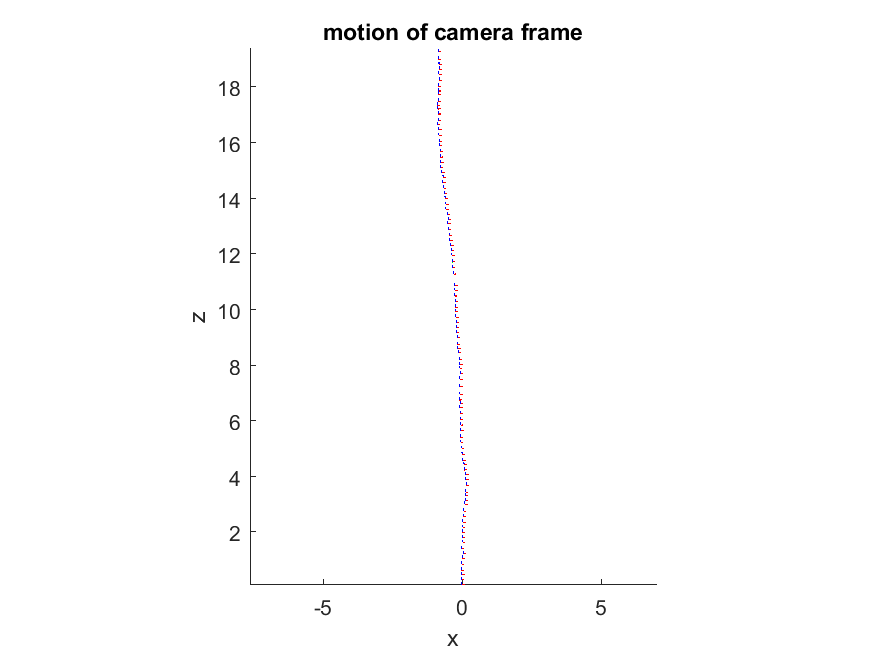
AER521 Assignment 3 Report

# Visual Odometry



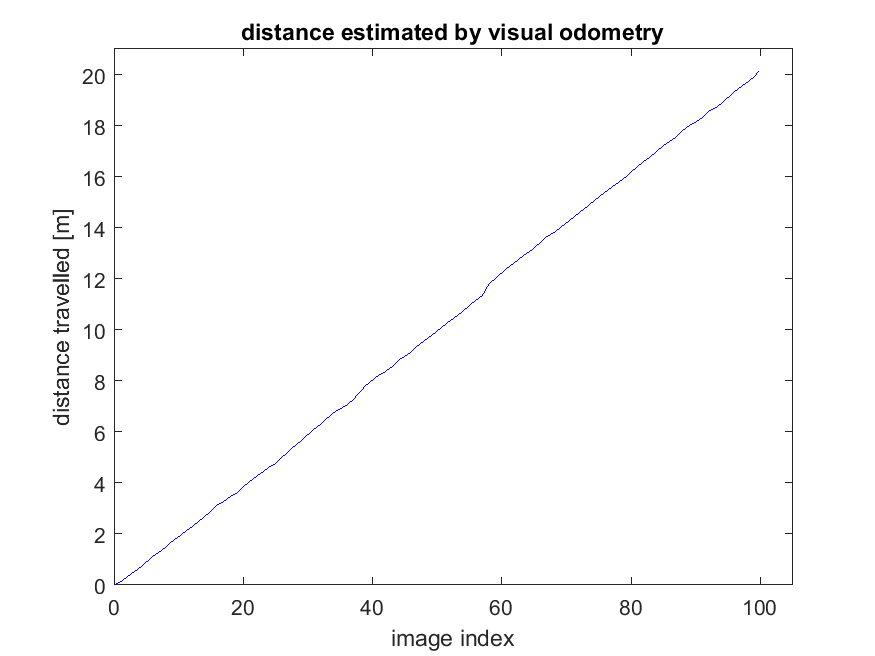


Figure 1: Motion of camera frame Figure 2: Distance traveled over images

The orientation of the frames all look identical to each other. This is expected as the camera/robot should be relatively level. There is a slight discontinuity at around 11 meters, which could be due an issue with the visual odometry, where the estimated the motion is higher than the actual motion.

The distance over image index appears to be a linear line going up. This is expected, as the images are taken at regular intervals and the robot can travel at a limited distance per time interval.

Both images look identical to the solution images, so there will be no comments on the implementation of the code.

# Appendix: MATLAB Code

% ======

% ass3.m

% ======

%

% This assignment will introduce you to the idea of estimating the motion

% of a mobile robot using stereo visual odometry. It uses a real image

% dataset gathered in the Canadian High Arctic in 2009.

%

% There is only one question to complete (10):

%

% Question: code the least-squares motion solution based on two

% pointclouds

%

% Fill in the required sections of this script with your code, run it to

% generate the requested plots, then paste the plots into a short report

% that includes a few comments about what you've observed. Append your

% version of this script to the report. Hand in the report as a PDF file.

%

% requires: basic Matlab, 'matches.mat', directory of images

%

% T D Barfoot, February 2016

%

function vo()

clear

close

clc

% watch the included video,

load matches.mat;

imax = size(matches,2);

% stereo camera parameters

b = 0.239977002; % baseline [m]

f = 387.599884033; % focal length [pixel]

cu = 253.755615234; % horiz image centre [pixel]

cv = 185.114852905; % vert image centre [pixel]

% global transform to camera (starts as identity)

T = eye(4);

% distance travelled (starts at zero)

d(1) = sqrt(T(1:3,4)'\*T(1:3,4));

% initialize some figures

h1 = figure(1); clf;

h2 = figure(2); clf;

% loop over the stereo pairs

for i=1:imax

i

% get number of feature matches

npoints = size( matches{i}, 1);

% use the inverse stereo camera model to turn the first stereo pair into a pointcloud

uL1 = matches{i}(:,1);

vL1 = matches{i}(:,2);

uR1 = matches{i}(:,3);

vR1 = matches{i}(:,4);

p1 = [ b\*( 0.5\*(uL1 + uR1) - cu ) ./ (uL1 - uR1) ...

b\*( 0.5\*(vL1 + vR1) - cv ) ./ (uL1 - uR1) ...

b\*f\*ones(size(uL1)) ./ (uL1 - uR1) ]';

% use the inverse stereo camera model to turn the first stereo pair into a pointcloud

uL2 = matches{i}(:,5);

vL2 = matches{i}(:,6);

uR2 = matches{i}(:,7);

vR2 = matches{i}(:,8);

p2 = [ b\*( 0.5\*(uL2 + uR2) - cu ) ./ (uL2 - uR2) ...

b\*( 0.5\*(vL2 + vR2) - cv ) ./ (uL2 - uR2) ...

b\*f\*ones(size(uL2)) ./ (uL2 - uR2) ]';

% RANSAC

maxinliers = 0;

bestinliers = [];

p1inliers = [];

p2inliers = [];

itermax = 1000;

iter = 0;

while iter < itermax && maxinliers < 50

iter = iter + 1;

% shuffle the points into a random order

pointorder = randperm(npoints);

% use the first 3 points to propose a motion for the camera

[C,r] = compute\_motion( p1(:,pointorder(1:3)), p2(:,pointorder(1:3)) );

% compute the Euclidean error on all points and threshold to

% count inliers

e = p2 - C\*(p1 - r\*ones(1,npoints));

reproj = sum(e.\*e,1);

inliers = find(reproj < 0.01);

ninliers = size(inliers,2);

if ninliers > maxinliers

maxinliers = ninliers;

bestinliers = inliers;

p1inliers = p1(:,inliers);

p2inliers = p2(:,inliers);

end

end

% recompute the incremental motion using all the inliers from the

% best motion hypothesis

[C,r] = compute\_motion(p1inliers,p2inliers);

% update global transform

T = [ C -C\*r; 0 0 0 1]\*T;

% update distance travelled

d(i+1) = sqrt(T(1:3,4)'\*T(1:3,4));

% this figure shows the feature tracks that were identified as

% inliers (green) and outliers (red)

figure(h1)

clf;

IL1 = imread(['images/grey-rectified-left-' num2str(i,'%06i') '.pgm'], 'pgm');

imshow(IL1);

hold on;

for k=1:npoints

set(plot( [uL1(k) uL2(k)], [vL1(k) vL2(k)], 'r-' ), 'LineWidth', 2);

end

for k=1:maxinliers

set(plot( [uL1(bestinliers(k)) uL2(bestinliers(k))], [vL1(bestinliers(k)) vL2(bestinliers(k))], 'g-' ), 'LineWidth', 2);

end

% this figure plots the camera reference frame as it moves through

% the world - try rotating in 3D to see the full motion

figure(h2)

hold on;

startaxis = [0.1 0 0 0; 0 0.1 0 0; 0 0 0.1 0; 1 1 1 1];

curraxis = inv(T)\*startaxis;

plot3( [curraxis(1,1) curraxis(1,4)], [curraxis(2,1) curraxis(2,4)], [curraxis(3,1) curraxis(3,4)], 'r-' );

plot3( [curraxis(1,2) curraxis(1,4)], [curraxis(2,2) curraxis(2,4)], [curraxis(3,2) curraxis(3,4)], 'g-' );

plot3( [curraxis(1,3) curraxis(1,4)], [curraxis(2,3) curraxis(2,4)], [curraxis(3,3) curraxis(3,4)], 'b-' );

axis equal;

pause(0.01);

end

% finish off this figure

figure(h2);

view(0, 0)

xlabel('x'); ylabel('y'); zlabel('z');

title('motion of camera frame');

print -dpng ass3\_motion.png

% this figure simply looks at the total distance travelled vs. image

% index

figure(3);

clf;

plot(linspace(0,imax,imax+1),d,'b-')

axis([0 105 0 21])

xlabel('image index');

ylabel('distance travelled [m]');

title('distance estimated by visual odometry');

print -dpng ass3\_distance.png

end

% this is the core function that computes motion from two pointclouds

function [C, r] = compute\_motion( p1, p2 )

% ------insert your motion-from-two-pointclouds algorithm here------

% Check if empty point cloud

if (size(p1, 2) == 0) || (size(p2, 2) == 0)

C = eye(3);

r = zeros(3, 1);

return

end

% Centriod of points

c1 = mean(p1, 2);

c2 = mean(p2, 2);

% Matrix to decompose

W = bsxfun(@minus, p2, c2) \* transpose(bsxfun(@minus, p1, c1)) / size(p1, 2);

% SVD decomposition

[V,~,U] = svd(W);

% Find rotation matrix

C = V \* [1 0 0; 0 1 0; 0 0 det(U)\*det(V)] \* transpose(U);

% Find translation

r = -transpose(C)\*c2 + c1;

% ------end of your motion-from-two-pointclouds algorithm-------

end