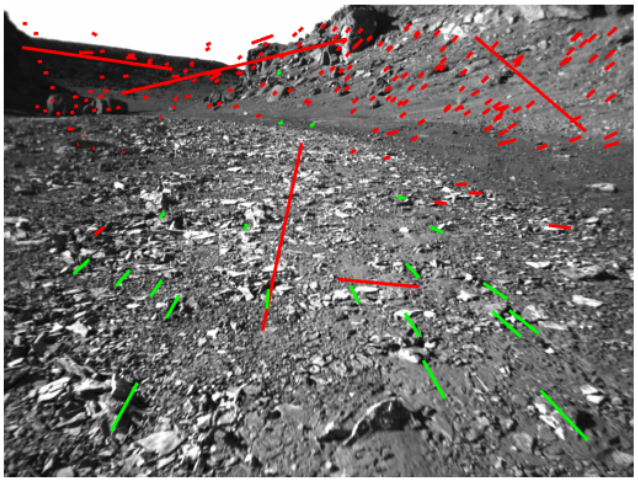
Assignment 3 Report
ROB521: Mobile Robotics and Perception

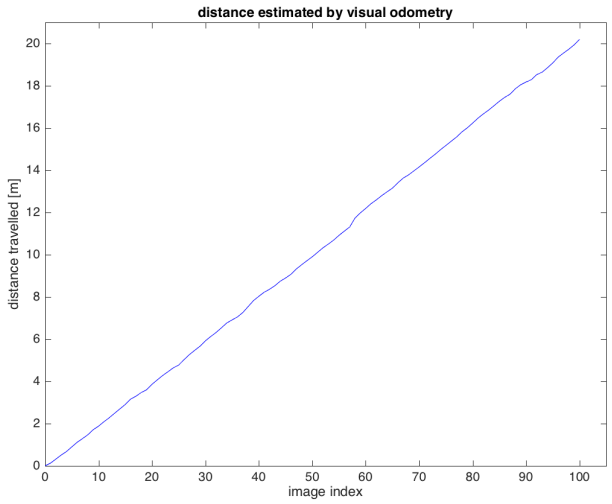
Name: Malcolm MacKay **Student number: 1002423308**

Question 1:

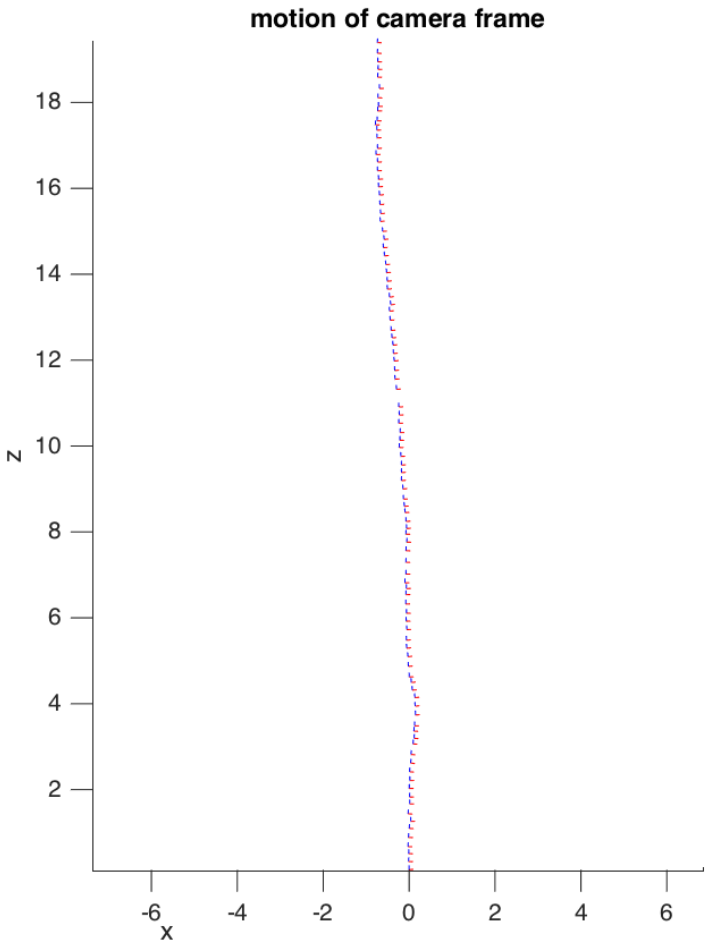
When running the code, we can see images of a the arctic with red and green lines drawn on them. These lines are used to track features in the images, and estimate the distance that the robot has moved in between taking the images. You can see an example of this below:



My distance results look identical to the solution. We can see the robot is travelling at a constant-ish speed due to how linear the resulting line is (i.e. the robot is moving approximately the same distance in between each image). This can be see below:



In order to get the proper motion of camera frame graph, the command 'rotate3d on' was used (as the solution has the results in X-Z coordinates, but running the assignment 3 code puts the results in X-Y coordinates). If we right click on the diagram and select X-Z coordinates, then we get exactly the same results as the solution. The motion of the camera frame show that the robot is mostly going in a straight direction along the X-Z plane. This can be seen below:



Appendix (Code)

Question 1:

```
% ======
% 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 all;

    % 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(best
        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) c
        plot3( [curraxis(1,2) curraxis(1,4)], [curraxis(2,2) curraxis(2,4)], [curraxis(3,2) c
        plot3( [curraxis(1,3) curraxis(1,4)], [curraxis(2,3) curraxis(2,4)], [curraxis(3,3) c
        axis equal;
        rotate3d on
        pause(0.01);
    end

    % finish off this figure
    figure(h2);
    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------

    %This is all in lecture 18 slide 18


    if size(p1) == [0 0]
        C = zeros(3);
        r = ones(3,1);
    else

        %1:Define Centroids and weights

        %w^j = 1 therefore w is just the number of columns
```

```
        w = size(p1,2);

        %again, w^j is 1, so the centroid equation simplifies to the sum of the
        %columns over w (i.e. the average of the columns)
        p1_c = sum(p1,2)./w;
        p2_c = sum(p2,2)./w;

        %2:Compute outer product matrix
        W_ba = ((p2 - p2_c)*(p1 - p1_c)')./w;

        %3:Use singular value decomposition
        [U, S, V] = svd(W_ba');

        %4: Compose final rotation and translation
        x = det(U)*det(V);
        X = [1 0 0;
             0 1 0;
             0 0 x];

        C = V * X * U';
        r = -C' * p2_c + p1_c;
    end


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

end
```