**Pose Estimation**

The six degree of freedom pose of the center of mass of the UAV in the world frame is (x, y, z, Φ, θ, ψ), where Φ is roll angle, θ is pitch angle, and ψ is yaw angle.

Body frame is the center of mass of the UAV. xb is the preferred forward direction and zb is perpendicular to the plane of the rotors pointing vertically up during perfect hover.

Use ZXY Euler angles to model rotation in the world frame. To translate from body frame to world frame:

* rotate about the zw axis by the yaw angle, ψ
* rotate about the intermediate y axis by the pitch angle, θ
* rotate about the xb axis by the roll angle, Φ

Rotation matrix to transform from body frame to world frame is

R = R(zw, ψ)R (y, θ) R(xb, Φ), where the Rs are elementary rotation matrices with respect to the x, y, and z (see [3] pages 4-9 for additional details).

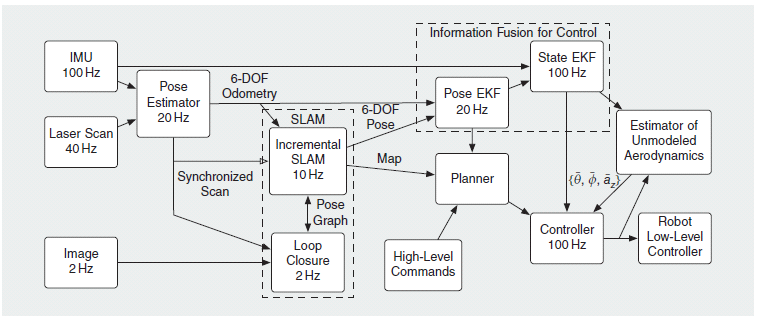


Figure 1. The architecture diagram with update rates from [1].

1. **Altitude Sensing**

The altitude sensing relies on the upward and downward deflected laser scans and the pressure sensor from the IMU on the Ardupilot, in our case. The following is based on [1, 17, 18]. [18] and the OpenQuadrotor.org code /Odometry/Lasertransformater\* do this for a single downward facing mirror. We need to do it for both downward and upward facing mirrors.

General Notes:

: LRF endpoint position from beam not deflected by mirrors

: The homogenous vector in 3D space for beams not deflected by mirrors.

: The point in the world frame for beams not deflected by mirrors.

: homogeneous transformation matrix form the IMU reference frame to the LRF reference frame, known from calibration measurement.

, where is the angle of the individual LRF beams not deflected by a mirror.

, where is a beam not deflected by the mirrors.

**Altitude Algorithm**:

: Distance of a single LRF beam along the x-y plane measured in the LFR coordinate frame for the upward mirror.

: Distance of a single LRF beam along the x-y plane measured in the LFR coordinate frame for the downward mirror.

: The homogenous upward mirror vector in 3D space.

: The homogenous downward mirror vector in 3D space.

: The point in the world frame for beams deflected by upward mirror.

: The point in the world frame for beams deflected by downward mirror.

: transformation matrix from the IMU reference frame to the virtual laser position that accounts for the effect of the upward mirror.

: transformation matrix from the IMU reference frame to the virtual laser position that accounts for the effect of the downward mirror.

: time dependent transformation from the world to the IMU, only computed from pitch and roll.

*PressureReading:* Pressure sensor output from ArduPilot IMU

*MaxVariance*: The variance threshold

*Altitude:* Estimated altitude

, where is the angle of the individual downward LRF beams

Calculate the variance of the downward pointing laser based on a moving average.

If ( {

Disregard downward scan

, where is the angle of the

individual upward LRF beams

Calculate the variance of the upward pointing laser

based on a moving average.

If {

Disregard upward scan

*Altitude* = Use onboard pressure sensor to estimate altitude

Lower robot height

}

else

return

}

else

*return*

Return *Altitude*

1. **2.5D Projection**

The first step is to compute the 2.5D projection, which uses the Laser Scan and the IMU reading (top left of Figure 1). The 2.5D projection is computed after obtaining the laser scan and the IMU reading, but prior to the Pose Estimation. The assumption is that the environment is a collection of vertical walls and horizontal ground planes, which are assumed to be piecewise constant.

2.5D projection Algorithm:

: The laser scan endpoints in the body frame.

: The 2.5 D laser scan projection.

IMU reading

: Deflected laser scan pointing up

: Deflected laser scan pointing down

Loop for all endpoints

If ((

Eliminate the scan from consideration

Else

Calculate the projection for the scan to 2.5D space such

that:

Return

1. **Pose Estimator**

The pose estimator has two steps. Step one estimates (x, y, ψ) using the ICP algorithm and the second step determines the height using Friegburg’s algorithm.

**II.a ICP Estimator**

The Iterative Closest Point (ICP) Algorithm (see [2] for an overview of the concept) finds the transformation between two sets of data points and provides an estimate of (x, y, ψ). This algorithm resides in the *Pose Estimator* process in Fig. 1. It will be necessary to improve on the speed of the basic algorithm.

The basic *ICP algorithm*:

*Input: A reference point cloud and the new data point cloud.*

*Output: An estimate of (x, y, ψ)*

*Step 1. Preprocessing cleans the data.*

*Step 2. Matches the associated points from the reference to the data using the neighbor search. It can use features to identify associated points.*

*Step 3. Weighting changes the importance of some pairs.*

*Step 4. Reject some of the pairs*

*Step 5. Compute the error for each pair and where they should be located.*

*Step 6. Find the best transformation (minimization)*

*Step 7. Loop back to step 2, unless there is convergence.*

Step 2 is very computationally intensive. Penn uses a grid-based search [4] to reduce the computation requirements of the closest point search. Another resource reduces the computational complexity [5], but I have not studied it enough to determine if it will reduce the computation enough (but it does go through the steps in more detail).

Chetverikov‘s paper [4] effectively breaks the point clouds down into regions (boxes). The paper uses squares, but the regions can be rectangles or arbitrary shapes. We can probably use squares or rectangles. This decomposition occurs prior to the neighborhood search.

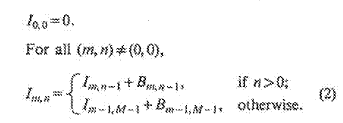
Chetverikov gives the algorithm for Boxing in his paper. It requires 2 passes, and has the predefined elements L1 which is the original non-ordered list of points, and the box data structure which is made up of a rearranged point list L2 and an index matrix Im,n. The basic algorithm is defined here:

*Pass 1:*

* *Step 1. Allocate an MxM size allocator array Bm,n which is to contain the number of points in each box. (the grid of all points is MxM).*
* *Step 2. Scan L1 and fill Bm,n. A point Ai is within the box whose indices are calculated using the following formula:*



* *Step 3. Fill Im,n using the following formula:*



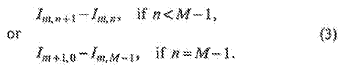
*Pass 2:*

* *Step 1. For all m and n, set Bm,n equal to zero.*
* *Step 2. Scan L1 again. Use the m and n coordinates calculated above as well as Bm,n and Im,n to fill L2. For L2 the first point of the (m,nth) box is indexed by Im,n, while the address of the subsequent points is controlled by Bm,n whose value is incremented each time a new point enters the box.*

After having boxed the point clouds, one can do a neighborhood search algorithm to determine neighboring points to a point A by using the following access procedure.

*Step 1. Compute the indices mi and ni of the box that contains A using the procedure above.*

*Step 2. Use the boxing data structure to retrieve the points Bj, where j=0,1,2,…qi-1 lying within nine boxes. (qi is used to denote all of the points in the nine boxes.) Im,n indexes the first point in the (m,n)th box, while the number of points is given by the following formula:*



*Step 3. Select the points Bj* *in the nine boxes that are within the 2D\*2D size neighborhood of Ai*

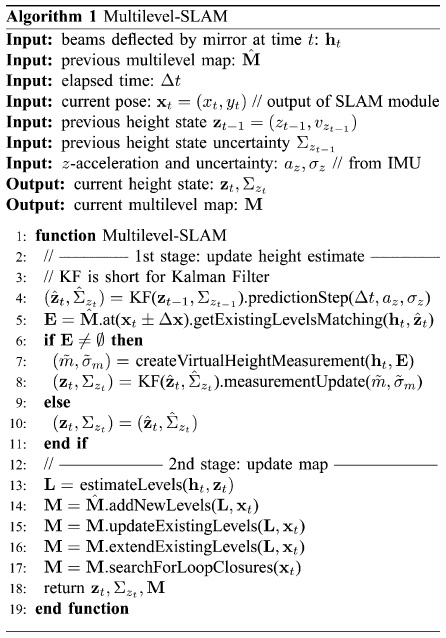


Figure 2. The Multilevel-SLAM algorithm from Univ. of Frieburg [6] that Penn uses a variant of the Kalman Filter, lines 3-11.

Penn estimates (Φ, θ) based solely on the IMU data.

**II.b Height Estimate**

The *Incremental SLAM* process completes portions of Freiburg’s algorithm, specifically, the matching (lines 5-10).

The output of the *Pose Estimator* and the *Incremental SLAM* processor are the inputs to the *Pose EKF* process. This process combines (fuses) the IMU data with the corrected laser range finder scans and provides altitude estimates. Penn uses a variant of work done at the University of Freiburg [6]. Note that Freiburg also uses mirrors with their LRF.

The input to the variant of the Kalman Filter (KF) [6], which actually uses two KFs. Frieburg’s paper does an excellent job of describing the algorithm in Fig 2, and the team needs to read this paper. Note also that [6] has a better picture of the mirrors they use for the LRF.

----------------------

The results of the ICP steps prior to step 5 are combined with the IMU data in order to correct the error in the ICP algorithm Step 5. The correction is done by aligning the incoming scans with the map. This comparison uses Chetverikov‘s boxing and searching algorithms.

This process uses an occupancy grid (as an example of occupancy grids pp. 281-290 of [7]) to track where items are located in the area. In my quick review of the papers, it is not clear from any of the papers which type of occupancy grid they are using. I suspect one that has a square grid with grid squares that are not too small, but also not too large. We can discuss what we think the grid sizes should be, but I suspect not smaller that the UAV, plus or minus the error in the LRF, and not so large that a window or a door would be missed.

A new layer of the multi-layered occupancy grid is created when a stable floor transition is detected by the pose estimator.

The output of the *Incremental SLAM* process is a Pose Graph. The Pose Graph includes a representation of the robot poses and the transitions between poses. The graph nodes are the individual robot pose positions and the links between nodes are the relationships between the poses. The pose note at time t0 is connected to the pose note at time t1, and may be connected to a pose node at some later time tx if the UAV has completed a loop and returns to the same position as represented by the node for t0. Figure 3 (in [1]) provides two examples of Pose Graphs that show a single loop (Figure 3.a) or multiple loops (Figure 3.b) through the environment.

The *Loop Closure* relies on SURF to detect features in the environment [9, 10]. Open source versions of SURF are available, and that is what we should use, we should not have to reimplement SURF.

SURF is a scale invariant feature detector that uses Haar wavelets to approximate differences of Gaussians filters and integral images, which quickly and efficiently generates the sum of values in a rectangular subset of a grid in the image, for convolution.

The basic algorithm overview is that the algorithm

* Is reduced into two parts: detector (or interest points) and descriptor (feature vector to “neighborhood” of detector).
* The descriptors are matched between different images based on distance, such as Euclidean
* SURF builds upon/combines previously existent detector/descriptor methodologies

The SURF binary files can be downloaded from the official SURF website [11]. In addition to the OpenSURF code (C++ or C#) [10], other versions are available [11, 12, 13, 14]. A few tutorials also exist [13, 14].

It is unclear what Penn’s fixed sized visual vocabulary, constructed off-line is. Based on Cummings and Newman [8], a set of features, using the concept of “Bag or words”, are created to represent locations in the environment. These features appear to be invariant to sensor location and direct matches, meaning that it can accommodate sensors at different locations and different elements in the environment that are transitional (e.g., bikes and cars). If this is indeed the case, there is a possibility that we may be able to directly use Penn’s visual vocabulary directly.

The “words”, features, are equally distinctive and are may not be independent. The features [8] are generated by:

1. Extracting the areas of interest from images using the Harris-affine detector.
2. Computing the SIFT descriptors around the areas of interest.
3. Mapping the SIFT space to words, features by
   1. clustering all the descriptors from a set of training images using a bottom-up (agglomerative) process.
   2. Quantizing the each SIFT in the images to is approximate nearest cluster center using a K-dimensional tree ([15] pages 738-740, [16]).

“Bag of words” representation of sensor data – each word is an attribute from a set (vocabulary) of size

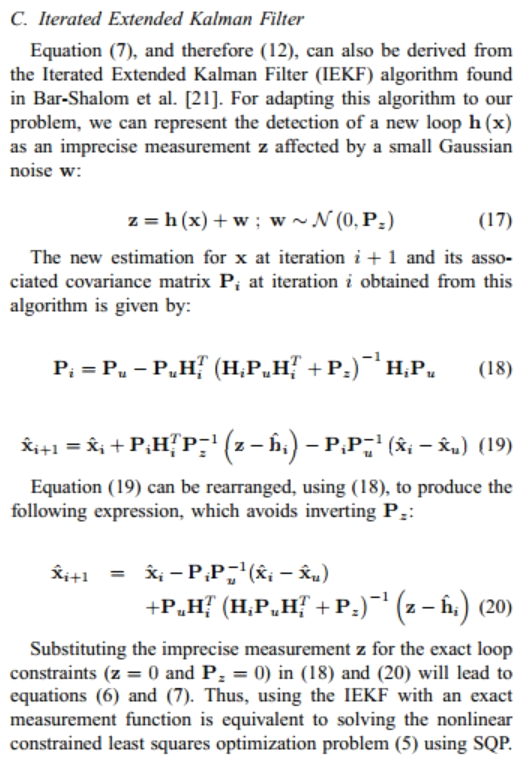
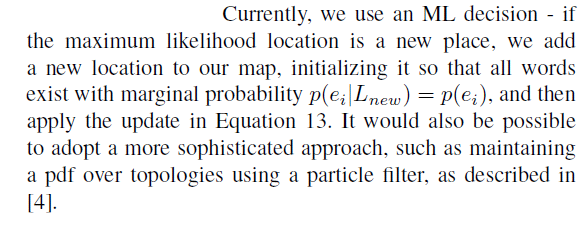
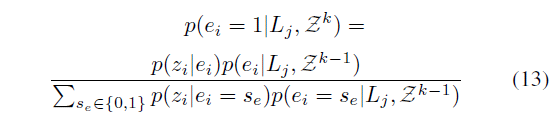
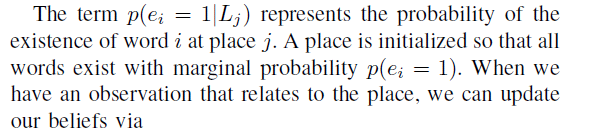
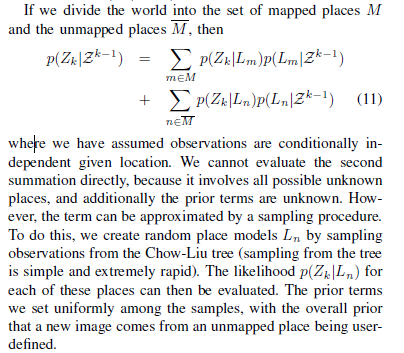
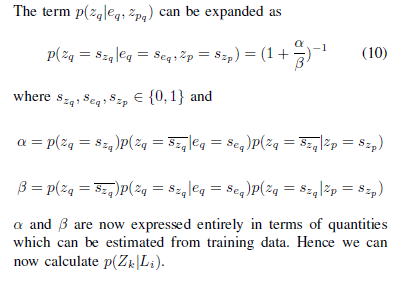
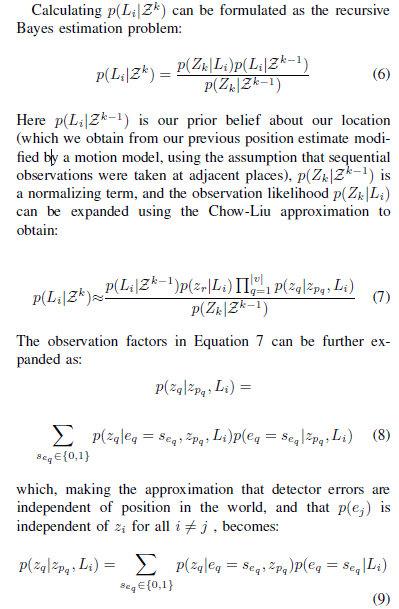
Observation at time denoted as , where is a binary variable indicating presence or absence of word of vocabulary. denotes set of all observations up to time .

Learning the observation model (compute once offline using set of training images and chosen features)

* Construct mutual information graph from data
* Construct complete graph of *n* nodes (one for each word in vocabulary)
* Set weight of each edge ( to the value of mutual information
* ,
* Find maximum weight spanning tree of
* This tree can now be used to approximate as
* , where is root of and is parent of in
* Compute each probability in the above expansion of from co-occurrence frequency in training data; smoothing may be necessary if have limited training data
* Model is now trained and can now calculate

Representing Location

* Inputs
* Detector error model (two scalar values): and (can be calibrated for the sensor) , where is the event that the sensor reports that the word exists and is the event that the word actually exists
* Prior probability that a new observation comes from a new place (used for computing equation 11 below)
* Represent world at time as collection of discrete and disjoint locations . Each location is described by the set , an estimate of the probability that each word exists at the location – goal is to compute for each location and select the location with the maximum likelihood



**References**

1. Shen, S., Michael, N., and Kumar, V. (2013) Obtaining liftoff Indoors: Autonomous Navigation in Confined Indoor Environments. *IEEE Robotics and Automation Magazine*, 20(4):40-48.
2. Iterative Closest Point Algorithm – overview. <http://en.wikipedia.org/wiki/Iterative_closest_point>
3. Kumar, V. (2014) Phase 2 MEAM 620 Project notes slides : [https://alliance.seas.upenn.edu/~meam620/wiki/index.php?n=Main.Projects?action=download HYPERLINK "https://alliance.seas.upenn.edu/~meam620/wiki/index.php?n=Main.Projects?action=download&upname=2014\_proj1\_phase2\_slides.pdf"& HYPERLINK "https://alliance.seas.upenn.edu/~meam620/wiki/index.php?n=Main.Projects?action=download&upname=2014\_proj1\_phase2\_slides.pdf"upname=2014\_proj1\_phase2\_slides.pdf](https://alliance.seas.upenn.edu/~meam620/wiki/index.php?n=Main.Projects?action=download&upname=2014_proj1_phase2_slides.pdf)
4. Cherverikov, D. (1991) Fast neighborhood search in planar point sets. *Pattern Recognition Letters,* 12(7):409-412.
5. Rusinkiewicz, S. and Levoy, M. (2001) Efficient Variants of the ICP Algorithm, *Proceedings of the Third International Conference on 3D Digital Imaging and Modeling.* <http://www.cs.princeton.edu/~smr/papers/fasticp/>
6. Grzona, S., Grisetti, G. and Burgard, W. (2012) A fully autonomous indoor quadrotor. *IEEE Transactions on Robotics*, 28(1): 90-100. [http://ieeexplore.ieee.org/xpls/abs\_all.jsp?arnumber=6004839 HYPERLINK "http://ieeexplore.ieee.org/xpls/abs\_all.jsp?arnumber=6004839&tag=1"& HYPERLINK "http://ieeexplore.ieee.org/xpls/abs\_all.jsp?arnumber=6004839&tag=1"tag=1](http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6004839&tag=1)
7. Siegwart, R., Nourbakhsh, I. R., and Scaramuzza, D. (2011) Introduction to Autonomous Mobile Robots. Cambridge, MA; The MIT Press.
8. Cummins, M.; Newman, P., "Probabilistic Appearance Based Navigation and Loop Closing," *Robotics and Automation, 2007 IEEE International Conference on* , vol., no., pp.2042,2048, 10-14 April 2007.
9. Bay, H., Ess, A., Tuytelaars. T., and Van Gool, L. (2008) Speeded-up Robust Features (SURF). *Computer vision and Image Understanding*, 110: 346-359. <http://glorfindel.mavrinac.com/~aaron/school/pdf/bay06_surf.pdf>
10. Evans, C. (2014) OpenSURF. Downloaded Feb 12. 2014. [*http://*www.chrisevansdev.com/computer-vision-**opensurf**.html](http://www.chrisevansdev.com/computer-vision-opensurf.html) *‎*
11. ETH SURF Website (2014) <http://www.vision.ee.ethz.ch/~surf/index.html>.
12. GitHub OpenSURF (2014) <https://github.com/amarburg/opencv-ffi-ext/tree/master/ext/opensurf>
13. MORF.LV (2014) <http://morf.lv/modules.php?name=tutorials&lasit=2>
14. OpenCV SURF (2014) <http://docs.opencv.org/trunk/doc/py_tutorials/py_feature2d/py_surf_intro/py_surf_intro.html#surf-in-opencv>
15. Russell, S. and Norvig, P. (2010) *Artificial Intelligence: A Modern Approach*, Third Edition. Upper Saddle River: Prentice Hall.
16. K-Dimensional Trees (2014) <http://en.wikipedia.org/wiki/K-d_tree>
17. Shen, S., Michael, N, and Kumar. V. (2011) Autonomous Multi-Floor Indoor Naviationg with a Computationally constrained MAV. *Proceedings of the IEEE International Conference on Robotics and Automation*, pp. 20-25.
18. Grzonka, S., Grisetti, G. and Burgard, W. (2009) Towards a navigation system for autonomous indoor flying. *Proceedings of the IEEE International Conference on Robotics and Automation*, pp. 2878-2883.

**APPENDIX: SURF Details**

Specifics:

-Finding Detectors: employs a Hessian matrix-based measure (<http://en.wikipedia.org/wiki/Hessian_matrix>). This is big because it uses integral images (also known as a summed area table) to represent the pictures they take, which allows for extremely fast computations over large areas of the image (using a box type convolution filter) for interest point detection. Very basically, this allows SURF to calculate pixel intensities over large areas using just 3 sums (after the initial integral image has been computed).

-Once the Hessian matrix is calculated, its determinant for various coordinates on the image can be approximated, and this will give a blob response, and for multiple locations, a blob map can be constructed. From what I understand, blobs are interesting features…

-They then sample the image with differently sized filters such as a Difference of Gaussians (DoG) filter to approx. the Laplacian of Gaussians (LoG). This brings speed at low costs to accuracy. It sounds like each iteration you further divide the field into more filtering boxes by at least a factor of 2.

If anyone has taken image processing, please share why this might be useful. It seems like it has to do with scale indifference for feature recognition.

-Descriptor: Once the interest points/ detectors are detected, a descriptor is found within the interest point regions by finding a sum of haar wavelet responses. This:

“We build on the distribution of first order haar wavelet responses in x and y direction rather than the gradient, exploit integral images for speed, and use only 64 dimensions (oh, only 64, good). This reduces the time for feature computation and matching, and has proven to simultaneously increase the robustness. Furthermore, we present a new indexing step based on the sign of the Laplacian, which increases not only the robustness of the descriptor, but also the matching speed (by a factor of two in the best case).”

* Firstly, Haar wavelet responses (<http://en.wikipedia.org/wiki/Haar_wavelet> go to the properties section…it has some relevance) are used to factor out image rotation.
* Note: for some applications, rotation invariance isn’t necessary.
* Then, “For the extraction of the descriptor, the first step consists of constructing a square region centered around the interest point and oriented along the orientation selected from the Haar wavelet responses.
* That square region is then split into 4x4 squares, which is then again split into 5x5. Taking the Haar wavelet responses to EACH of these, results in a descriptor vector of length 64. This achieves illumination bias invariance, while reducing the yielded vector to a unit vector further achieves contrast invariance.

-Matching: Indexing during the matching stage is sped up by including the sign of the Laplacian (the trace of the original Hessian Matrix) for each interest point. This is a free gain, because that sign can instantly rule out a mismatching of interest points that might otherwise appeared to match. Otherwise, matching is generally done by calculating distance between descriptor vectors on two different images relative to position (that came from the older paper, this one: <http://www.vision.ee.ethz.ch/~surf/eccv06.pdf> )

The SURF papers were pretty tough to swallow. I did the best I could to reduce the information into something feasible to understand, but I also cut out some formulas they mentioned that I didn’t understand. It appears you can get SURF binaries here: [http://www.vision.ee.ethz.ch/~surf/index.html](http://www.vision.ee.ethz.ch/~surf/index.html" \t "_blank)

Looks like OpenCV has implemented SURF (+other resources): [http://docs.opencv.org/trunk/doc/py\_tutorials/py\_feature2d/py\_surf\_intro/py\_surf\_intro.html](http://docs.opencv.org/trunk/doc/py_tutorials/py_feature2d/py_surf_intro/py_surf_intro.html#additional-resources)

<https://code.google.com/p/opensurf1/source/browse/#svn%2Fbranches%2FFastHessianDev%2FOpenSURF>

<http://morf.lv/modules.php?name=tutorials&lasit=2>