        imwrite(D_uint8, filename);
        if ~isempty(center)
            figure; imshow(D);
        end
        filename=['..\..\experimentData\photos\masks\mask_',L, '.bmp'];
        D_uint8 = uint8(255 * (finf));
        imwrite(D_uint8, filename);
    end
end
return
```

6 Conclusion

We created our own input: a height map, a brightness map and an accurate drone route.
From this trajectory we obtained output:
measurements of sensors 3 measurements of accelerometer, 3 measurements of gyroscope, measurement of altimeter and 3 measurements of magnetic field as a function of time (i * delta(t), where i is a measurement number).
Also from a trajectory and a map of heights and a map of brightness through a model of a pinhole camera with the help of proactive geometry we got a film of ground pictures.
The output obtained can be used to restore a drone trajectory

7 Bibliography

[Agrawal-03] M. Agrawal and L. Davis. Camera calibration using spheres: A dual-space approach. Research Report CAR-TR-984, Center for Automation Research, University of Maryland, 2003.

Aloimonos-90
J. Y. Aloimonos. Perspective approximations. Image and Vision Computing, 8(3):177– 192, August 1990.

Anandan-02
P. Anandan and M. Irani. Factorization with uncertainty. International Journal of Computer Vision, 49(2/3):101–116, 2002.

Armstrong-94
M. Armstrong, A. Zisserman, and P. Beardsley. Euclidean reconstruction from uncalibrated images. In Proc. British Machine Vision Conference, pages 509–518, 1994.

Armstrong-96a
M. Armstrong. Self-Calibration from Image Sequences. PhD thesis, University of Oxford, England, 1996.

Armstrong-96b
M. Armstrong, A. Zisserman, and R. Hartley. Self-calibration from image triplets. In Proc. European Conference on Computer Vision, LNCS 1064/5, pages 3–16. Springer-Verlag, 1996.

Astrom-98
K. A ˚ stro¨m and A. Heyden. Continuous time matching constraints for image streams. International Journal of Computer Vision, 28(1):85–96, 1998.

Avidan-98
S. Avidan and A. Shashua. Threading fundamental matrices. In Proc. 5th European Conference on Computer Vision, Freiburg, Germany, pages 124–140, 1998.

Baillard-99
C. Baillard and A. Zisserman. Automatic reconstruction of piecewise pla-

nar models from multiple views. In Proc. IEEE Conference on Computer Vision and Pattern Recognition, pages 559– 565, June 1999.

Barrett-92
E. B. Barrett, M. H. Brill, N. N. Haag, and P. M. Payton. Invariant linear methods in photogrammetry and model-matching. In J. L. Mundy and A. Zisserman, editors, Geometric invariance in computer vision. MIT Press, Cambridge, 1992. [Bascle-98] B. Bascle and A. Blake. Separability of pose and expression in facial tracing and animation. In Proc. International Conference on Computer Vision, pages 323–328, 1998.

Basri-99
R. Basri and D. Jacobs. Projective alignment with regions. In Proc. 7th International Conference on Computer Vision, Kerkyra, Greece, pages 1158–1164, 1999.

Bathe-76
K-J. Bathe and E. Wilson. Numerical methods in finite element analysis. Prentice Hall, 1976. [Beardsley-92] P. A. Beardsley, D. Sinclair, and A. Zisserman. Ego-motion from six points. Insight meeting, Catholic University Leuven, February 1992. [Beardsley-94] P. A. Beardsley, A. Zisserman, and D. W. Murray. Navigation using affine structure and motion. In Proc. European Conference on Computer Vision, LNCS 800/801, pages 85–96. Springer- Verlag, 1994.

Beardsley-95a
P. A. Beardsley and A. Zisserman. Affine calibration of mobile vehicles. In Europe– China workshop on Geometrical Modelling and Invariants for Computer Vision, pages 214–221. Xidan University Press, Xi'an, China, 1995.

Beardsley-95b

P. A. Beardsley, I. D. Reid, A. Zisserman, and D. W. Murray. Active visual navigation using non-metric structure. In Proc. International Conference on Computer Vision, pages 58–64, 1995.