Project Report

Project 7: Visual SLAM using a single camera

Advanced Robotics, Spring Semester 2015,
University of Technology Sydney(UTS).

Student Name: Simone Magri
Student Number: 11797723

Project Supervisor: Dr Shoudong Huang

Introduction

I will begin by giving a brief description of the nature of the main dataset used in this project, which gives a critical background understanding to the work done to complete this project. The main dataset consists of a a series of sensor observations made at discrete time steps. In this case the sensor is a single lens camera and it will take a picture, thus make an observation, at these discrete time steps. So essentially the camera and device it is mounted to, a person holding the camera or our 'robot like' device, takes images with a single optical sensor.

The successive images will be processed using various well known algorithms to identify matching features in these images. Data associated with the matched image features is then fed into code which implements the SLAM(Simultaneous Localisation And Mapping) algorithm. The SLAM algorithm computes an estimate of the camera's location in 3D Euclidean space, that is it estimates camera pose; and at the same time it will use the matched images to build a map of the environment, that is determines the 3D Euclidean coordinates of our image features. This is all done while the is camera is moving through it's environment and at each time step.

Project Scope

Write a computer program to demonstrate feature matching of successive camera images using the SURF(Speeded Up Robust Features) algorithm. Followed by SLAM(Simultaneous Localisation and Mapping) using the matched image data, initial relative camera pose, and the ParallaxBA Bundle Adjustment algorithm. The essence of this can be demonstrated by using just two input images.

Input image data was provided by Shoudong Huang, along with the intrinsic camera parameters, like focal length and principle point.

The brief given by my supervisor, Dr Shoudong Huang, was that I would focus on understanding the inputs and outputs of the various functions that are required to complete this task, and write code using these functions and display their outputs. This is what I have done to the best of my ability and in the very limited time allowed for completion of this task.

Methods:

Provided to Shoudong Huang via email is a file called **11797723project.zip** which contains all the code and data related to this project. In this file, of most significance:

- the **final source code** is in just a single file called **fbmslam.cpp**
- fbmslam.cpp is built with a single makefile called **Makefile**
- the **images** referred to in this report are located in the **images/ directory**
- for **SLAM** the **ParallaxBA** the library and binary where built:
  - parallaxBA/ParallaxBA/trunk/linux/bin/ParallaxBA contains is the binary
  - parallaxBA/ParallaxBA/trunk/linux/lib/libParallaxBAImp.a is the library
- the only other significant code referred to in this report is **hello-world.cpp**
- **opencv3.0gold has been built in opencv/opencv3.0.0goldPlusContrib/ and was installed in /usr/local**

The following commands are sufficient for running fbmslam.cpp:

- **./fbmslam ./box.png ./box_in_scene.png**
- **./fbmslam ./image2.jpg ./image3.jpg**

The code :

- was all written in **C++** on an **Ubuntu 14.04 linux** system.
- **OpenCV**(Open Computer Vision library) version **3.0.0** was used for feature matching and determining initial relative camera pose.
- **ParallaxBA** library was used for **SLAM**(Simultaneous Localisation and Mapping), written by **Shoudong** Huang and **colleges**.

Datasets:

A set of five successive images of a car park were provided by Shoudong Huang. All images given by Shoudong Huang were used at one stage or another while achieving the aims of this project. However, the ones displayed in this report were most heavily used.

In addition to the given 'car park scene' images provided by Shoudong Huang I found box.png and box_in_a_scene.png, shown below, most useful in gaining an understanding of inliers and outliers.

General Process:

Initially, a huge amount of researching was done on the internet to determine the most appropriate programming libraries to use for this project. Quite often reaching dead ends or finally realising that the **OpenCV**(Open Computer Vision library) provided the most concise option to achieve a task.

In my search for the appropriate libraries I was keenly on the lookout for ones that had a well defined api, with good and easy to use documentation, and if possible example code that was readily available. I was also trying to minimise the number of different libraries I had to use because learning to build the library itself and in addition compile code with it could be quite time consuming. In addition, interfacing one library to another could sometimes be difficult and time consuming also.

I decided quite early on, and given the limited time frame and the focus of my brief, that I needed to find a lot of example code on the internet to achieve my task for this project. As you will see the number of internet references used is quite significant.

Bearing all this in mind my research culminated in the realisation that opencv was the best library to use for the feature matching of successive images and for determining the initial relative camera pose, both of which were required as input to the ParallaxBA SLAM algorithm. ParallaxBA was chosen for the SLAM algorithm mainly because is had a very well documented api and easily followed information on how to build the library and your code with it.

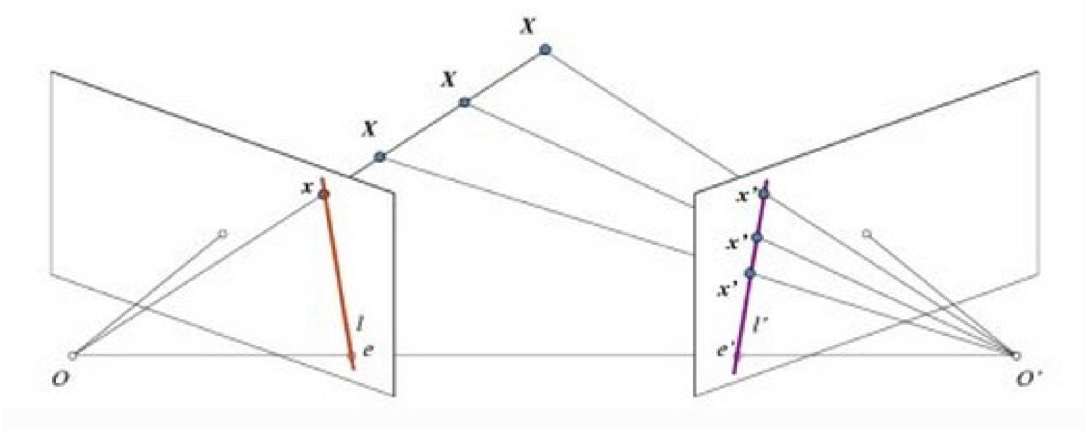
What I will describe in the rest of this document are the libraries and most significant functions required to achieve my task, look at the code in fbmslam.cpp for all the details. Along the way some background theory related to them is given so as to gain a greater understanding of the robotics related to this task. The order of explanations will follow the order in which these libraries or functions were required to achieve the task at hand. As essentially one part depended on the previous part for completion.

Feature Matching:

Shoudong Huang made me aware of the main image feature matching algorithms, **SIFT**(Scale-invariant feature transform) and **SURF**(Speeded Up Robust Features)[57][58].

I have chosen to use the SURF algorithm for feature matching. This algorithm was purely chosen because there was adequate example code easily available[1]. The algorithm does it's feature matching based on what are known as **keypoints** and **descriptors**. Matching is done by using the descriptor information from the two successive images. I have used the **Flann** based **matcher algorithm** for matching these two sets of descriptors. The matches coming out of this are then converted to a set of 2D pixel point vectors for each image, called image1 and image2. These vectors are used extensively in the many functions that follow and are required for the task at hand.

The matches are based on epipolar lines, refer to the Epipolar Geometry diagram, **epipolarGeometryGeneral.png** below, where they are the red and purple coloured lines. Where a point on image plane 1 labelled as **x**, is found on image plane 2 at **x'**, **x'** lies on the purple epipolar line. One can mount the same argument, in reverse, for the red epipolar line. Thus, the matching algorithm doesn't need to search the whole image for a matching point, just searching on the epipolar line is sufficient.



epipolarGeometryGeneral.png[11]

A brief description of this diagram is useful at this point as elements of it are referred to later:

- the epipolar line, ***l'***, is the line through the point ***x'*** and the **epipole** ***e'***.
- the epipolar line, ***l***, is the line through the point ***x*** and the **epipole** ***e***.
- the epipoles ***e*** and ***e'*** are the projections of the ***0'*** and ***0*** camera centres respectively.
- all epipolar lines pass through the epipole.

The results of my feature matching code follows, but after a few words about the steps required to build a working executable, **fbmslam**, with my code. And also after the discussion about calculating the essential matrix **E** as this is where we come across **inliers** and **outliers.** All three images regarding feature matching and inliners and outliers a best viewed and compared at one time and together.

I attempted to build the example code[1] found regarding the feature matching, but initially had errors. More specifically, the file:

```
#include "opencv2/nonfree/nonfree.hpp"
```

was not found. After some investigation it was noted that the directory /usr/include/opencv2/non-free could not be found on my ubuntu14.04 linux development system. This was further confirmed by the opencv documentation[2].

So now I had discovered that the SURF related functions in opencv are part of the '**non-free**' package, that was not installed with the standard **libopencv-dev** package. Installation is detailed below.

Installing opencv 2.4.8 on ubuntu 14.04:

Was simply a matter of:

- **apt-get install libopencv-dev**

Installing the 'nonfree' packaged:

This mostly consists of adding the **opencv-nonfree** package repository to your **sources.list** file on the development platform[3]

- **sudo add-apt-repository --yes ppa:xqms/opencv-nonfree**
- **sudo apt-get update**
- **sudo apt-get install libopencv-nonfree-dev**

After this I had numerous problems building my fbmslam executable with my Makefile:

So I downloaded the opencv hello-world.cpp to debug my issues. Some time after I had a clean and workable Makefile for the project[4][5][6][7].

One may wonder why SURF modules were removed from the default install of opencv(2.4.8 and 3.0.0).

SURF is an example of an algorithm that OpenCV calls “non-free”. It is a patented algorithm. However, it is freely available for academic and research uses. But you need a licence from the creators to use it commercially[27]. So it has been decided by the opencv developers that these modules are not installed by default and you need to explicitly enable them when compiling and installing OpenCV to obtain access to them.

Calculate the Essential Matrix E[60]:

The results of my feature matching code are best shown after a discussion that sees how the Essential matrix(related to our image matches) is determined. In addition, calculating the essential matrix is very important as we will see later how it is used to determine the ***rotation matrix R*** *and* ***translation matrix t,*** *which facilitate us in determining the initial relative camera pose. Remember that the* ***feature match data, initial relative camera pose,*** *and* ***camera calibration*** *matrix are the only* ***required inputs*** *to the* ***ParallaxBA SLAM*** *algorithm.*

Part of Shoudong Huang's brief was also to remove outliers from the matched feature data by using the ***RANSAC****(Random Sample Consensus) algorithm.* ***Outliers*** *in these images are the false matches between a successive pair of images. Also, he required that the essential matrix be calculated using the* ***five-point*** *or* ***eight-point*** *algorithm. Note that the five-point algorithm used here requires a set of five matched image points at a minimum. Following is the single opencv 3.0.0 function required to do all this[60].*

```
E = findEssentialMat(image1, image2,
                     focalLength, principalPoint,
                     FM_RANSAC, probability, threshold, mask);
```

- image1, image2 are the 2D pixel (x,y) point vectors resulting from the feature matching described earlier
- focalLength and principalPoint are the 'focal length' and 'principal point' of the camera, determined from the given **Camera Calibration Matrix K**, shown and described later later.
- FM_RANSAC indicates the **RANSAC** algorithm is to be used to calculate the **Fundamental Matrix F** and calculate the mask value. Note that using the ransac algorithm requires at least **8 matching pairs of points** in image1 and image2.
  - threshold is used for RANSAC. ***It is the maximum distance from a point to an epipolar line, in pixels, beyond which the point is considered an outlier*** and is not used for computing the final fundamental matrix, I used the default value here.
  - probability is also used for RANSAC. It specifies a desirable level of confidence (probability) that the estimated matrix is correct, I used the default value here.
- mask is a N x 1 matrix, one entry per image point pair in image1 and image2.
  - if the matrix element/row is =1 then the corresponding image point pair is an **inlier**
  - if the matrix element/row is =0 then the corresponding image point pair is an **outlier**

The reason why the **Fundamental Matrix F** is calculated here is because it is **related to the Essential matrix E** as follows:

$E = K'^{T}FK$ where **K** and **K'** are the intrinsic calibration matrices of the two images involved, in our case **K=K'** as we are using the same camera from image to image.[54][55]

So finally this is used to calculate the **Essential Matrix E** and it does so using the **five-point** algorithm.

Again, **outliers** in these images are the false matches between a successive pair of images. Consequently, **inliers** are correct matches between a successive pair of images. The **inliers** are also defined by the **epipolar constraint**[47]:

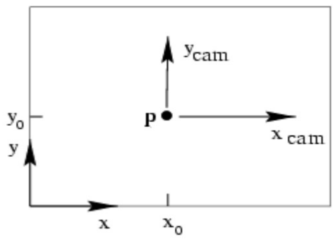
$$x'Fx=0;$$

where **x** is the point on one image plane while **x'** is the corresponding point in the other image plane and **F** is the fundamental matrix of course. Also note that the epipolar lines described earlier with the **epipolarGeometryGeneral.png** image also follow the equation **$l' = Fx$**[47]**.** Hence the importance of the fundamental and essential matrices. And thus, the epipolar line helps the search for corresponding features in both images, and allows for the detection of wrong correspondences. One should also note that the term **x'Fx** need not equal to **0** exactly, but it should be close to zero[12].

Camera Calibration Matrix and Principal Point[47]**:**

$$K = \begin{bmatrix} \alpha_x & 0 & p_x \\ 0 & \alpha_y & p_y \\ 0 & 0 & 1 \end{bmatrix}$$

Non-square pixels



$[p_x, p_y]$ is the coordinates of the principle point in image plane

For this project, the calibration matrix **K** was given by Shoudong Huang:

- α_x = 923.5295
- α_y = 922.241
- focalLength = sqrt(pow(α_x,2) + pow(α_y,2));

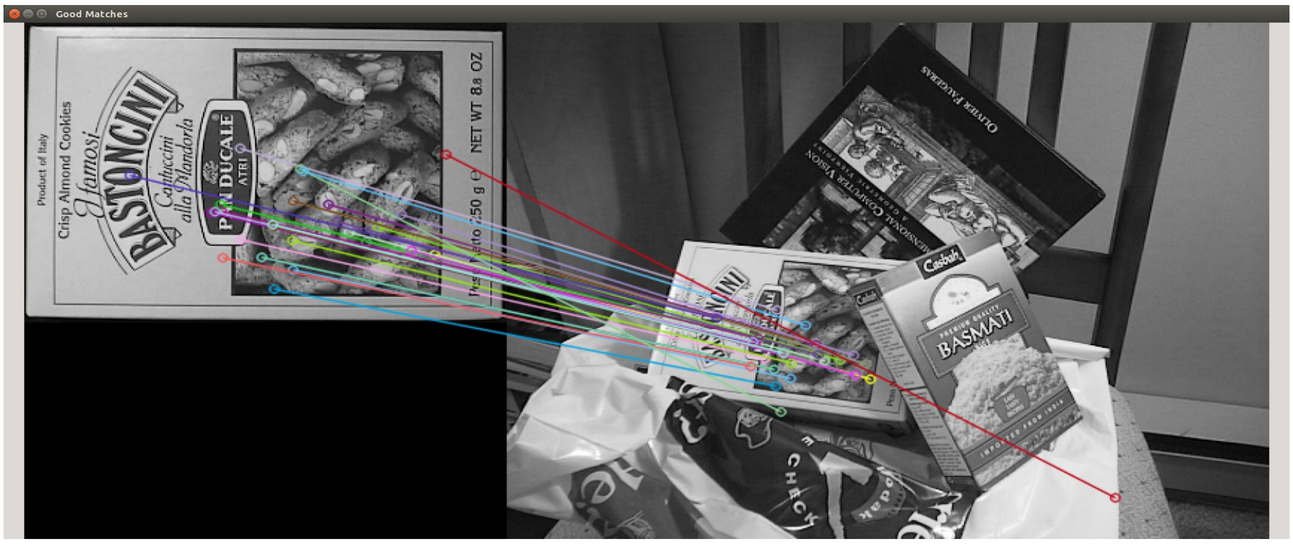
- p_x = 507.2222
- p_y = 383.5822
- principalPoint = [p_x, p_y]

- α_x and α_y represent the **focal length** in **pixels**, in x and y directions in the image plane.
- [p_x, p_y] is the coordinate of the **principle point** or **centre** of the image plane, in **pixels.**

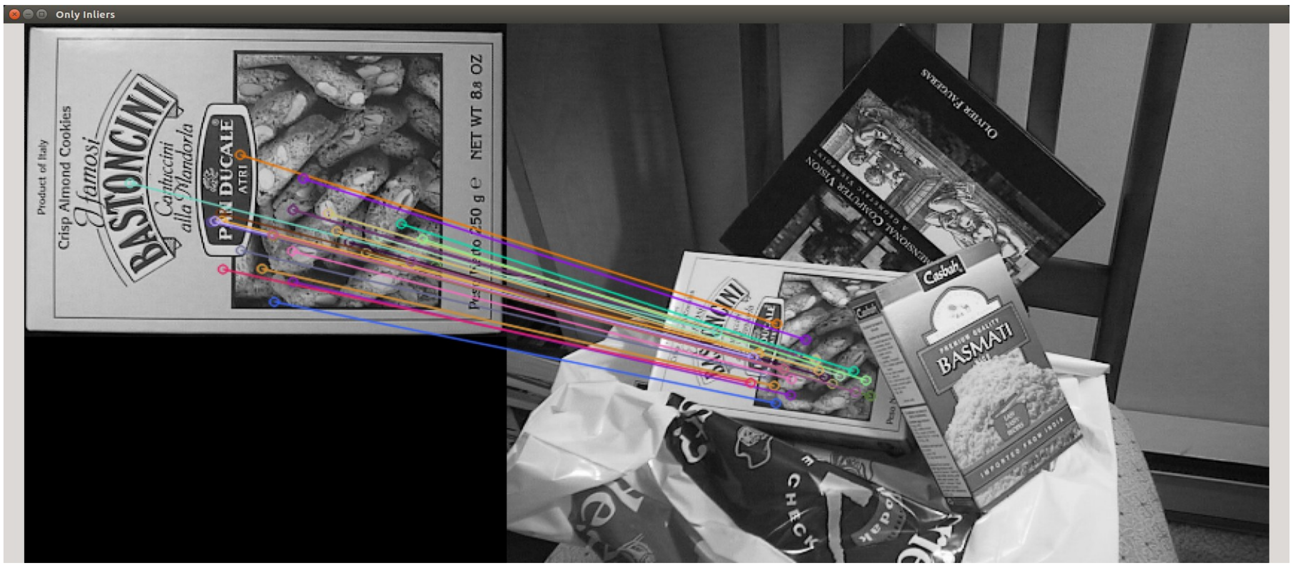
Feature Matching Results:

I found it most useful when developing the feature matching code to use a pair of 'box based images', box.png and box_in_scene.png[61][62]for testing. The results are the following three images, the first, **goodMatchesBox.png** shows all the matches found by the matching function between the LHS box image and the RHS box scene, it includes both inliers and outliers. The second, **onlyInlers.png,** only shows the inliers. And the third, **onlyOutliers.png,** only shows the outliers. The command to produce these is: **./fbmslam ./box.png ./box_in_scene.png**

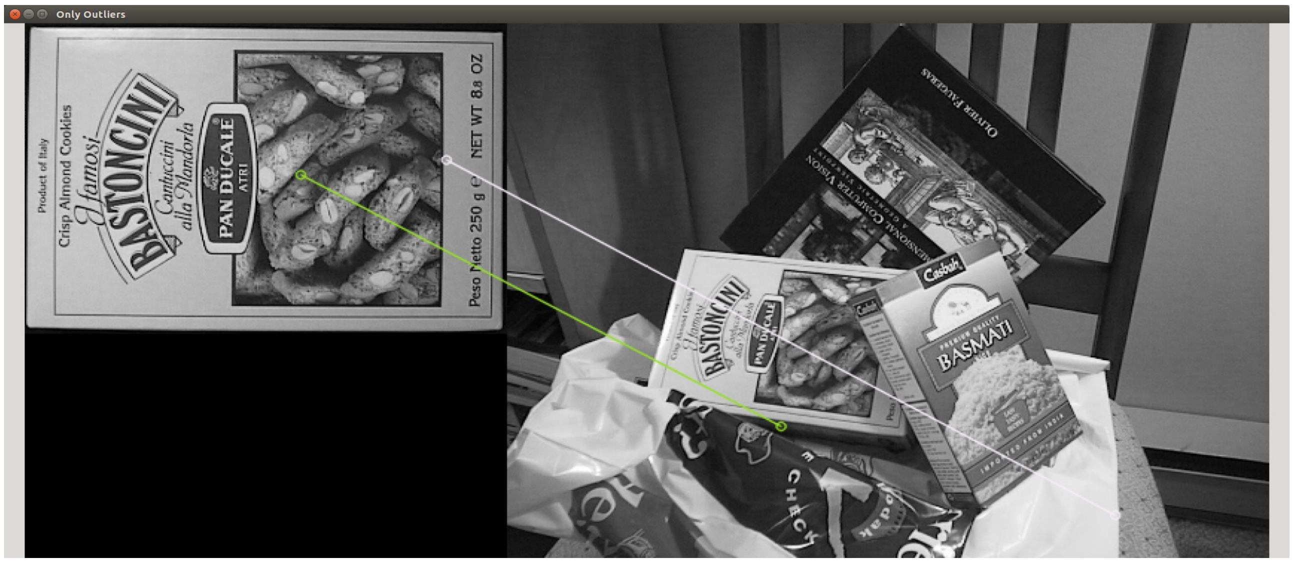
Then another pair of images from the target dataset, given by Shoudong Huang, were used to produce **goodMatchesCarParkImg2Img3.png** where we have both inliers and outliers displayed. **onlyIniersCarParkImg2Img3.png** displays just inliers and **onlyOutliersCarParkImg2Img3.png** displays only the outliers. The command to produce these is: **./fbmslam ./image2.jpg ./image3.jpg.** Note also that image2.jpg is aka CAMERA_RIGHT_1226322046.524413.jpg and image3.jpg is aka CAMERA_RIGHT_1226322046.924674.jpg in the target dataset.



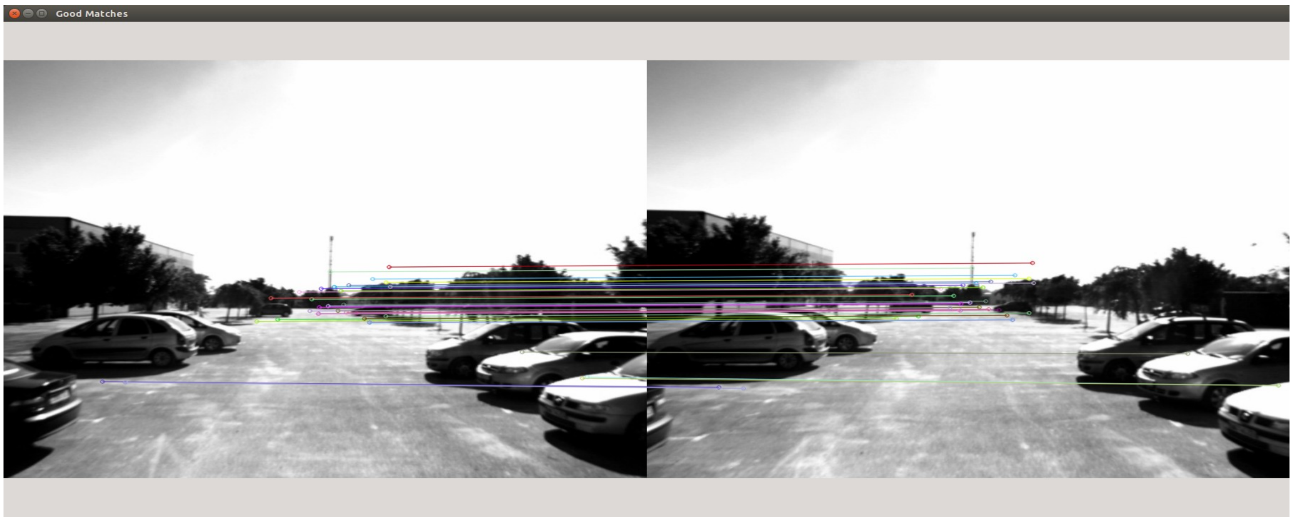
goodMatchesBox.png



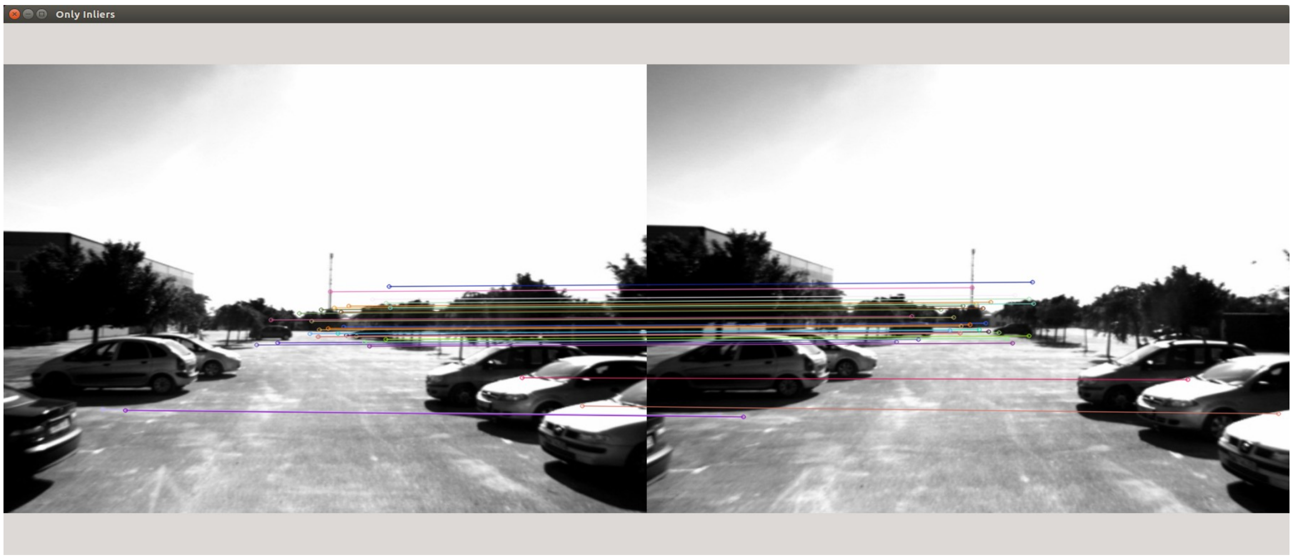
onlyInlers.png



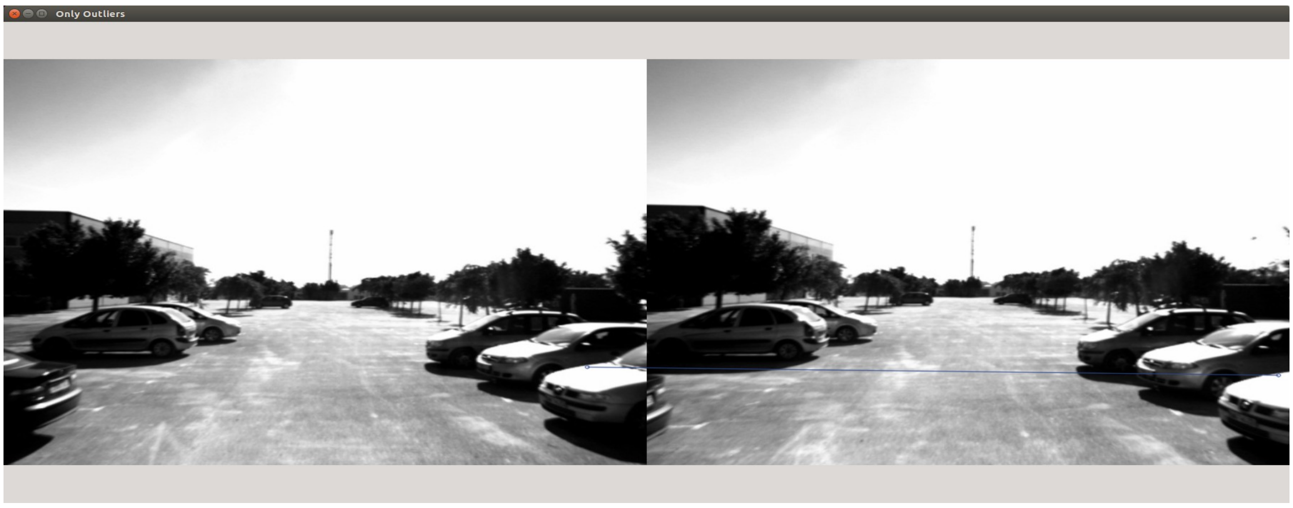
onlyOutliers.png



goodMatchesCarParkImg2Img3.png



onlyIniersCarParkImg2Img3.png



onlyOutliersCarParkImg2Img3.png

Using the Essential Matrix E to determine the relative camera pose:

OpenCV3.0.0 has a function that helps us easily determine the relative camera pose. This function does not exist in opencv2.4 and I had to especially install opencv3.0 to use it. It is an incredibly compact way to determine camera pose, which I found very appealing as I did not have time to learn a lot of new maths. The function description follows, along with the relevant minimal maths theory required.

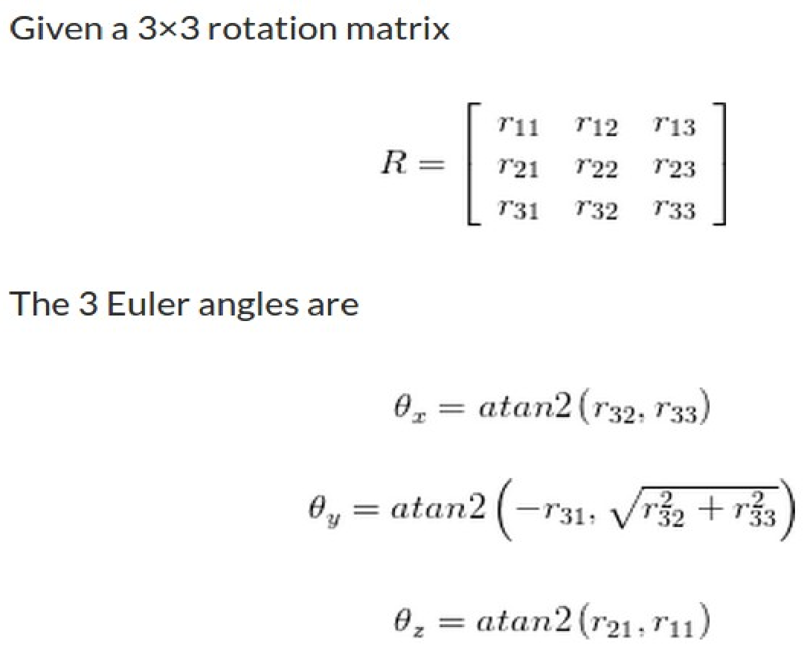
```
recoverPose(E, image1, image2,
            R, t,
            focalLength, principalPoint,
            mask);
```

- Outputs **Rotation Matrix**, **R**, *used to recover pose.*
- Outputs **Translation Matrix**, **t**, *used to recover pose.*
- **Essential Matrix E** as **input**, as seen earlier.
- image1 and image2 pixel (x,y) points, as seen earlier.
- focalLength, principalPoint, as seen earlier.
- **mask**, as seen earlier but ***only inliers will be used to recover pose***

Decomposing the Rotation Matrix, **R**, we get the **Euler Angles** thetaX, thetaY, thetaZ as shown below:

- **thetaX** is the rotation around the X axis, in a counter clockwise direction.
- **thetaY** is the rotation around the Y axis, in a counter clockwise direction.
- **thetaZ** is the rotation around the Z axis, in a counter clockwise direction.

The **translation matrix**, **t**, represents the centre of the camera[x, y, z]^T



Given a 3×3 rotation matrix

$$R = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix}$$

The 3 Euler angles are

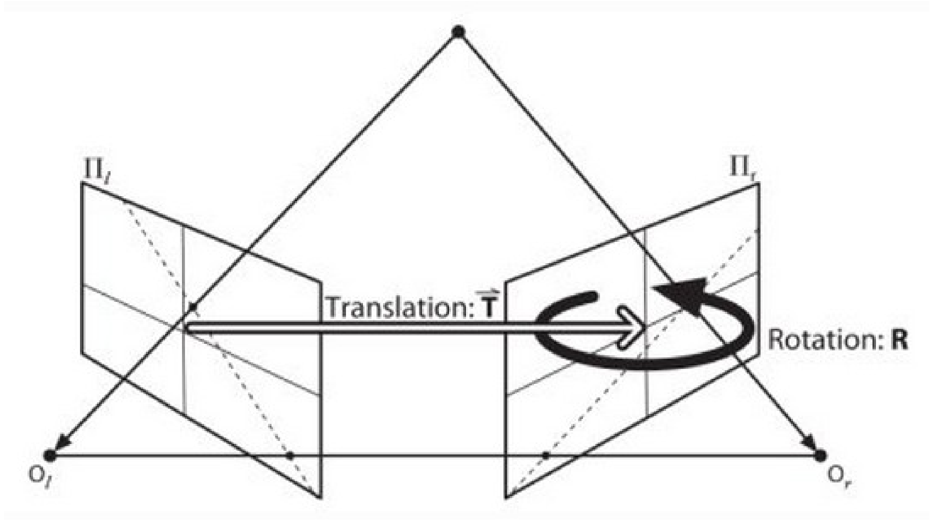
$$\theta_x = atan2\left(r_{32}, r_{33}\right)$$

$$\theta_y = atan2\left(-r_{31}, \sqrt{r_{32}^2 + r_{33}^2}\right)$$

$$\theta_z = atan2\left(r_{21}, r_{11}\right)$$

decompsingARotationMatrix.png

Hence thetaX, thetaY, thetaZ and t are the relative camera pose that we input into the SLAM algorithm ParallaxBA. This describes the location of the second camera relative to the first camera in global/world coordinates. As shown in figure **rotationAndTranslation.png** below. So the translation is related to the change in x, y, z coordinates of the camera and the Euler rotations angles give the new orientation.



rotationAndTranslation.png[11]

Projection Matrix:

Also of significance here, and for completeness, is that **R** and **t** can be used to determine the **projection matrix**, **P**:

P = K[R | t]

- where **R** is rotation matrix between cameras as seen earlier
- where **t** is translation matrix between cameras as seen earlier
- where **K** is camera calibration matrix, which can be determined from the **focal length** and **principle point** of the the camera image plane. However, an empirical method can also be used.

Further of importance **x = PX;** where, **x=(u,v)** projected point on image of, **X = [x,y,z]** a 3D Euclidean world point[47].

Initially, i thought opencv version 2.4.8, the default version of opencv on ubuntu 14.04. was all I needed. However, I had, all this time, in fact been looking at the documentation for the **recoverPose()** opencv function on the **opencv 3.0.0** documentation pages **not** the **opencv 2.4.8** function documentation pages. So to use this function I had to install opencv 3.0.0. Below is how I went about it. ***I have made a point of detailing such processes here and in other parts of this document as sometimes days were lost before my development system was operational again.***

However, installing opencv 3.0.0 fixed one problem but it did create another problem. In that it required me to **port** the existing fbmslam.cpp from opencv2.4.8 to opencv3.0.0, as there had been a number of **api changes** to the SURF based algorithm functions; many internet sources were used to solve this problem[25][26][27][28][29].

Installing opencv3.0.0 on ubuntu 14.4:

To do this I downloaded the **.zip** from the **git repository** and I eventually worked out how to install **opencv3.0.0gold** on my development machine, quite a number of internet sources where referred to about this process[18][19][20]. Note that this build is located at:

- **opencv/opencv3.0.0goldPlusContrib**/ and was installed in **/usr/local**

I also used internet sources to learn how to build an executable with the new library, a hello-world.cpp example was followed in fact. The fbmslam.cpp Makefile reflects the outcome here.[21]

Again there was an issue with needing to separately build the **non-free modules** in **opencv 3.0.0**[22][23][24]:

- This meant I needed to build the **opencv_contrib/ modules**(new non-free modules)**.** Source was again downloaded from the git repository and built.
- Of significance, build errors were solved by commenting out three lines of code in **opencv_contrib/modules/ximgproc**... directory. Given I wouldn't be using this module I just hacked it to make it build.
- **pkg-config --modversions** indicated that I now had version 3.0.0 of opencv installed, as required.
- hello-world.cpp built ok with existing makefile.
- fbmslam.cpp did not build straight away, a number of compile bugs needed to be fixed, basically needed to port to new SURF api in opencv3.0.0.

Bundle adjustment:

Given our **set of images** depicting a number of 3D points, that is, our features: bundle adjustment **simultaneously** refines the 3D coordinates of our features(**Mapping**) as well as camera poses(**Localisation**). Hence, bundle adjustment is used to provide SLAM, simultaneous localisation of the camera pose while building a map of our environment.[32]

It does this while minimising the re-projection error of all the 3D features, that is, minimises the error between the observed and predicted image locations. It uses a least squares based iterative algorithm like LM (Levenberg-Marquardt). LM was trailed here, popular because it's easy to implement and converges quickly. I suspect it utilises the projection matrix, discussed earlier, to do this[36].

I have chosen to investigate the **ParallaxBA**, **Parallax Bundle Adjustment**, implementation of this algorithm[36][32]. More specifically, ParallaxBA is, Bundle Adjustment using Parallax Angle Feature Parameterisation. The algorithm and library api are well documented[31][32], source code is also readily available[33].

But of key importance is that the **inputs** to the ParallaxBA algorithm, they are:

- the matched image points, or feature locations in pixels(x,y)
- camera calibration matrix, also in pixels
- as well as the initial values of the camera poses: given by the Euler angles derived from the rotation matrix(thetaX,thetaY,thetaZ) and the centre of the camera given by the translation matrix, t[x,y,z]

Also, he **outputs** of the ParallaxBA algorithm are:

- the **optimised 3D feature locations** as well as the **camera poses**.

Progress on bundle adjustment code:

I got as far as building the static library and binaries for ParallaxBA, and they are located in:
parallaxBA/ParallaxBA/trunk/linux/lib/ParallaxBAImp.a and
parallaxBA/ParallaxBA/trunk/linux/bin/ParallaxBA

I also ran a few of the examples such as, for the Malaga dataset :

```
ParallaxBA -cam Cam170.txt calib cal170.txt -fea Feature170.txt -solve LM -report report.txt
```

I ran out of time while I was building and debugging my source for use with the ParallaxBA library and only a brief look at the source code was made, but I believe implementing ParallaxBA SLAM may have just been a matter of integrating the following line into my code:

```
pba_run("-cam camPose.txt", "-fea features.txt", "-calib K.txt", "-solve LM",
        "-report report.txt", "-pose oCamPose.txt", "-3D oFeatures.txt")
```

Building ParallaxBA:

This package was not present on my base ubuntu14.4 linux development environment, so following are the steps taken to install it[56]. Any linux commands not familiar to the reader can be understood by typing '**man** command-name' on the command line of any ubuntu 14.04 linux system, or determined by a search of the internet. However, I have given very brief explanations where I think it was necessary.

On Linux platform, install and run ParallaxBA as follows**[56]**:

- **sudo apt-get install cmake**, already there.
- **sudo apt-get install libeigen3-dev**, already there.
- **sudo apt-get install libsuitesparse-dev**, had to install.

Download source from svn repository:

This was quite slow, it took about 10minues, but that time was mostly due to downloading example dataset (.rar) files:

svn co https://svn.openslam.org/data/svn/ParallaxBA

Briefly **cmake** is a cross-platform program used to configure the build system for a project, in scripts. Project configuration settings may be specified on the command line with the -D option.(cmake man page).

Of note, I found lots of these types of errors produced during the build, which I found a bit surprising:

lu1/smagri/uni/subj/ar/project/proj/parallaxBA/ParallaxBA/trunk/linux/src/ParallaxBAImp/ParallaxBAImp.cpp:2485:10: ***warning: unused variable 'bT' [-Wunused-variable]***

Build via cmake:

- **cd linux**
- **mkdir build**
- **cd build**
- **cmake ../**
- **make**
- **sudo make install**

Sample execution:

Uncompress Village.rar**[33]**:

- **apt-get install unrar**
- **unrar x Village.rar**

cd Village
ParallaxBA -cam Cam90.txt -fea Feature90.txt -calib cal90.txt -solve LN -report report.txt

Future work:

- Ideally one would write code with a framework where images are read on the fly from the camera and SLAM output is visible in real time on a computer screen.

- Or at the very least write code to read two successive images from your computer***
  - perform feature mapping with the removal of outliers
  - compute the relative camera pose
  - and input these into the SLAM algorithm
  - display the outputs of the SLAM algorithm
  - then repeat again from the beginning***

- **Of course the first thing that should happen though is to build parallaxBA SLAM support into fbmslam.cpp.** It may not require much more work for this anyhow, probably needed another couple of days to work on this.

- I also need to **verify** the output results of fbmslam.cpp, aside from the images shown above, have sensible values, this applies to:
  - rotation matrix R
  - translation matrix t
  - projection matrix P
  - and the Euler angles
  -
  - sample outputs are in output.box and output.image23

References:

[1] http://docs.opencv.org/doc/tutorials/features2d/feature_homography/feature_homography.html#feature-homography
[2] http://docs.opencv.org/modules/features2d/doc/common_interfaces_of_feature_detectors.html?highlight=featuredetector#FeatureDetector
[3] http://stackoverflow.com/questions/27481849/include-nonfree-opencv-2-4-10-on-ubuntu.
[4] https://github.com/adamb/opencv-samples/blob/master/hello-world.cpp
[5] http://stackoverflow.com/questions/9870297/makefile-to-compile-opencv-code-in-c-on-ubuntu-linux
[6] http://www.cs.iit.edu/~agam/cs512/lect-notes/opencv-intro/opencv-intro.html#SECTION00025000000000000000
[7] http://docs.opencv.org/doc/tutorials/introduction/linux_eclipse/linux_eclipse.html
[8] http://stackoverflow.com/questions/24456788/opencv-how-to-get-inlier-points-using-findhomography-findfundamental-and-ra
[9] http://www.mrpt.org/tutorials/programming/maths-and-geometry/ransac-c-examples/.
[10] https://github.com/MRPT/mrpt/blob/master/samples/ransac-demo-applications/test.cpp
[11] http://docs.opencv.org/master/da/de9/tutorial_py_epipolar_geometry.html#gsc.tab=0
[12] https://www8.cs.umu.se/kurser/TDBD19/VT05/reconstruct-4.pdf.
[13] http://stackoverflow.com/questions/24456788/opencv-how-to-get-inlier-points-using-findhomography-findfundamental-and-ra
[14] http://answers.opencv.org/question/12295/compute-inliers-for-a-homography/
[15] https://github.com/Itseez/opencv/blob/master/modules/calib3d/src/five-point.cpp
[16] http://docs.opencv.org/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html
[17] http://opencv-python-tutroals.readthedocs.org/en/latest/py_tutorials/py_calib3d/py_epipolar_geometry/py_epipolar_geometry.html
[18] http://rodrigoberriel.com/2014/10/installing-opencv-3-0-0-on-ubuntu-14-04/
[19] http://www.bogotobogo.com/OpenCV/opencv_3_tutorial_ubuntu14_install_cmake.php
[20] http://www.humbug.in/2015/compile-and-install-opencv-3-0-0-on-ubuntu-14-04/
[21] http://www.cs.iit.edu/~agam/cs512/lect-notes/opencv-intro/opencv-intro.html#SECTION00024000000000000000
[22] http://stackoverflow.com/questions/27418668/nonfree-module-is-missing-in-opencv-3-0
[23] https://github.com/itseez/opencv_contrib/
[24] http://answers.opencv.org/question/52001/how-to-compile-nonfree-module-in-opencv-30-beta/
[25] https://github.com/Itseez/opencv_contrib/commit/97cf5ade023466d90861b696322a1e479a5bab0c
[26] http://stackoverflow.com/questions/27533203/how-do-i-use-sift-in-opencv-3-0-with-c
[27] http://www.pyimagesearch.com/2015/07/16/where-did-sift-and-surf-go-in-opencv-3/
[28] https://github.com/Itseez/opencv_contrib/tree/master/modules/reg/samples
[29] http://docs.opencv.org/master/d5/df7/classcv_1_1xfeatures2d_1_1SURF.html#gsc.tab=0
[30] http://www.pyimagesearch.com/2015/07/16/where-did-sift-and-surf-go-in-opencv-3/
[31] https://svn.openslam.org/data/svn/ParallaxBA/info/info.xml.
[32] http://services.eng.uts.edu.au/~sdhuang/Documentation_ParallaxBA.pdf.
[33] http://www.cyberciti.biz/faq/open-rar-file-or-extract-rar-files-under-linux-or-unix/
http://ubuntuforums.org/showthread.php?t=2043444
[34] http://docs.opencv.org/2.4/doc/tutorials/core/mat_the_basic_image_container/mat_the_basic_image_container.html
[35] http://stackoverflow.com/questions/10936099/matrix-multiplication-in-opencv
[36] https://en.wikipedia.org/wiki/Bundle_adjustment
[37] http://docs.opencv.org/2.4/doc/tutorials/core/how_to_scan_images/how_to_scan_images.html
[38] https://sites.google.com/site/marcelopetry/tutorials/mat-matrix---element-access
[39] http://stackoverflow.com/questions/1844736/accesing-a-matrix-element-in-the-mat-object-not-the-cvmat-object-in-opencv-c
[40] https://sites.google.com/site/marcelopetry/tutorials/mat-matrix---element-access

[41] http://www.prism.gatech.edu/~ahuaman3/docs/OpenCV_Docs/tutorials/basic_0/basic_0.html
[42] http://answers.opencv.org/question/8842/rotation-matrix-to-euler-angle/
[43] http://stackoverflow.com/questions/21569934/resolving-rotation-matrices-to-obtain-the-angles
[44] http://stackoverflow.com/questions/15022630/how-to-calculate-the-angle-from-roational-matrix
[45] http://nghiaho.com/?page_id=846
[46] http://planning.cs.uiuc.edu/node102.html
[47] Shoudong Huang's Lecture notes on Visual SLAM:
https://online.uts.edu.au/bbcswebdav/pid-654676-dt-content-rid-
4965984_1/courses/49274/vision_Shoudong.pdf
[48] https://svn.openslam.org/data/svn/ParallaxBA/
[49] https://github.com/Itseez/opencv/blob/master/modules/calib3d/src/five-point.cpp
[50 https://en.wikipedia.org/wiki/RANSAC
[51] https://en.wikipedia.org/wiki/Outlier
[52] https://en.wikipedia.org/wiki/Outlier
[53] https://en.wikipedia.org/wiki/Eight-point_algorithm
[54] https://en.wikipedia.org/wiki/Essential_matrix
[55] https://en.wikipedia.org/wiki/Fundamental_matrix_(computer_vision)
[56] https://svn.openslam.org/data/svn/ParallaxBA/trunk/README.txt
[57] https://en.wikipedia.org/wiki/Scale-invariant_feature_transform
[58] https://en.wikipedia.org/wiki/Speeded_up_robust_features
[59] http://docs.opencv.org/master/d2/d29/classcv_1_1KeyPoint.html#gsc.tab=0
[60]http://docs.opencv.org/3.0-
beta/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html#findessentialmat
[61]http://www.google.com.au/imgres?imgurl=https://raw.githubusercontent.com/introlab/find-
object/master/bin/box.png&imgrefurl=http://introlab.github.io/find-
object/&h=223&w=324&tbnid=yLsuHpxilBDv1M:&docid=I9K1t9gbGkFVPM&ei=6oY9Vsr6LcOx0ATkoKi
YBQ&tbm=isch&ved=0CCAQMygAMABqFQoTCMqXrp_F_cgCFcMYIAodZBAKUw
[62]https://www.google.com.au/search?
q=box+png+and+box_in_scene.png+opencv&es_sm=122&biw=1838&bih=991&tbm=isch&imgil=Qkokd
v8lAouakM%253A%253BI9K1t9gbGkFVPM%253Bhttp%25253A%25252F%25252Fintrolab.github.io
%25252Ffind-object%25252F&source=iu&pf=m&fir=Qkokdv8lAouakM%253A%252CI9K1t9gbGkFVPM
%252C_&usg=__UxcJNuTeeah0HkpJQbLx3opQc6w%3D#imgrc=Qkokdv8lAouakM
%3A&usg=__UxcJNuTeeah0HkpJQbLx3opQc6w%3D