

Exploration of Design Approaches in Stereoendoscopy

Christopher Liu^{1,*}

¹College of Optical Sciences, University of Arizona, 1630 E. University Blvd., Tucson, AZ 85721, USA

*cliu@email.arizona.edu

Abstract

Laparoscopy has been a vital tool to enable minimally-invasive abdominal surgery. While it is desirable to integrate high-quality optics and surgical instruments into a single endoscope tube, the achievable diameter of the tube is limited by the need to minimize incision length as well as by manufacturing constraints. This imposes exacting requirements on the image-forming and illumination optics. It is especially important that the surgeon be able to see a clear image with a wide field of view in order for the endoscope to be useful. It is also desirable for the surgeon to be able to see stereoscopically, though this is not without its tradeoffs. Early stereoendoscope designs replicated the binocular layout of the human visual system by using a pair of laterally-displaced cameras. Although this provides basic stereoscopy, its field of view is generally limited in comparison to single-view cameras. Therefore, novel methods to achieve stereoscopy have been proposed. Many proposals divide an aperture in two halves based on a property such as wavelength or polarization. However, such approaches often encounter difficulties such as illumination efficiency and confounding effects with the spectral or polarization properties of the tissue. Therefore, one desires a method by which one can capture depth information over a wide field of view without degrading the quality of visible imaging over the same field of view. One promising method is time-of-flight sensing, which uses the speed-of-light definition to determine distance from the sensor to the target. Another potential method is to use more than two cameras to cover the desired wide field of view, which then allows conventional image-stitching and stereo-calibration algorithms to be applied to derive stereoscopic information. This report will explore the theoretical and practical justification for both methods as well as their advantages and disadvantages. It will also cover its author's preliminary study regarding the time-of-flight and multi-camera sensing approaches.

Background: Requirements for Laparoscopy

Endoscopic imaging produces demanding requirements on many aspects of image quality as well as significant physical constraints on the design of the imaging system. Resolution, distortion, and field of view are all important, but the specifications one normally imposes on visual or astronomical optics are impractical to satisfy in a system that is confined to the endoscope tube and uses few lens elements. The main concern is that the small length scale complicates the selection of individual lens-element profiles as well as the arrangement of elements. As a result, it is difficult to minimize distortion, especially in conjunction with the other criteria. To be useful for further processing, the image from each individual camera must be undistorted in accord with a calibration process. This means that the field of view achievable from a single camera is especially limited. A common method to achieve stereoscopy involves a pair of cameras that look toward the scene on axes parallel to each other and to the mechanical axis of the endoscope tube. The field of view

achievable by this method is relatively limited compared to that achievable by a single camera. Theoretically, a single camera could make use of a wide-angle or fish-eye lens to widen the field of view, but the necessary undistortion procedure would pose difficulty. The calibration must be very accurate in order to provide sufficient quality for multiple images to be aligned. Also, it is desirable for alignment to have the best resolution possible, the most uniform resolution possible across the field, and as much uniformity in image brightness and other features as possible. These properties are difficult to achieve in a miniaturized camera of endoscopic dimensions, so postprocessing of the captured images for undistortion and sharpening is generally required. Resolution in particular is harmed by image interpolation; the stronger the camera distortion, the greater the factor by which some pixels must be up- or downsampled. Therefore, endoscopic cameras are conventionally designed to provide low distortion, potentially sacrificing field of view.

The principle of stereoscopic vision in endoscopy is necessarily based on the operation of stereoscopy in the human visual system. The viewer's two eyes converge such that their optical axes intersect, and they accommodate such that they are optimally focused at said point of intersection. The difference in object position on the left and right retinas is known as disparity. For the human brain to interpret such lateral disparity as depth information, the features must fall upon a small portion of the fovea, known as the Panum region. The range of distances over which depth information can successfully be perceived is limited by the size of the Panum region. The human visual system varies the region of depth sensitivity by changing the angle of convergence. This effect can be replicated with a binocular display system but not with a display that uses a single planar reproduction screen. In monocular endoscopy, the entire scene is perceived to be located at a depth corresponding to the focal distance of the endoscope optics, and the field of view is limited to that from a single camera.^[11]

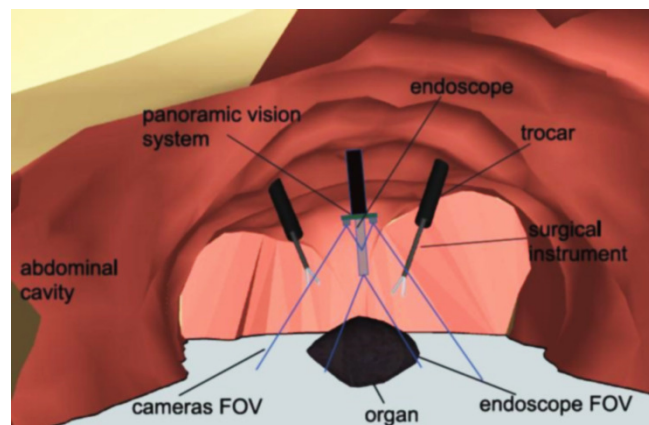


Fig. 1. Schematic of laparoscopic abdominal surgery setup.^[9]

In addition to considerations of optical performance, there are practical requirements on the design of the laparoscope. The system should require as few incisions into the patient as possible. The diameter of each tube should be as small as possible. The system, especially its illumination components, must not consume excessive power, as thermal dissipation could injure the bodily tissue or damage the endoscope itself. Surgical instruments integrated with electrical or optical parts must be reasonably small in terms of tube cross-sectional area. Although some commercial endoscopes integrate

imaging optics, illumination, and endo-therapy instruments into the same tube, this is not necessarily practical for all situations, especially where novel optical configurations and features are desired. Commonly, the cameras are placed in one tube and the instruments in another. This poses a challenge for the practical operation of the laparoscopic system, especially where stereoscopy is desired, as the spatial separation between the camera and instruments requires a wide field of view.

A rigid endoscope tube (or at least tip) is currently preferred to a flexible tube in terms of optical quality. One reason is that a flexible tube typically requires a fiberscope design, which limits the effective resolution (pixel count) to the number of fibers in the bundle. Another reason is that a rigid tube allows for constraint of multiple optical elements, as needed for high optical performance; by comparison, fiberscopes typically have a single imaging lens at the tip in front of the fibers. This consideration is mentioned by R. Korniski et al. in "3D imaging with a single-aperture 3-mm objective lens: concept, fabrication and test." (The system specifically proposed in this paper will be discussed later.^[5])

Comparison of 2D and 3D Endoscope Field of View

Existing 2D endoscopes typically achieve a field of view about twice that of 3D endoscopes, as discussed in the paper "Field of View Comparison Between Two-Dimensional and Three-Dimensional Endoscopy" by J. Van Gompel et al. The paper compares two examples of existing commercial endoscopes, using a Karl Storz 0° endoscope as the example 2D system and a Visionsense, Ltd. 3.3 mm, 0° rigid endoscope as the 3D system. The authors found a 52% reduction in field of view from the 2D system to the 3D system, from 8.3 cm visible diameter in 2D to 4.0 cm in 3D.^[10]

The measurements were based on a working distance of 6 cm from a standardized bulls-eye target as shown in Fig. 2. An additional demonstration is as given in Fig. 3.

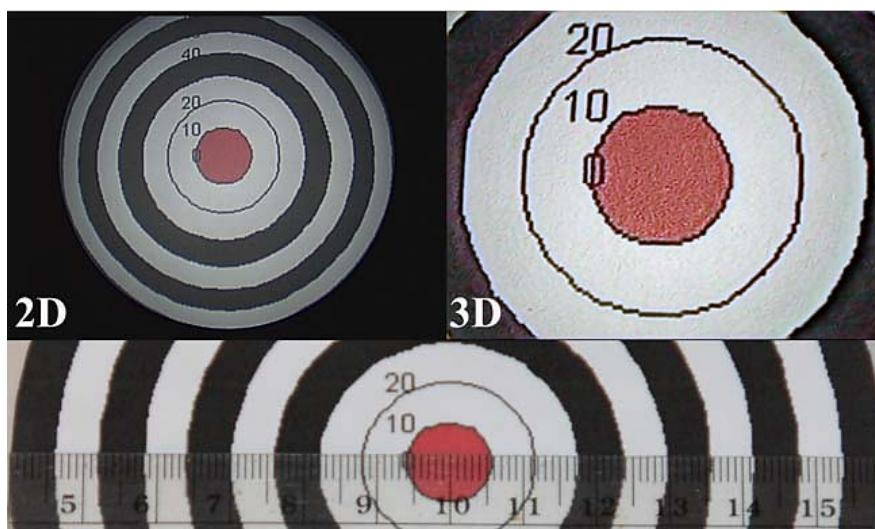


Fig. 2. Field-of-view demonstration for 2D and 3D endoscopes, from Fig. 1 in^[10].

The systems were also clinically tested for use in sinus surgery, where a 55% reduction in FOV was seen in simultaneous use of the two endoscopes.^[10] The authors propose that the reduction in field of view resulting from the inclusion of

stereoscopy is due to limited interpupillary distance and limited aperture diameter. In particular, current 2D endoscopes typically use the full 4 mm aperture diameter allowed by the tube, while the 3D endoscope tube tested here provides an aperture diameter of 3.3 mm despite using a tube of 4 mm diameter. The authors believe that the primary benefit of 3D endoscopy is that it makes it more efficient for the surgeon to learn and carry out a procedure. They also mention that the higher magnification inherent in existing 3D endoscopes, corresponding to a working distance of 3-4 cm, may be an advantage since it reduces the chance that surgical instruments will interfere with the view. Finally, the authors suggest that further work should generally be done to establish a practice for the surgical use of 3D endoscopy.

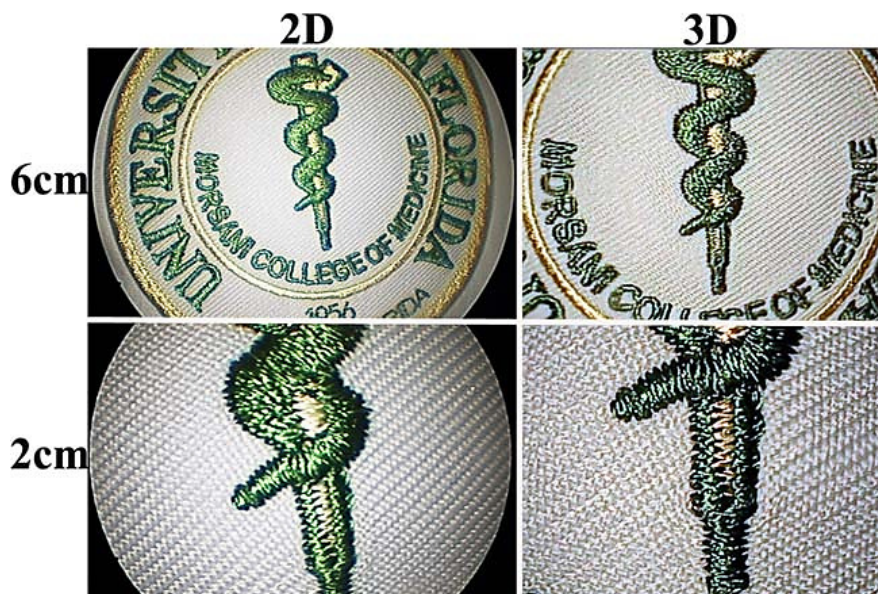


Fig. 3. Field-of-view demonstration for 2D and 3D endoscopes, from Fig. 2 in ^[10].

The authors attribute the difference primarily to the limited interpupillary distance and aperture diameter imposed by a practical endoscope tube.

This paper's conclusions about sinus surgery must be interpreted with caution when considering abdominal surgery; the latter is fundamentally different in that it requires laparotomy and a greater angular separation between the endoscope and instrument tubes.

Effect of Stereoscopy on Surgical Speed and Reliability

The conventional approach to stereoscopy involves the use of two cameras to provide separate views to each eye of the operator. In a typical endoscope, the cameras are laterally displaced; although some systems tilt the camera optical axes relative to the tube mechanical axes, such an arrangement is difficult to provide without additional optical parts. One implementation of two-camera stereoscopy is evaluated in "Autostereoscopic three-dimensional viewer evaluation through comparison with conventional interfaces in laparoscopic surgery" by Silvestri et al.^[7] The authors acknowledge that the "image quality, detail, and color sharpness" provided by existing 2D endoscopes are often compromised in the

implementation of 3D, but a surgeon not accustomed to laparoscopy must exert greater relearning effort to adapt to 2D than 3D laparoscopic surgery. The gold standard for 3D visualization of the surgical field is the da Vinci robotic system, which uses a binocular-visor display to show the image captured by two 4mm-aperture cameras in a tube of 12mm diameter. A binocular-visor display consists of two separate video displays, one for each eye of the viewer. The most common alternative is an autostereoscopic monitor, which has not yet found significant medical applications. Although autostereoscopic displays can be viewed by multiple people, other technologies such as polarized glasses currently offer superior performance for the surgeon.

The camera system built and tested by these authors consists of two VGA-resolution CMOS color sensors, measuring 8 mm x 8 mm x 9 mm. Each sensor has a corresponding pinhole lens, and the pupil centers are separated by 8.7 mm. Each system (2D, 3D with autostereoscopic monitor, and 3D with binocular visor) was tested with surgeons who had no previous experience with 3D endoscopy, performing the following tasks using laparoscopic tools (Fig. 4): a visual task, counting the squares in a checkerboard target; a pick-and-place task, placing rings over pins; a peg-in-hole task, inserting needles into holes; a cutting task, cutting a rubber surface between pre-marked lines; and a suturing task, performing a single suture on a tissue-simulator surface. Each task was evaluated objectively on execution time (except for the visual task) and number of errors (except for the suturing task). The procedures were also evaluated subjectively via a questionnaire.

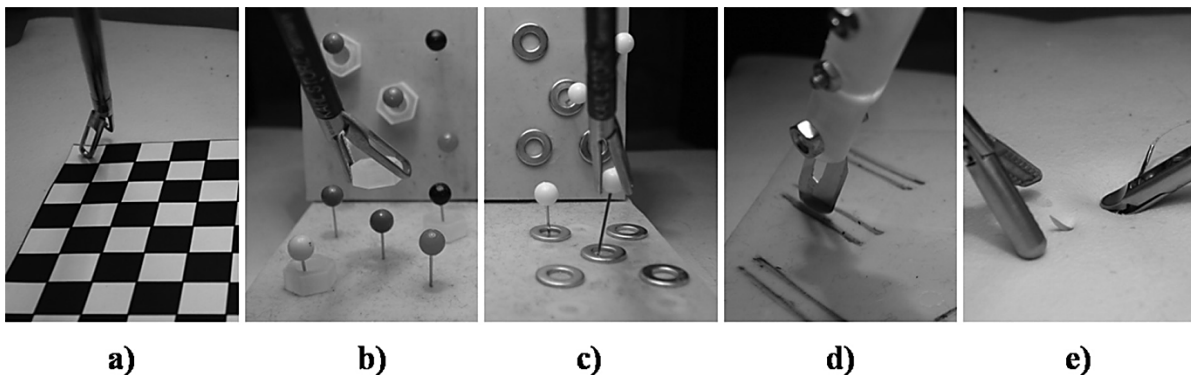


Fig. 4. Photographs of test tasks, from Fig. 4 in ^[7]

Statistical testing consisted of a one-way ANOVA followed by post-hoc analysis of individual pairwise relationships. No surgeons made any errors on the visual task or the pick-and-place task, so there was no statistical analysis to perform for the visual task. For the other tasks, the following results were obtained: For the pick-and-place task, the autostereoscopic monitor and the binocular visor produced statistically-significant improvements in execution time over the 2D display, but the two 3D displays did not have statistically-significant differences with each other. For the peg-in-hole task, no statistically-significant differences were obtained for execution time or error rate; this was attributed to the limited resolution of the camera. For the cutting task, the autostereoscopic and 2D systems were significantly better than the binocular-visor system in terms of speed, while no statistically-significant differences were found in error rate. For the suturing task, both 3D systems were significantly faster than the 2D system, while the 3D systems did not have

statistically-significant differences with each other.^[7]

The results from the questionnaire indicated general satisfaction with the depth and relative-motion perception provided by the 3D systems, with the binocular visor being typically preferred to the autostereoscopic monitor due to the fatigue caused by the latter.

Task	Average Execution Time (s)			Overall P	P		
	AM	BV	2D		AM-BV	AM-2D	2D-BV
Pick-and-place	128 ± 30	136 ± 25	144 ± 31	.004	.8197	.0011	.0008
Peg-in-hole	119 ± 59	107 ± 57	114 ± 47	.79	.5019	.8019	.80
Cutting	21 ± 2	27 ± 1	25 ± 3	.034	.0426	.7847	.0469
Suturing	105 ± 30	102 ± 45	124 ± 40	.0099	.4986	.0202	.0059

Abbreviations: AM, autostereoscopic monitor; BV, binocular visor; 2D, 2-dimensional monitor.
^aFor each task, the 3 average execution times and the overall P value, carried out applying the one-way analysis of variance on the 3 groups, are presented. In addition, this table shows the P values obtained with the post hoc analysis for each pair of visualization systems.

Task	Average Number of Errors			Overall P
	AM	BV	2D	
Peg-in-hole	0.50 ± 0.85	0.71 ± 0.87	0.57 ± 0.64	.7784
Cutting	0.40 ± 0.50	0.73 ± 1.03	1.00 ± 0.92	.1676

Abbreviations: AM, autostereoscopic monitor; BV, binocular visor; 2D, 2-dimensional monitor.
^aFor the peg-in-hole and cutting tasks, the 3 average numbers of errors and the overall P value, carried out applying the one-way analysis of variance on the 3 groups, are presented.

Table 1. Quantitative results of the test tasks described above.^[7]

Overall, 3D endoscopy with either type of display produced an improvement in task execution time. The authors suggest that, despite the viewing-angle limitations and associated training demands associated with an autostereoscopic monitor, the performance of autostereoscopic displays has improved over past studies to the point where surgical results would not significantly differ from other display methods. Since autostereoscopic monitors and binocular visors put different stresses on the surgeon (eyestrain for the former and poor posture for the latter), future improvements in AMs may further stimulate their adoption for stereoendoscopic surgery. Implicit in this discussion is the assumption that the display device, rather than the image sensor, is the limiter of final image quality. As mentioned above, this assumption does not always hold. Where it does not, such as with the aforementioned peg-in-hole task, the advantages of 3D endoscopy may be less evident. The remainder of this report will thus focus on image capture.

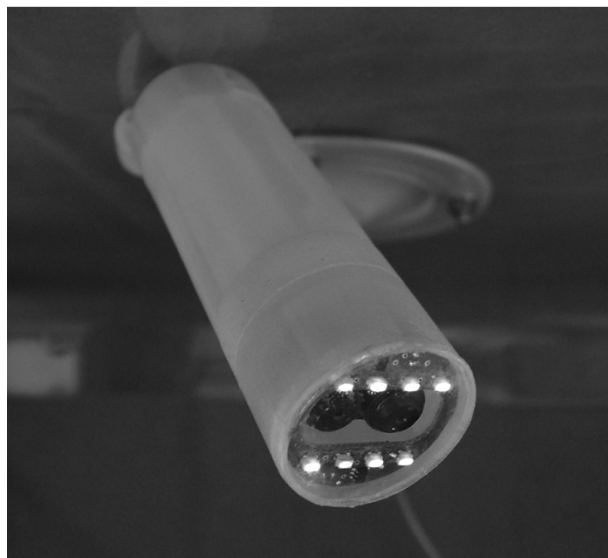


Fig. 5. Endoscope head from Fig. 2 in^[7]

The potential benefits of stereoscopic viewing to the surgeon are also discussed in "Design of the computerized 3D endoscopic imaging system for delicate endoscopic surgery" by C.G.Song et al. This paper mentions the polarization and shutter-glass methods of 3D display operation. It considers the effect of each type of 3D viewing on performance of simulated surgical tasks, finding that the polarized glasses produce a more consistent performance benefit (Table 2).

Mean time as a function of the experience level and mode of visualization (mean[s]±sd)

Mode		2D imaging	Electric shutter-type 3D	Polarization-type 3D
Task				
Task:1 loop passing	Student	205±50	195±35	189±29
	Surgeon	127±37	144±28	135±26
Task:2 suturing	Student	690±107	532±55	512±38
	Surgeon	257±61	230±47	220±53

Table 2. Task completion times from Table 1 in^[8]

The paper also discusses stereoscopic displays (in particular a polarized display and glasses), video multiplexing/demultiplexing, and computerized storage. These are beyond the scope of the present report.

Split-Aperture Image-Capture Methods



Fig. 6. Example of binocular stereoendoscope, from "Insertable stereoscopic 3D surgical imaging device with pan and tilt" by Hu et al.^[4] Although pan-tilt capability may be a desirable workaround for the field-of-view limitations, it involves significant tradeoffs in mechanical complexity and is not frequently implemented in commercial endoscopes. The tube diameter is 15mm, which is the practical limit imposed by the surgical tools used to create and maintain the incision.

Polarization division

Alternatives to the two-camera image-capture approach often involve the division of a single aperture into two halves. One proposed scheme divides the aperture based on polarization, as described by T. Hattori et al. in "Disparity and distortion free stereoscopic fiberscope."^[3] The article describes an endoscope tube design which produces a binocular

image pair compatible with an autostereoscopic monitor. The tip of the tube contains a GRIN lens, followed by an aperture mask in which each half is filled with a linear polarizer. The two polarizers are each at 45 degrees to the split line and 90 degrees to each other. Behind the polarizers lies another GRIN lens and the fiber bundle that carries the image. The imaging light then passes through the fibers to additional focusing optics and a polarizing beamsplitter, which splits the polarization components to two CCD sensors. Around the periphery of the tube is an additional fiber bundle to provide illumination.

The system achieves an equivalent pixel size of 24 μ m in the composited image and satisfies the 30 fps frame rate required for real-time video. The primary limitation is that carrying the image light via fiber bundles significantly limits the achievable resolution. As discussed previously, modern commercial endoscopes deprecate the fiberscope concept in favor of fully-electronic image transmission.^[3]

The authors suggest that future work could focus on simplifying the design of the system via the use of polarization-maintaining fibers in the bundles. It does not appear that this change is sufficiently economical to be worthwhile, as even if fiberscopes were still comparable to the optical state of the art, one would need to overcome the expense of commercially selecting and bundling polarization-maintaining fibers that have consistent polarization-dispersion properties across the visible wavelength range.

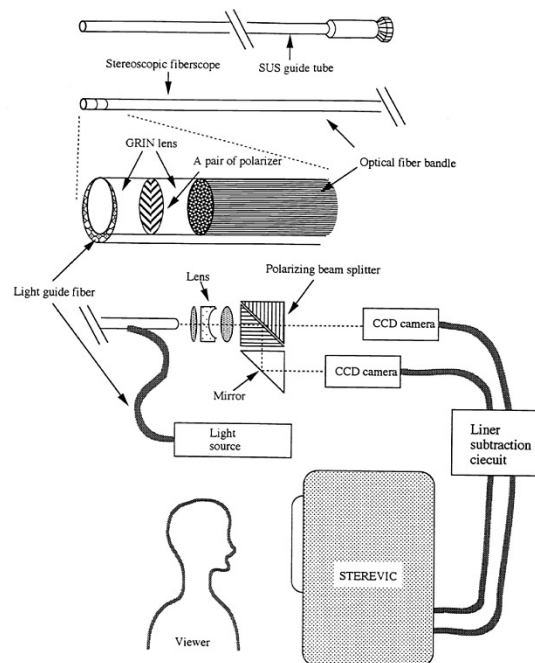


Fig. 7. Schematic of stereoscopic system with polarization-divided aperture. From Fig. 3 in ^[3]

Wavelength division – Complementary multi-bandpass filter

An alternative method to divide a single aperture into two halves is to use color filters such that each filter passes a different set of wavelength bands, each of which is sufficient to reconstruct a full-color image. This complementary multi-bandpass filter (CMBF) method was described by R. Korniski et al. in "3D imaging with a single-aperture 3-mm objective

lens: concept, fabrication and test."^[5] The endoscope design in this paper focused on the minimally-invasive surgery (MIS) subtype of neurosurgery. A 4 mm tube diameter is the existing state of the art for this application, which is a tighter restriction than the aforementioned 12-15 mm for abdominal surgery. The CMBF approach was chosen because of its suitability for the small aperture diameter. The general CMBF principle is that the filter in one half of the aperture passes three wavelength bands corresponding roughly to red, green, and blue, while the other, complementary filter passes a different set of three bands such that the combined transmission covers the entire visible spectrum. A tunable illumination source selects one of the bands for imaging at a time. The initial lab test setup used a xenon lamp with a tunable filter to generate the illumination, with two full-scale laboratory cameras provided with COTS achromatic imaging lenses. Due to the small dimensions required, the mounting hardware required custom rapid prototyping.

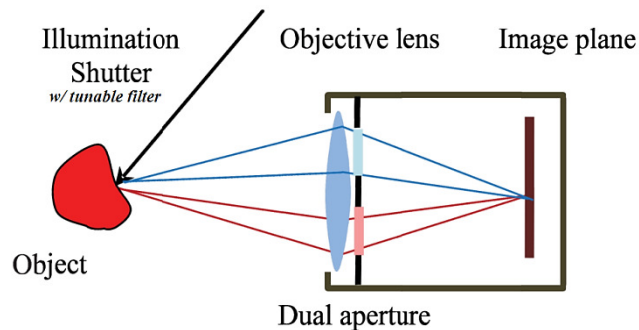


Figure 3: Illustration of CMBF technique

Fig. 8. Schematic of aperture with divided CMBF, from Fig. 3 in ^[5]

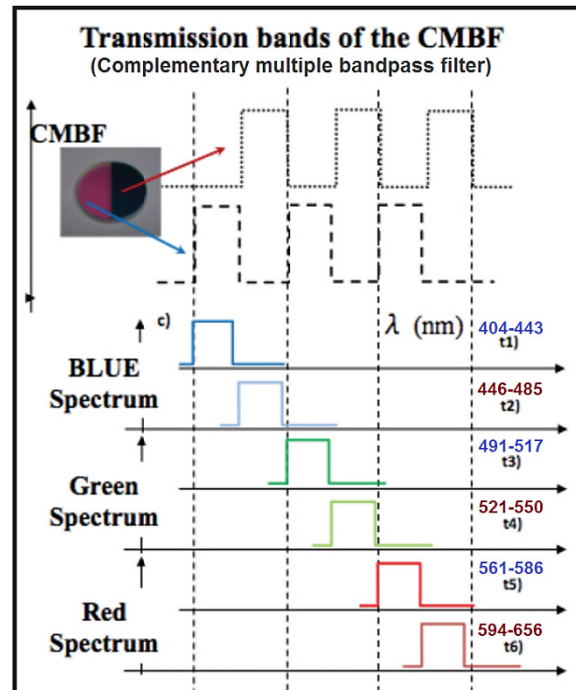


Figure 1: Idealized interdigitated CMBF transmission bands

Fig. 9. Schematic of CMBF band choices, labeled with measured passbands, adapted from Fig. 1 and 2 in ^[5]

After this successful demonstration of the principle, the setup was redesigned to the diameter scale appropriate for the actual endoscope, giving effectively two apertures that were each about 0.8 mm in diameter and with centers separated by 1.2 mm. The designed total field of view was 52 degrees. The frame rate achieved was about 0.5 fps, which the authors recognized as grossly insufficient and in particular far below the standard 30 fps. The authors attributed this to the 10% overall visible-light transmission of the tunable filter, which required correspondingly long exposure times. The effective disparity corresponded to an effective scene distance of 1740 mm (5.7 ft), which was considered by the authors to be within a useful range despite being longer than arm's length. Finally, the authors noted that the different filter bands resulted in some noticeable color differences between the two views. Although this was considered acceptable for a human operator given the fusion processing done by the brain, the authors suggest that postprocessing color correction could be pursued later to further improve color perception.^[5]

The primary obstacle that prevents this system's performance from reaching the commercial state of the art is the illumination issue. Although this would be less severe in an endoscope of greater diameter, such as that possible for abdominal use, the illumination and filtering are still of concern. In principle, the situation could be improved by providing three light sources to allow all three wavelength bands for one eye to be illuminated at a time. However, such an implementation would likely be impractical.

Liquid-crystal deflection

Another approach uses a liquid-crystal device to provide two separate deflected optical paths from the aperture to the image sensor. This method was described by M. Fenske et al. in "A Design of a Liquid Crystal Based Single-Lens Stereo Endoscope."^[1]

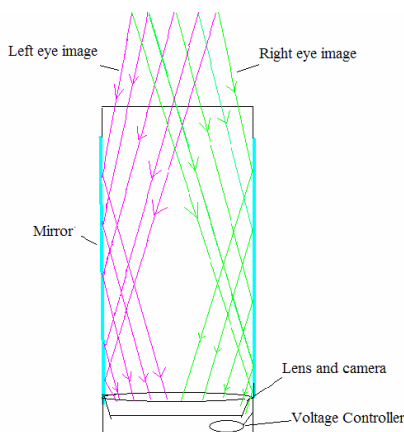


Fig. 10. Schematic of liquid-crystal deflection device, from Fig. 3 in ^[1]

The primary disadvantages of this system are those related to the limitations of liquid-crystal technology. Specifically, polarization properties of the tissue being imaged are possible as a confounding factor, a disadvantage shared with the dual-polarizer method. Also, the switching speed of the liquid crystals may be an issue; although signal frequencies up to

1 kHz successfully produced a detectable LC response, the actual time response of the LC device is necessarily non-ideal and may introduce some crosstalk between the views. Finally, the authors acknowledge that illumination efficiency as well as field of view are of concern due to the limited angle of acceptance of the LC device. They suggest that the former may be tolerable with a sufficiently bright light source, though they do not describe the illumination issue quantitatively nor suggest a specific design change to resolve it.^[1]

Other Sensing Approaches

Shape-from-Polarization

Another use of polarized light is the shape-from-polarization principle, described in "Shape-from-Polarization in Laparoscopy" by Sergio E. Martinez Herrera et al.^[6] The shape-from-polarization principle is proposed as an alternative to shape-from-shading, which attempts to recover depth information from a single monocular image. Shape-from-shading is obviously unsuitable for endoscopy due to nonuniformities in tissue and the possibility of specular reflections. Shape-from-polarization instead operates based on the polarization properties of reflection at a dielectric interface, specifically the water film on the body tissue. By taking images with three states of a linear polarizer (0, 45, and 90°), the surface normal vector relative to the camera at each pixel can be calculated.

The authors' implementation used a mechanically-rotated polarizer.^[6] This is undesirable for the manufacturing and operation of a commercial device, since it would be expensive and impractical to incorporate a suitable motor for electromechanical rotation. Instead, the polarization state should be varied by a liquid-crystal device or other electro-optical means.

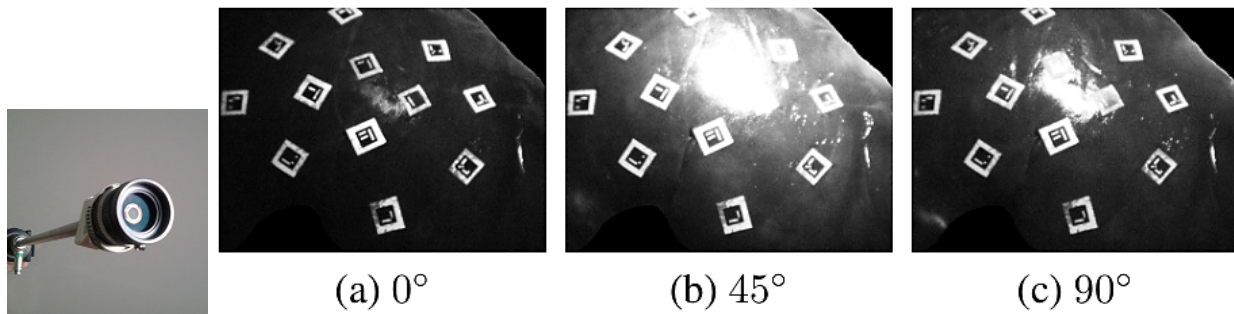


Fig. 11. Camera assembly and example images captured, from Fig. 2 and 4 in^[6]

The authors demonstrate that their method is clearly superior to shape-from-shading, which is mentioned only for comparison and not seriously proposed for practical use. However, the shape-from-polarization method appears not to be sufficiently general to achieve adequate accuracy across many types of endoscopy. It depends on the water film to be of sufficient, consistent thickness. This assumption is not sufficiently general to cover all tissue types, and it fails especially in any area where a large amount of liquid is accumulated. In milder cases, the smoothness of the water film may cause the loss of some depth details present in the underlying tissue.

The present report's author proposes that the preferable use of a liquid-crystal element in an endoscope tube would be to vary the polarization state of a linear polarizer or the focusing power of a lens. Neither of these uses overcome the assumption that the polarization properties of tissue are negligible, which would itself need testing for each organ on which the endoscope is to be applied.

Time-of-Flight Sensing

The stereoscopic methods described thus far have used multiple visible-light images. An alternative is to capture simultaneously a monocular visible-light image and out-of-band depth information. One approach is to use a time-of-flight sensor to acquire the depth information. Time-of-flight sensing uses the principle of the speed of light to measure distance, either by timing a pulsed signal or by interferometry of continuous illumination. A paper that describes this is "Development of a real-time image-guided surgery system for stereo-endoscopic sinus surgery" by A. Hattori et al.^[2] As mentioned in the paper, a depth image is aligned to a visible-light image and used to render a 3D model of the surgical field, which is then output to a stereoscopic monitor. The system was tested on phantoms and in actual sinus surgery. The frame rate achieved was 8-10 fps; the authors did not propose specific suggestions for improvement. The authors noted the importance of field of view: For endoscopy in general, "because of the narrow field of view, it is not easy to recognize the internal structures of the nasal cavity ... even if the surgeon can refer to the patient's MRI or CT images."^[2] It was not stated what specific field of view was achieved for a single frame in this experiment. A schematic of the system is as given in Fig. 12.

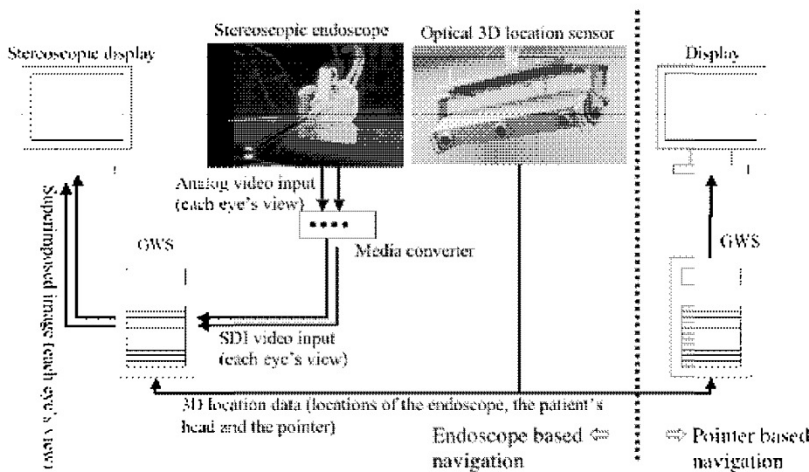


Figure 1. System outline.

Fig. 12. Schematic of endoscope system with time-of-flight depth sensing, from Fig. 1 in^[2]

A comparison of the approaches described above is as given in Table 3.

Description	Pros	Cons	Frame rate	Field of view (° total)
Conventional two-camera stereoscopy ^[7,10]	Proven technique which approximates design of	Limited field of view; wasted tube space	≥30 fps sensor capability; typically	37° (calculated from 4±0.1 cm viewing area)

	human visual system; little computation required for stereoscopic displays		limited by display	at 6 cm working distance given for 3D endoscope in ^[10])
Liquid-crystal deflector ^[1]	Simple lens optics	Limited field of view; inefficient illumination	1 kHz switching speed; usable frame rate uncertain	N/A
Polarization-divided aperture ^[3]	Low distortion, simple optical design	Inefficient illumination; misbehaves if tissue significantly polarizing	30 fps	N/A
Wavelength-divided aperture ^[5]	Low distortion, simple optical design	Inefficient illumination; may require processing to correct color	0.5 fps	52°
Shape-from-polarization ^[6]	Simple optical design with requirements comparable to single-vision endoscope	Electro-optic implementation needs to be designed (to replace mechanical rotation of polarizer); misbehaves with dry or polarizing tissue	N/A (video not possible in experiment due to mechanically-rotated polarizer)	N/A
Time-of-flight sensing ^[2]	Monoscopic visible-light image simplifies design of visible-light optics (to allow low distortion, wide field of view, and/or good illumination efficiency)	Image alignment and 3D rendering sometimes computationally intensive; depth image may have lower resolution than visible image due to sensor limitations; further attention needed to manufacturability in commercial form factor	8-10 fps	N/A

Table 3. Comparison of previously-discussed stereoscopic sensing approaches.

Common Problems in Stereoscopy

As mentioned previously, common problems with the working principles proposed so far for stereoendoscopy include field of view, image resolution, illumination, and optomechanical manufacturing considerations. There are several limiting factors to usable field of view, including lateral displacement, aperture size, and distortion. The more radial distortion present in a single-camera image, the more post-processing is necessary to correct that distortion, causing the corners of the image to lose sharpness relative to the center. (When barrel distortion is corrected, the image content must be resampled to larger dimensions in the corners. This creates a loss of sharpness because the resolution information can derive only from the original image rather than the finer pixel scale of the output. When pincushion distortion is corrected, the image information in the corners is downsampled, throwing away resolution information.) This can be a complication if any image alignment and/or stitching is required; multi-camera stereoscopy and time-of-

flight imaging are both affected. The latter is especially an issue since many existing time-of-flight cameras have lower pixel count than comparable visible-light cameras.

Illumination efficiency is another significant objection to several proposed methods. The light-collecting ability of a system is determined by its aperture size and acceptance angle; the former in particular is often limited by the practical size of the endoscope tube, and any filter in the aperture directly attenuates the light. An ideal polarizer transmits 50% of the irradiance upon it; a color filter may cause arbitrary losses and may require additional computation to recover accurate color information for the displayed image. Any loss of illumination requires a proportionally longer exposure to achieve the same number of photons collected by the sensor. A longer exposure (slower shutter speed) directly slows down the achievable frame rate; generally at least 30 fps is required, and 60 is desirable, for real-time video. Many of the existing approaches reviewed earlier fail this criterion.

As for mechanical considerations, it is obviously desirable to avoid unnecessary moving parts, especially at the tip of the endoscope tube. This is a problem with schemes whose current implementations require a polarizer or other part to be mechanically rotated.

Preliminary Study

Time-of-Flight Sensing

A preliminary study was conducted to explore the possibility of a novel design approach or improvement of an existing design to achieve stereoscopy in an endoscope.

The testing setup for time-of-flight image capture was planned as a proof of concept using commercial off-the-shelf parts. The time-of-flight camera chosen was a PMD Technologies pmd[vision] CamBoard picoS 71.19k (Fig. 13), primarily based on cost considerations. The specifications are as given in Table 4.

The pmd[vision][®] CamBoard pico[®] 71.19k is a reference design especially for the Infineon[®] 3D Image Sensor, which is a joint development of Infineon and pmdtechnologies, based on pmd's technology. It is a functional platform which facilitates the development of specific depth sensing systems and reduces innovation cycles. Form factor and power consumption enable integration into devices, e.g. All-in-One PCs, notebooks or tablets.

Parameter	CamBoard pico [®] 71.19k
Dimensions	89mm x 17mm x 6mm ¹⁾
ToF-Sensor	IRS1010C Infineon [®] 3D Image Sensor IC based on pmd intelligence
Measurement range	20 - 100 cm
Framerate	45 fps (3D frames)
Acquisition time per frame	4.9 ms typ.
Power consumption	1W ²⁾
Illumination	850 nm, LED
Software	C/C++ SDK, Matlab-SDK and gesture detection middleware available
Resolution	160 x 120 (19k) px
Viewing angle (H x V)	82° x 66°
Interface	USB2.0 / USB3.0
Depth resolution ³⁾	< 3 mm @ 50 cm < 6 mm @ 100 cm
Operating System	Win 7 / 8, Ubuntu 12.04, Mac OS X (>=10.9.2)

1) incl. housing
2) typical power consumption: 1W (20fps), 2W (45fps)
3) for lambertian reflection of 75%, 4.9ms acquisition time per frame

Table 4. Specifications for time-of-flight camera used in preliminary study.



Fig. 13. Photo of time-of-flight camera unit. Illumination port on left; detection lens on right.

The testing setup used gauge blocks adhered to a cardboard surface to provide direct control of the depth relationships in the scene (Fig. 14).



Fig. 14. Example setup with ToF camera viewing gauge blocks.

The initial goal was to achieve an absolute depth resolution of 0.5 mm or better under typical laboratory conditions of working distance. The specifications indicated that the chosen sensor had a reasonable chance to be capable of this: The indicated depth resolution is "<3 mm @ 50 cm, <6 mm @ 100 cm" implying a percentage error of 0.6%. At the typical laboratory working distance of 15 cm, this would imply a depth error of 0.9 mm. However, practical results indicated difficulty achieving any consistent depth resolution over the full field. The sensor was able to distinguish gauge blocks of 0.5 mm difference in thickness, both through air and through standard uncoated glass lenses, but in light of the errors to be mentioned, it is unclear how much real depth resolution was exhibited and how much the effect can be attributed to variations in surface finish and cleanliness. Also, specular reflections off the lenses as well as the gauge blocks themselves produced spurious depth information (Fig. 15-18).

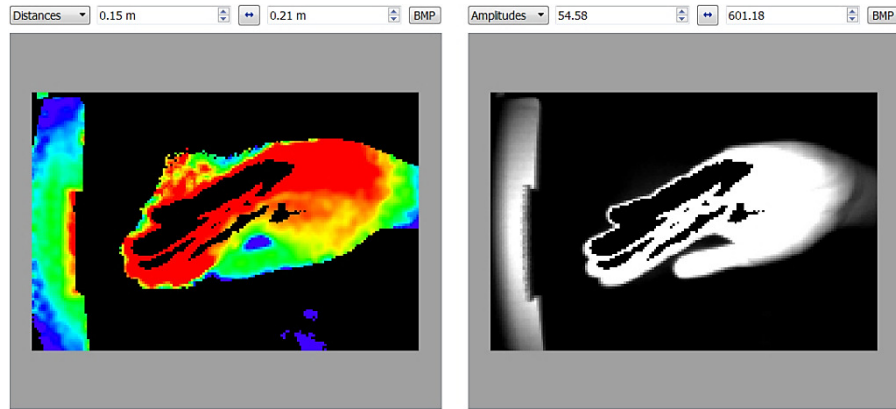


Fig. 15. Example time-of-flight image showing data dropouts due to sensor saturation. Distances on left (color scale uses red for near to violet for far); conventional IR image on right. Amplitude units are arbitrarily assigned by the sensor. This convention applies to the other time-of-flight images shown below, except where noted.

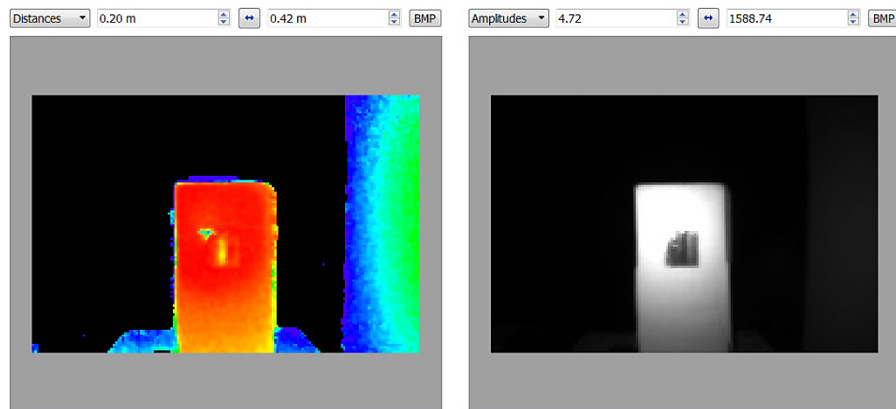


Fig. 16. Example time-of-flight image showing confusion of depth relationships due to illumination variation. Although the gauge blocks were set up in a progression from left to right, the center one is claimed by the sensor to be at a noticeably farther depth than the others. Also note the dimple caused by the illumination hot spot at the upper left.

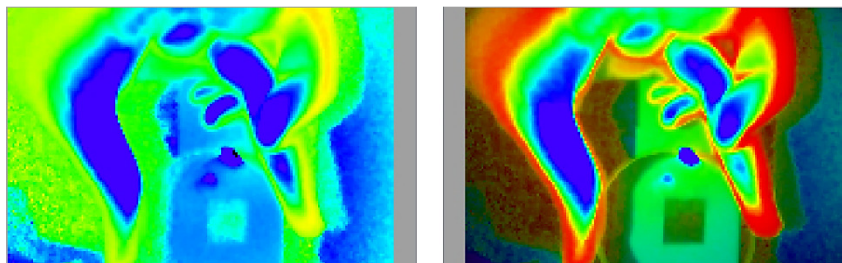


Fig. 17. Example time-of-flight image showing ability to see through a lens. Note the reversed depth relationships on the hands holding the lens and the hot spot on the lens itself. Right image consists of calculated distances (hue) overlaid on the standard IR image (brightness).

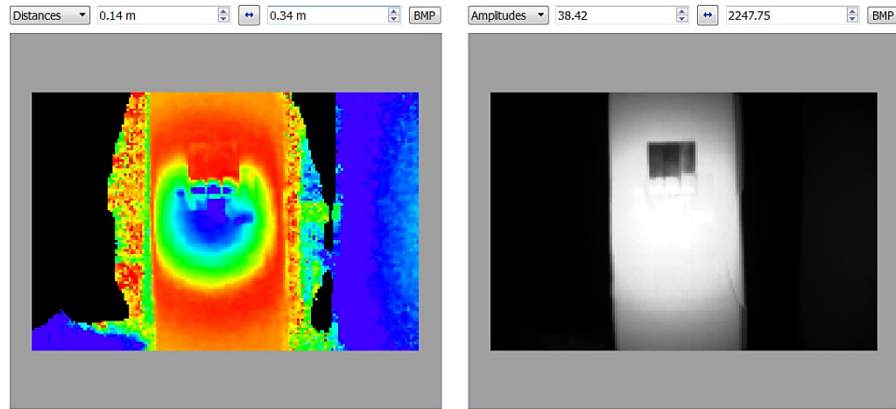


Fig. 18. False depression in depth readings created by overillumination.

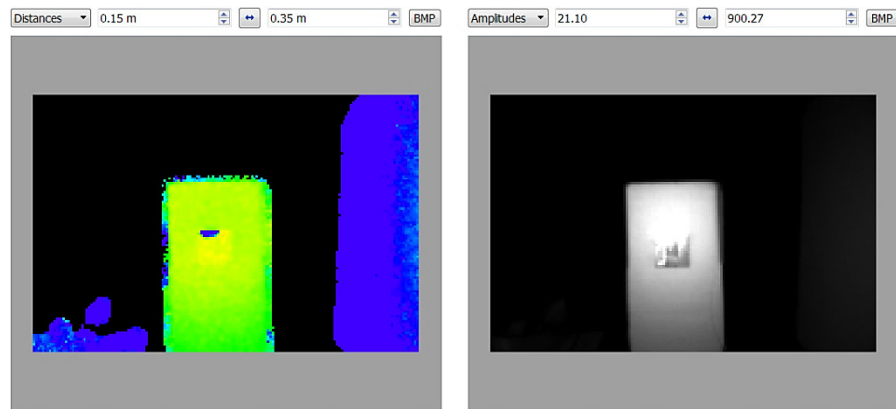


Fig. 19. Result with plastic holographic diffuser rated for 80° diffusion angle applied to illumination port. Note that there is still an erroneous depression near the top of the gauge blocks.

The results were unsatisfactory due primarily to nonuniformity of the infrared illumination. In some areas that were overilluminated but not so much as to saturate the sensor, the depth relationships appeared reversed. The working distances used were typically around 15 cm, which was somewhat shorter than the minimum 20 cm recommended in the spec sheet but still a very long distance on endoscopic scales. Attempts to attenuate the illumination with neutral-density filters or to apply diffusers had little effect (Fig. 19).

Due to the difficulties that were encountered with this method, no registration to a visible image was attempted. However, such a process could have been accomplished by detecting edges in the visible and ToF images and using the edge information to register the images by any common method. If the visible camera were of sufficiently low distortion and otherwise good optical quality, a simple perspective transformation (homography) could satisfactorily align the images. Otherwise, an optical-flow method (as discussed later) could be used to account for arbitrary nonlinear relative distortions between the ToF and visible images.

Image Stitching of Multiple Views

Overview

It was decided to proceed with an alternative approach, using multiple views from which a ring of images could be stitched.

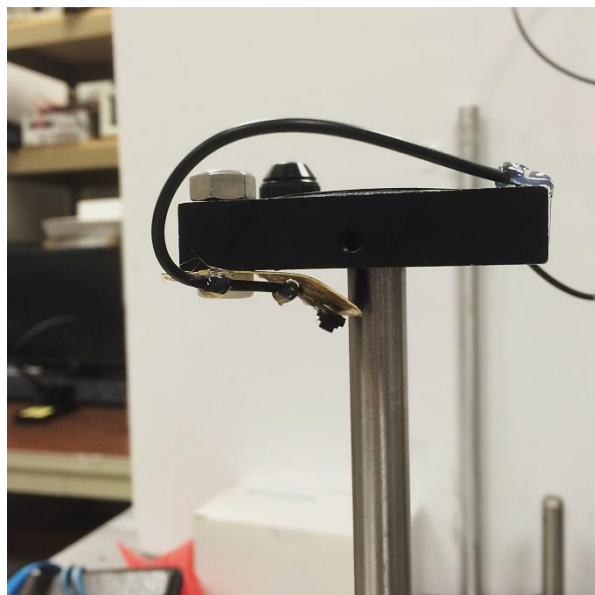
The multiple-camera stitching approach was chosen for the second phase of this preliminary study due to its proven technological basis. On the surface, it is a logical extension of the two-camera stereoscopy approach. The images from the multiple cameras can be laid out so as to directly augment the field of view. Conventionally, an endoscope with this design incorporates a ring of cameras pointing laterally around the tip of the endoscope tube; ideally an additional camera is provided looking axially out from the tip, to close the ring field of view into a hemisphere.

For this application, it is important to remember the limitations of optical quality for visible-light cameras small enough to incorporate into an endoscope tube, due to the lens-design constraints imposed by the small size.

No actual stereoscopy was implemented in this study, due to a multitude of image-quality limitations that will be discussed.

Testing Setup

Due to equipment limitations, only one camera was used, and the multiple views were acquired with the use of a rotation stage (Fig. 20). The camera was as specified in Fig. 21.



ELECTRONICS123.COM, INC.
YOUR ELECTRONIC HOBBY STORE

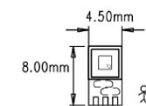
muC303
Micro-Camera Module
(Preliminary spec)

This is a family of products based on the most advance CMOS mixed signal technology. It integrates image array, signal processing, timing and control circuitry, all on a single chip. It is ideal for applications requiring a small footprint, low power and low cost.

Features:

- > Small size : 4.6Wx8Lx4.5T mm (include lens)
- > Resolution: 400x400 pixels
- > Vertical view
- > Lens f1.1mm, F2.8, FOV 120
- > Focus adjustable
- > Operation voltage 3.3V
- > Low power consumption (48mW typ.)
- > Cable size: 1.95mm OD
- > Cable length: 1M

PCB Dimension



Pin Description

1.	VDD	3.3VDC
2.	GND	Ground
3.	CLK	Clock input from backend
4.	VTO	Analog video out
5.	SDA	I2C data
6.	SCL	I2C clock

Specification

Imager	CMOS imager sensor OV6930
Optical Format	1/10.6"
Video Output	Analog
Scan mode	Progressive
Data format	Raw RGB
Picture Element	400x400 pixel
S/N Ratio	38dB
Dynamic range	68dB
Lens info	
Focal length	1.2mm
Aperture	2.8
FOV	120deg
Operation Voltage	3.3VDC
Operation Current	15.4mA max
Connector	6pin cable
Connection	VDD, GND, CLK, VTO, SDA, SCL
Dimension	4.6W x8Lx4.5T mm

Fig. 20 (Left). Camera attached to rotation stage. Note that the camera is oriented such that the image produced is upside-down (rotated 180°) relative to its natural orientation.

Fig. 21 (Right). Specifications for camera.

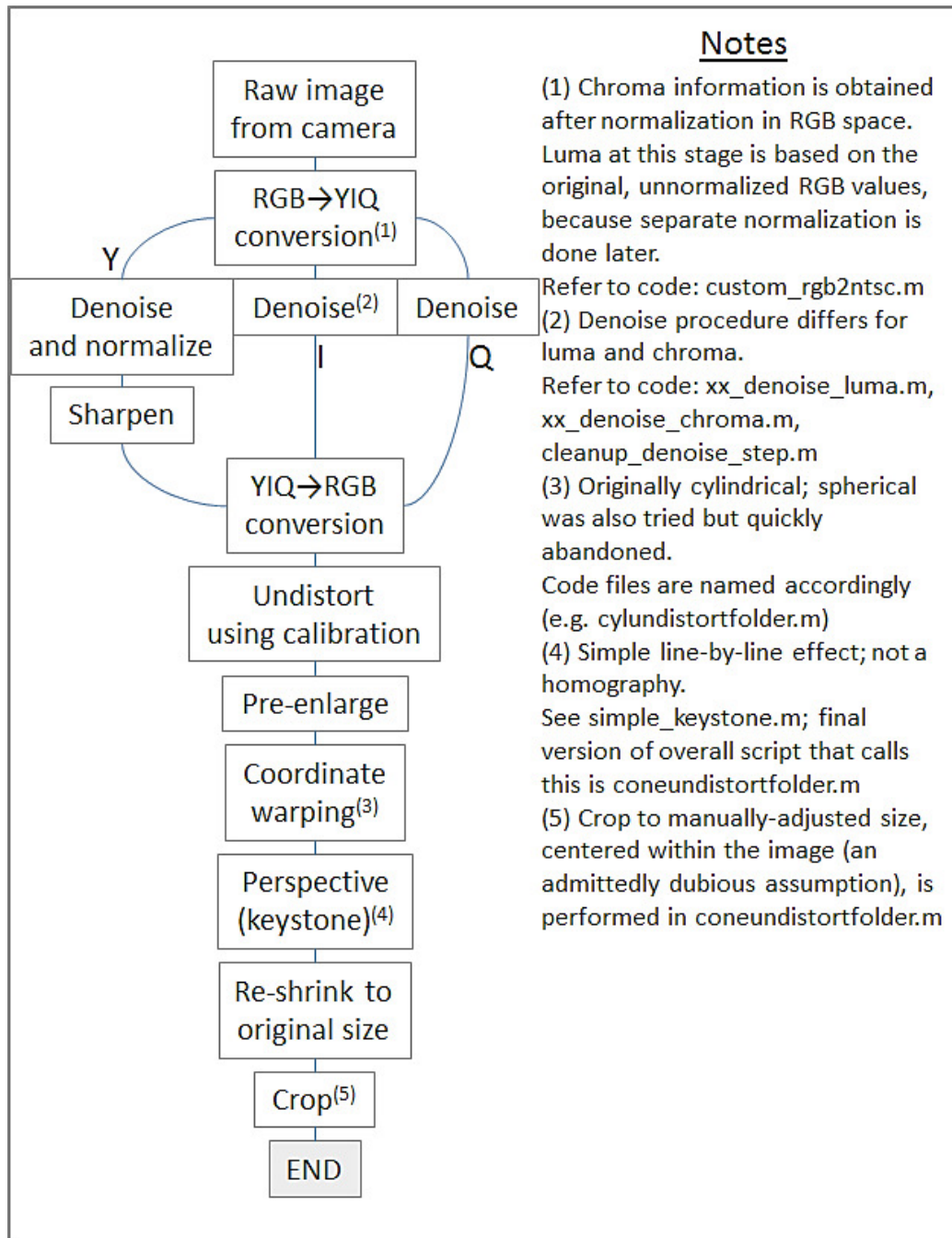


Fig. 22. Block diagram of MATLAB code for preprocessing of each image.

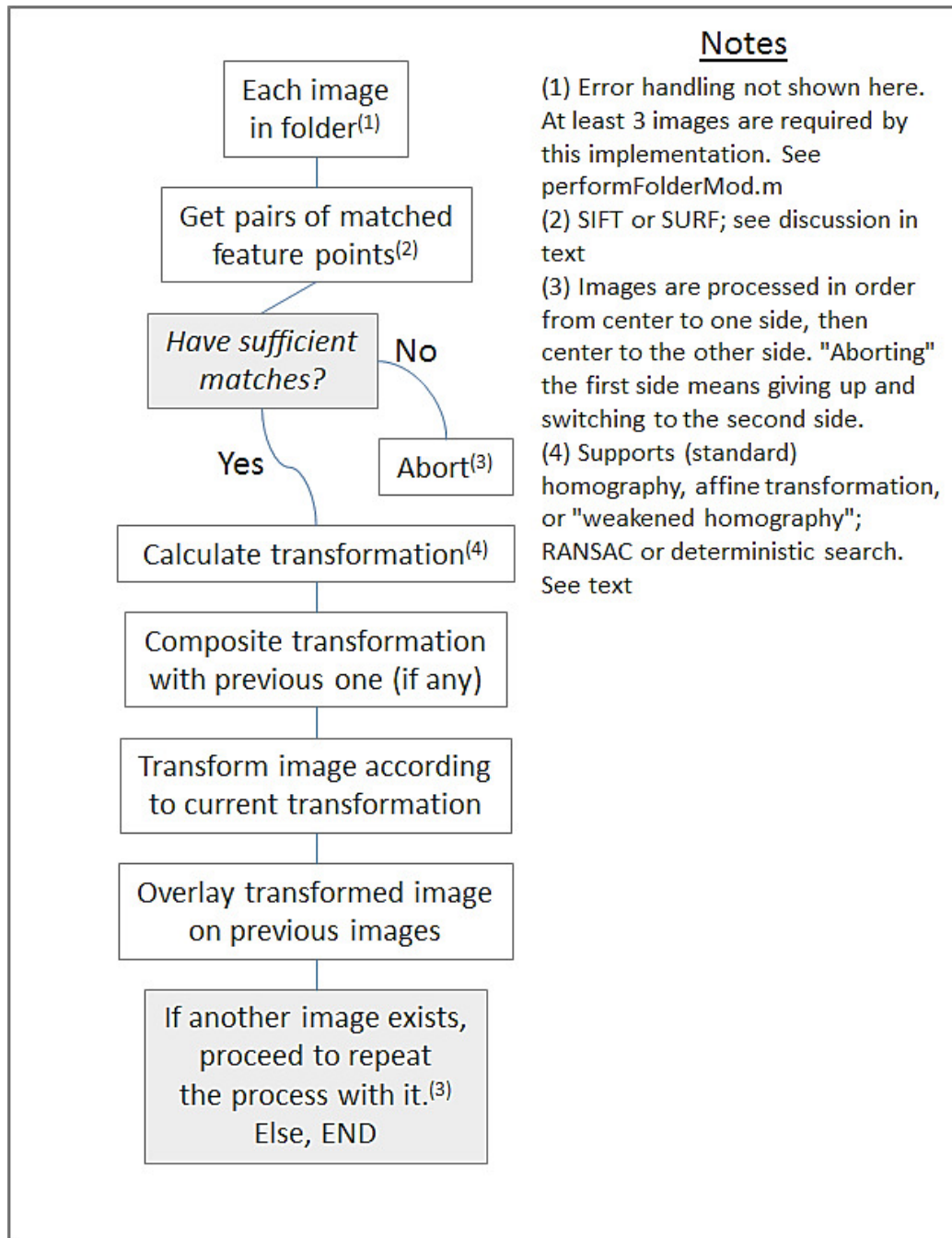


Fig. 23. Block diagram of MATLAB code for feature-detection/linear-transformation stitching approach.

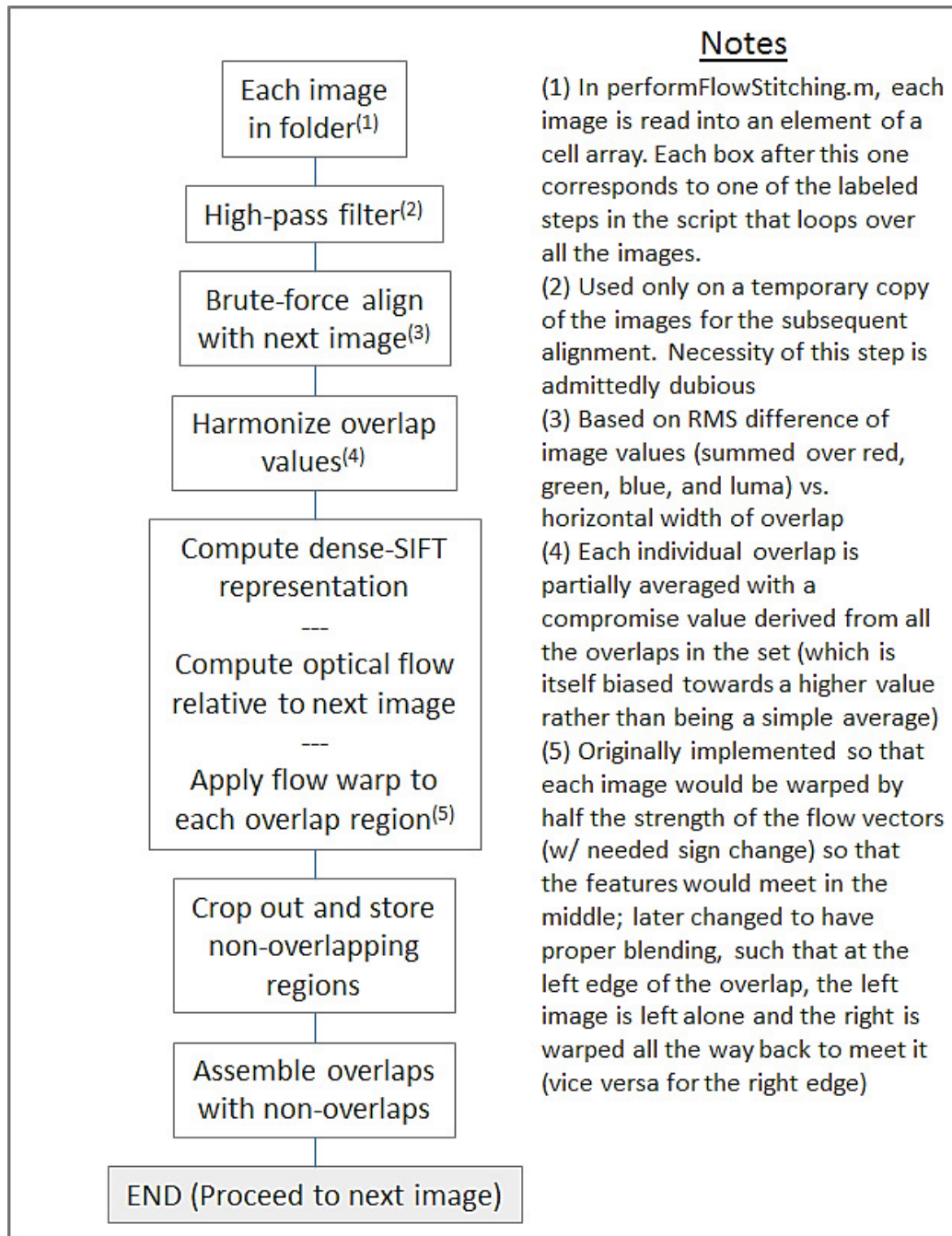


Fig. 24. Block diagram of MATLAB code for optical-flow stitching approach.

The MATLAB code that was written for this project followed the structure shown in Fig. 22-24.

Background on Image Processing

Features in an image may be detected by the SIFT (Scale-Invariant Feature Transform) or SURF (Speeded-Up Robust Features) methods. The former is implemented as an external binary blob, while the latter is built in to MATLAB's

Computer Vision System Toolbox.^{[12][15]} The SIFT method obtains a multi-scale representation of the image information via Gaussian filters, determines the locations of likely feature points based on the maxima and minima of the difference of these Gaussians, and samples additional information (especially the gradient) around each point.^[16] These methods detect blob-like features, that is, regions of a value clearly distinguishable from the surrounding background. This definition of a feature may not be ideal in our context, since it may not accommodate all combinations of errors such as perspective, lighting, noise, and sharpness differences between images. These limitations may thus frustrate the detection and comparison of legitimate features. It may be preferable to detect edges and/or corners, as such features are intuitively less likely to be vulnerable to variations in lighting and are more critical for the quality of a stitched image. This aspect could be approached in future work.

Since the feature detectors were intended to operate on grayscale images, they were run on the luminance as well as the individual RGB channels of each image, and the resulting feature points were pooled together.

RANSAC (Random Sample Consensus) is a method by which the best transformation is chosen to correspond to a set of matched point pairs. It operates by calculating transformations based on a limited number of random subsets of the input points, keeping the transformation that acceptably replicates the largest number of matches. The assumption is that any points which fail to match acceptably in this manner are likely outliers.^[13] The standard implementation uses a hard (go/no-go) threshold to decide whether each point is acceptable; after some experimentation, the present author replaced this with a Gaussian function. In an effort to ensure determinism, the present author also replaced the random search with a full brute-force search of subsets whose size was chosen to allow a reasonable limit on the iteration count.

Preprocessing of Images

The preprocessing script first converts the images from the original RGB to a YIQ color representation. In order to provide some level of color correction, this involves a partial normalization in RGB space. Then, each channel of the YIQ representation is denoised and processed additionally: The luma undergoes denoising, normalization, and sharpening, while the chroma channels are solely denoised. The denoising process was chosen on an ad-hoc basis to consist of several differently-sized kernels of Wiener filters and median filters averaged together. The purpose of the Wiener filters was to attenuate noise in smooth areas of the image, while the median filters were intended to remove speckle noise, both avoiding loss of sharpness on simple edges. The fine-tuning of this process was worked on at the same time as the tuning of the feature detection and matching. Originally the process was iterated until the RMS change in the image values was less than a threshold, but it was soon discovered that no one threshold worked acceptably for all images. Sometimes, certain images in an acquisition set would have their details completely destroyed, while others might not be effectively denoised at all. Therefore, the process was changed to run for a fixed number of iterations. The images are then undistorted using the calibration previously obtained from the MATLAB Single Camera Calibrator widget. Next, the image is upsampled by a factor of two in order to provide anti-aliasing for the subsequent cylindrical-warp step. The cylindrical warp was implemented using naive nearest-neighbor interpolation; if time had allowed, a later reimplementation would have used a proper resampling method in this step in order to obviate the pre-scaling. The

purpose of the cylindrical warp is to provide a coordinate transformation that maps the assumed cylindrical angle of each input image column to the corresponding rectangular coordinates in the output.^[13] Next, perspective correction is done as needed, using a simple keystone effect. The images are then downsampled back to the original size and saved. In this implementation, the files must be manually moved to a chosen subfolder before the actual stitching can be run. This could of course be addressed in future work.

Test of Camera Field of View

After a few preliminary tests of the stitching process, the raw field of view of the camera was tested as a sanity check. This was performed via imaging of a grid target as shown in Fig. 25. The grid was placed close enough to the camera to fill the entire field of view.

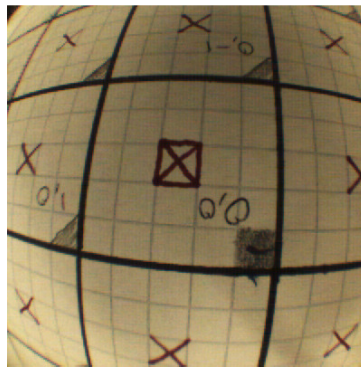


Fig. 25. Image taken during field-of-view test.

In this figure, the camera was measured to be 4.4 cm from the target. The upper-left and lower-right corners of the image were separated by 3.4 in (8.6 cm) in the horizontal direction and 3.6 in (9.1 cm) in the vertical direction as shown. This gives a full-field-of-view angle of 110 degrees, which is reasonably close to the manufacturer's specification of 120 degrees and suggests that the camera's raw field of view should not be a significant limitation on stitching performance. The field of view after undistortion is necessarily less, since the undistortion process introduces a relative pincushion distortion to correct the camera's barrel distortion; this discards image information especially at the corners. This effect was not measured accurately in my work, due in part to multiple trials being made on the camera calibration. The measurement of field of view of a stitched image collection was also explored via the same method, but no reasonable values were obtained, because the image quality was limited and the pitch-angle adjustment of the camera (bending the metal plate shown in Fig. 20) had no numerical ruling. The corner-to-corner field of view is not directly relevant for stitching anyway, since the views are offset horizontally. These issues would need to be explored in future work.

Feature-Detection Stitching: Problems and Workarounds

Upon implementing the stitching, the most commonly-seen failure mode was that one or more images were compressed into a narrow line. This logically results from the least-squares nature of the linear algebra involved, with the possibility

for poorly-conditioned matrices and accompanying stability issues. The offending images were often stretched very long and thin, forcing a large padding area to be allocated in the output canvas. This sometimes caused the script to abort when it detected insufficient memory.

Originally, the code considered the combination of already-stitched images as one of two inputs to the stitching algorithm at each step. There were occasional artifacts in which an image that matched poorly with the ones previously stitched was placed improperly into the black padding space in the upper-left corner. This appeared to be the result of SIFT misdetecting the padding space as a feature. The solution was to keep track of the images by explicitly composing the transformations.

Even when that was addressed, the images in each stitched combination only subtended about a half circle within the stitched view. A contributing factor was that the algorithm was intended to be fully general and was not explicitly given prior knowledge of the images forming a closed circle. Accuracy limitations of the undistortion, cylindrical warping, and perspective correction may also have contributed to this.

Although affine transformations typically had better worst-case behavior than homographies, homographies were generally necessary in order to achieve reasonable accuracy with tricky perspective cases in sets with smaller numbers of images.

To provide the full degrees of freedom of a homography while avoiding obviously unreasonable perspectives, a method of biasing a homography toward an affine transformation was devised. This method, here referred to as a "weakened homography," calculates the best homography and the best affine transformation, transforms some key points according to each, averages the results by point, and computes a new homography based on said points. (Since a homography is not "linear" in the same sense as an affine transformation, the compromise cannot be performed simply by averaging matrix coefficients.) Although this method saw some initial success in improving the results, further testing revealed that it did not solve the fundamental problem that unreasonable transformations could be generated.

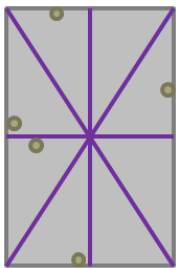


Fig. 26. Schematic of feature points and their bounding box, as used for the weak-homography method and for transformation reasonableness scoring.

Another attempt to work around this issue involved the addition of a reasonableness heuristic to the scoring used to select the best transformation. The sum of the the point distance metrics was multiplied by a value representing the similarity of the prospective transformation to a rigid motion. The chosen method to calculate this was based on the

change in length of lines running corner-to-corner as well as vertically and horizontally through the middle of the bounding rectangle of the input feature points, as shown in Fig. 26. (It was deemed impractical to give the RANSAC or deterministic-search function an understanding of overall image size, since this would require additional parameters to be passed. Therefore the bounding box was based solely on the detected feature points of each image. In retrospect, this may have hampered the effectiveness of this reasonableness check. Similarly, the weakened-homography computation described above should also have accepted parameters representing the bounds of the entire original image, but this would also have complicated the code.)

For affine transformations, such glitches often manifested themselves as the script aborting due to internal NaNs or infinities that were unacceptable in the transformation methods. For pure and weakened homographies, the script would sometimes freeze or abort when the bad transformation exhausted available memory in stitching.

It was at this point that the idea of perspective-correcting the images prior to stitching was implemented. This did give slightly more consistent results, but the problem remained that the images formed neither a closed circle nor a horizontal strip.



Fig. 27. Stitched image set with example of unreasonable transformation.



Fig. 28. Stitched image set with example of misalignment to empty space



Fig. 29. Example of a better-quality, but still far from ideal, stitch of the same image set



Fig. 30. A rough hand-stitched approximation, using only rigid rotations and translations, for the sole purpose of showing the intended arrangement of the images

An implementation decision was made to adjust the perspective of each input image as part of the pre-processing phase, prior to the actual stitching. Recall that the physical arrangement of the camera put the images in an inverted orientation, such that the top of the image corresponded to the bottom of the scene, and that the camera was tilted downward such that the tops of the images overlapped more closely than the bottoms. Therefore, it is the case that the top of each raw image represents a greater angular subtense in the stitched panorama than does the bottom. It was thus decided to expand each image at the top and shrink it at the bottom. To avoid complications with the naively-centered cropping and to minimize interpolation artifacts, it was decided to perform this transformation on a simple line-by-line basis rather than defining it as a true homography.

Optical Flow-based Stitching: Overview of Implementation

For good results, a nonrigid alignment is required for two reasons. First, the scene may contain depth variations that create parallax differences. Second, it may be difficult to correct all distortion in a low-quality camera via the calibration procedure. Note that Image Alignment Toolbox (a third-party MATLAB toolbox) uses the word "rigid" to refer to any transformation type that can be described in terms of linear algebra, as opposed to optical flow.^[14] This is in contrast to the strictly correct sense of a "rigid transformation" referring to simple translations and rotations. The optical-flow methods implemented in MATLAB's Computer Vision System Toolbox are intended primarily to detect subpixel motions; the Lucas-Kanade and Lucas-Kanade/difference-of-Gaussians (LK-DoG) methods are restricted as such. The latter requires more than two frames in order to perform the difference-of-Gaussians competition, so it is unsuitable for this purpose. The Farneback method operates over multiple scales and so is in principle able to detect multi-pixel motions, but it would be difficult to configure to handle the large motions required in this situation. There is a more serious implementation problem: The result of the optical flow is returned in a special object type that has diverse uses in Simulink but can only be plotted from within a standard MATLAB script. Therefore, an alternative implementation of optical flow was required. The primary requirements were that the implementation support arbitrarily large flow distances, require only two input images at a time, and output the flow vectors in ordinary MATLAB arrays. These factors led to the choice of the SIFTflow algorithm implementation from Image Alignment Toolbox.

The chosen approach for optical flow was to stitch the images together as a horizontal strip and then warp this strip into the desired circular form. First, a naive horizontal-translational alignment is performed on each pair of consecutive images. Then, the optical flow is applied to align the features.

The implementation requires generating forward and reverse motion vectors that map the n th image to the $(n+1)$ st and vice versa, respectively. An initial implementation performed simple unweighted averaging of the vectors and of the warped images. This produced obvious sharp boundaries between the overlapping and non-overlapping segments of the image, where the features would not exactly align. This issue was addressed initially by interpolating the flow vectors linearly based on horizontal position within each overlap. Although this maintained basic continuity of position, the angle of long straight features continued to show discontinuity. This was resolved by the use of a sinusoidal function (peak to peak) for the position dependence of interpolation. (A cubic spline would have been preferable in order to avoid discontinuities with higher-order derivatives, but the effect would not likely be noticeable in light of other image-quality issues.) Finally, to reduce the visibility of some artifacts resulting from inaccuracy in the optical flow itself, the interpolation of the image content was weighted in terms of horizontal position.

It would have been preferable for both methods to be combined: a pre-alignment by linearly transforming the images in accord with detected features, followed by a fine non-rigid alignment by optical flow. However, this would require overcoming the problem of non-rectangular images, which are not well-supported by existing feature-detection implementations.

Stitching Problems and Future Work

One major limitation regarding the current testing setup is the limited resolution. The main limitation to resolution in this setup is the camera used to capture the images. The camera is of relatively low pixel count (400x400). The lens quality is limited by the small package size, and the depth of field is accordingly limited. Also, the significant amount of undistortion and denoising processing required further limit the clarity of the image. A better-quality camera would produce inherently sharper images while permitting the image alignment to be tuned more easily. In practice, the camera choice is limited by the form factor imposed by the endoscope tube.

Additionally, the repeatability of positioning is an issue, especially the height of the rotation stage relative to the table, where the post stage provided no built-in ruling for the height axis. In order to avoid the need for external measurement with a ruler or calipers, such a stage should have been provided.

Further limitations to image quality may result from the naive method by which the pre-alignment was performed. The initial assumption was that it would suffice to perform a keystone perspective correction followed by translational alignment in a simple horizontal strip. One problem was that each image needed to be cropped to a rectangle to prevent the feature-detection or optical-flow algorithms from mis-recognizing the padding area as a feature. This means that a satisfactory amount of perspective correction for the horizontal-strip situation would have required a significant amount of image information at the upper left and right corners to be cropped away. The perspective correction was thus hand-tuned primarily to provide a reliable stitch result in the context of the linear-transformation method, but it was not optimal for either that or the optical-flow method. With this in mind, it was far from ideal to use a naive translational alignment and expect the optical flow to absorb any residual errors. With either feature detection/linear transformation alone or optical flow alone, there were often errors and inconsistencies especially in the handling of periodic patterns in the scene. Such patterns resulted from the screw holes on the optical lab table as well as the checkerboard pattern of the calibration target (which we often included in the scene even after the initial calibration was completed). The errors in question generally took one of two forms: In the feature detection/linear transformation method, a given feature point in one image would sometimes be matched to a point in the next image that was one period off from the correct feature, causing an inaccurate perspective to be chosen. In the optical-flow method, the holes in two consecutive images would simply fail to be aligned with each other adequately. Errors may result from noise, interpolation artifacts, and variations in specular-reflection angle between camera positions. These may outweigh the legitimate cues of feature shape and therefore trigger misalignment. Again, the previously-mentioned proposal to combine the two alignment methods could improve the situation.

Regardless of the specific stitching algorithm used, one would not want to run a fully-general stitching procedure on every frame of a video signal obtained from an endoscope. This would be an inefficient use of computing power, since the spatial relationships of the cameras do not change significantly from frame to frame. Instead, the designer should provide a means to stitch a single reference image set as a means to obtain a calibration for the camera positions in the

assembled endoscope. This saved calibration is then used to align the video frames obtained from each camera.

In addition to the issues mentioned above, future work should focus on verifying the distortion calibration of the camera; choosing a better method of pre-alignment, which may include reworking the stitching procedure to combine feature detection/linear transformation with optical flow; choosing a better implementation of optical flow; and optimizing the preprocessing of the images so as to ensure that subsequent algorithms correctly detect the features. Such fundamental problems with the image-stitching procedure would need to be resolved before one could consider the implementation of stereoscopy or the measurement of resolution and MTF.

Conclusions

It is generally beneficial in laparoscopic surgery for the endoscope to provide stereoscopic vision. The benefits include faster and more reliable completion of the surgical task, both while learning the procedure and for experienced surgeons. In addition to this justification for stereoscopy, this report has reviewed several methods by which it can be achieved. These include the simple two-camera approach, polarization division, complementary multiple bandpass filters, liquid-crystal deflection, and time-of-flight sensing. A preliminary study was conducted with time-of-flight sensing chosen as the initial approach, because it appeared to have the least difficulty with illumination efficiency and overall visible-image quality. As it was found that a commercial time-of-flight sensor encountered issues with over-illumination at short working distances, a decision was made to abandon this approach in favor of stitching multiple image views. This approach was chosen because of its proven physical-mathematical basis in augmenting the field of view. Although the fundamental principle was demonstrated to be feasible, this also encountered difficulty, due in part to the choice of stitching algorithms and in part to the low resolution of the camera. Future work should consider other algorithmic variations, tested with a better-quality camera that still meets the size requirements for endoscopy.

References

Used in theoretical-review sections

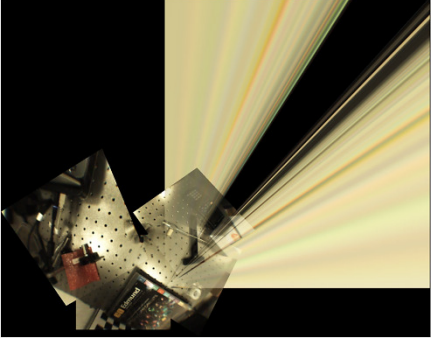



- [1] Fenske, M., et al. "A Design of a Liquid Crystal Based Single-Lens Stereo Endoscope." IEEE Xplore p. 43-44 (2006). DOI: 10.1109/NEBC.2006.1629743
- [2] Hattori, A., et al. "Development of a real-time image-guided system for stereo-endoscopic sinus surgery." Studies in Health Technology and Informatics 142 p. 112-116 (2009).
- [3] Hattori, T., et al. "Disparity and distortion free stereoscopic fiberscope." Proc. of SPIE 2653 p. 80-84 (1996).
- [4] Hu, T., et al. "Insertable Stereoscopic 3D Surgical Imaging Device with Pan and Tilt." IEEE / RAS-EMBS International Conference on Biomedical Robotics and Biomechanics (BIOROB) p. 311-316 (2008). DOI: 10.1.1.160.8134
- [5] Korniski, R., et al. "3D imaging with a single-aperture 3-mm objective lens: concept, fabrication, and test." Proc. of SPIE 8129 (812904) p. 1-11 (2011).
- [6] Martinez Herrera, S., et al. "Shape-from-Polarization in Laparoscopy." Proceedings / IEEE International Symposium on Biomedical Imaging: from nano to macro. IEEE International Symposium on Biomedical Imaging. (2013). DOI: 10.1109/ISBI.2013.6556798
- [7] Silvestri, M., et al. "Autostereoscopic Three-Dimensional Viewer Evaluation Through Comparison With Conventional Interfaces in Laparoscopic Surgery." Surgical Innovation XX(X) 1-8 (2011). DOI: 10.1177/1553350611411491
- [8] Song, C.-G., and Jin U. Kang. "Design of the Computerized 3D Endoscopic Imaging System for Delicate Endoscopic Surgery." J Med Syst 35:135-141 (2011). DOI 10.1007/s10916-009-9350-1
- [9] Tamadazte, B., et al. "Multi-view vision system for laparoscopy surgery." Int J CARS, published online (2014). DOI 10.1007/s11548-014-1064-2
- [10] Van Gompel, J., et al. "Field of View Comparison Between Two-Dimensional and Three-Dimensional Endoscopy." The Laryngoscope 00:000-000 p. 1-4 (2013). DOI: 10.1002/lary.24222
- [11] Zobel, J. "Basics of Three-Dimensional Endoscopic Vision." Minimally Invasive Therapy, 1 (1) p.36-9 (1993).




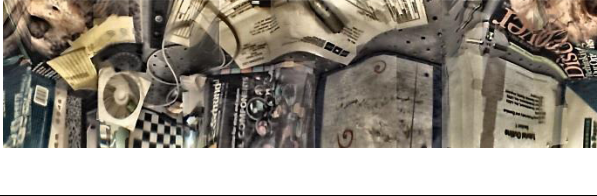


Used in preliminary-study sections

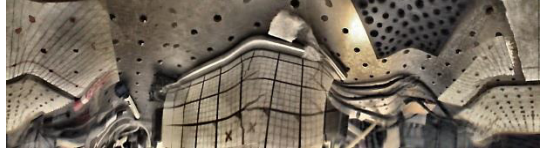

- [12] "Detect SURF features and return SURFPoints object - MATLAB detectSURFFeatures." The MathWorks, Inc. <https://www.mathworks.com/help/vision/ref/detectsurffeatures.html>. 2016. Accessed September 11, 2016.
- [13] Hoiem, D., and Russ Hewett. "Image Stitching – Computational Photography." University of Illinois. <https://courses.engr.illinois.edu/cs498dwh/fa2010/lectures/Lecture%2017%20-%20Photo%20Stitching.pdf>. October 19, 2010. Accessed September 11, 2016.
- [14] "iat_SIFTflow – Image Alignment Toolbox (IAT)." http://iatool.net/iat_SIFTflow/. 2016. Accessed September 11, 2016.
- [15] Lowe, D. "Demo Software: SIFT Keypoint Detector." University of British Columbia. <https://www.cs.ubc.ca/~lowe/keypoints/>. July 2005. Accessed September 11, 2016.
- [16] Lowe, D. "Object Recognition from Local Scale-Invariant Features." Proc. of the International Conference on Computer Vision, Corfu (Sept. 1999):1-8. <https://www.cs.ubc.ca/~lowe/papers/iccv99.pdf>. Accessed September 11, 2016.

Appendix A: Comparison of Image Stitching Results

Condition	Image	Notes

<p>Feature detection/linear transformation – standard homography (different image set from all other figures for this stitching method)</p>		<p>Example of grossly unreasonable transformation – perhaps due to misbehavior on periodic pattern</p>
<p>Feature detection/linear transformation method – affine transformation</p>		<p>Example of inadequate feature-point matches leading to incomplete stitching</p>
<p>Same input images as above – initial implementation of weakened homography method (not fully tuned)</p>		<p>A superior result for this configuration after some tuning is as shown in Fig. ##</p>
<p>RANSAC replaced with deterministic search, after some tuning – all images from here on use weakened homography</p>		<p>RANSAC tends to match some features very well while leaving others in bad condition; the deterministic search tends to make more tradeoffs and leave everything in a mediocre condition</p>

<p>SIFT feature detector replaced with SURF, after some tuning</p>		<p>A common error in all versions of the stitching procedure was the "c" in "Discover" being matched to the "o" in the adjacent shot</p>
<p>Spherical warp instead of cylindrical</p>		<p>Reverted this change (back to cylindrical warp) as all results were similarly poor</p>
<p><i>Optical flow introduced here</i></p>		
<p>Initial implementation of optical flow</p>		
<p>Smooth blend of flow vectors instead of simple average; parameter tuning</p>		
<p>Use sinusoidal curve for flow-vector blending in each overlap region</p>		
<p>Smoothly blend image content</p>		<p>Note "Dis<u>cc</u>over" and other matching difficulties for the near</p>

		depths at the top of the image, due in part to incomplete perspective pre-correction
Final parameter tuning		

Appendix B: MATLAB Source Code

coneundistortfolder.m (a manually-run script, not a function; older versions were named *cylundistortfolder.m* before the keystone correction was implemented, as mentioned in comments of the other script files)

```

%
% Before running, make sure:
% 1. needsConeUndistort folder exists and contains images from the camera
% 2. cameraParams has been loaded into the workspace somehow
% 3. Undistorted images from any previous run have been moved elsewhere

%
% averaged from several calibration trials
focLen = 377.803;

tempScaleFactor = 2;
scaledFocLen = focLen * tempScaleFactor;

tempInterpMethod = 'lanczos3';

midGray = 0.5;

% hand-tuned values, just sufficient to crop away all padding
targetHeight = 486;
targetWidth = 414;

% positive values make the "top" of each image wider than the "bottom"
perspectiveStrength = 0.52; % hand-tuned value

```

```

origFNames = dir(fullfile('needsConeUndistort','*.bmp'));
numImgs = size(origFNames, 1);
fprintf('Found %i images\n', numImgs);
for i=1:numImgs
    disp(strcat(num2str(i),': loading from disk'));

    curImg = imread(fullfile('needsConeUndistort',origFNames(i).name));

    disp(strcat(num2str(i),': initial cleanup'));
    curYIQ = custom_rgb2ntsc(curImg);

    disp('Working on luma...');
    curY = curYIQ(:,:,1);
    curY =
custom_usm(xx_denoise_luma(custom_ahc(curY)).*.75+custom_ahc(xx_denoise_luma(curY)).*.25);
    %
    disp('Working on chroma I...');
    curI = curYIQ(:,:,2);
    curI = xx_denoise_chroma(curI);
    %
    disp('Working on chroma Q...');
    curQ = curYIQ(:,:,3);
    curQ = xx_denoise_chroma(curQ);
    %
    curCleanupResult = ntsc2rgb(cat(3, curY, curI, curQ));

    disp(strcat(num2str(i), ': undistorting'));
    [curUndResult, ~] = undistortImage(curCleanupResult, cameraParams, 'cubic',
'OutputView', 'full', 'FillValues', midGray);
    % XXX: We throw away the focal-point location for convenience, but this is technically
inaccurate
    % Instead, should crop the undistortion result to bring the focal point to the image
center
    %
    % Rationale for parameters:
    % 'cubic': Make output as sharp as possible
    % 'full': Avoid throwing away any image data before the cylindrical warp, which will
itself require some cropping
    % ('valid' doesn't work for this calibration)
    % Use middle gray instead of default black for padding, to make sure flaws in the
cropping
    % are distinguishable from later flaws in stitching

    disp(strcat(num2str(i), ': temp enlarging'));
    curEnlarged = imresize(curUndResult, tempScaleFactor, tempInterpMethod);

```

```

disp(strcat(num2str(i), ': cylindrically warping'));
tempHgt = size(curEnlarged, 1);
tempWth = size(curEnlarged, 2);
curBigCyl = curEnlarged;
% We crop away unfilled space on the left/right side,
% for reasons of performance and accuracy with the perspective transform
% in the next step:
cylLeftPos = round(tempWth/2);
cylRightPos = round(tempWth/2);
% Initializing curBigCyl to a copy of curEnlarged is just in order to make sure it is
of the same class
% (to prevent misinterpretation of 0-255 vs. 0-1 scales). We crop away the bad parts
anyway...
for imageYIndex = 1:tempHgt
    for imageXIndex = 1:tempWth
        cylinderPoint = [(imageXIndex - tempWth/2)/scaledFocLen; (imageYIndex
- tempHgt/2)/scaledFocLen; 1];
        cylinderPoint = cylinderPoint ./ sqrt(cylinderPoint(1)^2 +
cylinderPoint(3)^2);

        % XXX: should use interp2 instead of this nearest-neighbor method
        % and do away with the pre-enlarge/re-shrink steps
        warpedImageTheta =
round(scaledFocLen*real(asin(cylinderPoint(1)))+tempWth/2);
        warpedImageH = round(scaledFocLen*cylinderPoint(2)+tempHgt/2);

        if warpedImageH > 0 && warpedImageH <= tempHgt && warpedImageTheta > 0
&& warpedImageTheta <= tempWth
            curBigCyl(warpedImageH, warpedImageTheta, 1) =
curEnlarged(imageYIndex, imageXIndex, 1);
            curBigCyl(warpedImageH, warpedImageTheta, 2) =
curEnlarged(imageYIndex, imageXIndex, 2);
            curBigCyl(warpedImageH, warpedImageTheta, 3) =
curEnlarged(imageYIndex, imageXIndex, 3);

            cylLeftPos = min(cylLeftPos, warpedImageTheta);
            cylRightPos = max(cylRightPos, warpedImageTheta);
        end
    end
end
disp('DEBUG: cropping unused space on left/right');
curBigCyl = curBigCyl(:, cylLeftPos:cylRightPos, :);
tempWth = size(curBigCyl, 2);

disp(strcat(num2str(i), ': adjusting perspective'));
refAngle = pi/numImgs;
perspOffset = perspectiveStrength*tan(refAngle)*tempHgt/2;
if i==1 % better to display at the end (i==numImgs)?

```

```

        fprintf('DEBUG: perspOffset is %.4f (compare tempWth %i)\n', perspOffset,
tempWth);
    end
    curBigCylr = curBigCyl(:, :, 1);
    curBigCylg = curBigCyl(:, :, 2);
    curBigCylb = curBigCyl(:, :, 3);
    curBigCylPr = simple_keystone(curBigCylr, perspOffset);
    curBigCylPg = simple_keystone(curBigCylg, perspOffset);
    curBigCylPb = simple_keystone(curBigCylb, perspOffset);
    curBigCylP = cat(3, curBigCylPr, curBigCylPg, curBigCylPb);

    disp(strcat(num2str(i), ': re-shrinking'));
    curSmallCyl = imresize(curBigCylP, 1/tempScaleFactor, tempInterpMethod);

    disp(strcat(num2str(i), ': cropping to final size'));
    smallCylHgt = size(curSmallCyl, 1);
    smallCylWth = size(curSmallCyl, 2);
    if smallCylHgt > targetHeight && smallCylWth > targetWidth
        yOrigin = round((smallCylHgt-targetHeight)/2);
        xOrigin = round((smallCylWth-targetWidth)/2);
        curCylResult = imcrop(curSmallCyl, [xOrigin yOrigin targetWidth-1
targetHeight-1]);
    else
        disp('intermediate image too small to crop!');
        disp('Manual attention required - check crop settings in this script');
        curCylResult = curSmallCyl;
    end

    disp(strcat(num2str(i), ': saving to disk'));
    imwrite(curCylResult, strcat('coneUndistorted', num2str(i), '.png'));

end

```

custom_rgb2ntsc.m

```

%
% Luma information is taken verbatim from the unprocessed image, because we
% normalize it ourselves later.
% Chroma information is taken after some contrast stretching in RGB space.
function [ im_ntsc ] = custom_rgb2ntsc( im_rgb )

raw_result = rgb2ntsc(im_rgb);

adj_separate = rgb2ntsc(cat(3, imadjust(im_rgb(:, :, 1)), imadjust(im_rgb(:, :, 2)),
imadjust(im_rgb(:, :, 3))));

im_ntsc = cat(3, raw_result(:, :, 1), (raw_result(:, :, 2)+adj_separate(:, :, 2))./2,
(raw_result(:, :, 3)+adj_separate(:, :, 3))./2);

```

```
end %function
```

custom_usm.m

```
%
% input image must be grayscale and of class double
% (we call rgb2ntsc and extract individual channels in cylundistortfolder.m)
function [ im_sharpened ] = custom_usm( img )

% Configuration constants
amo = 4.8; % tuned visually for optical-flow method; SIFT/RANSAC needs lower
thr = 1.0/32.0;

% based on the "octave sharpening" technique sometimes used for photos
usm1 = imsharpen(img, 'Radius', 1/sqrt(2), 'Amount', amo, 'Threshold', thr);
usm2 = imsharpen(img, 'Radius', sqrt(2), 'Amount', amo, 'Threshold', thr);
usm3 = imsharpen(img, 'Radius', 2*sqrt(2), 'Amount', amo, 'Threshold', thr);
usm4 = imsharpen(img, 'Radius', 4*sqrt(2), 'Amount', amo, 'Threshold', thr);
%
im_sharpened = (invmedsharp(img, amo) .* 16 + usm1 .* 8 + usm2 .* 4 + usm3 .* 2 + usm4) ./ 31;

end %function
```

invmedsharp.m

```
% An ad-hoc sharpening method intended to enhance small blob features while avoiding edges
% input image must be grayscale(?)
% (we call rgb2ntsc and extract individual channels in cylundistortfolder.m)
function [ im_sharpened ] = invmedsharp( img, amount )

mf_small = medfilt2(img, [3 3], 'symmetric');
mf_wide = medfilt2(img, [3 5], 'symmetric');
mf_tall = medfilt2(img, [5 3], 'symmetric');
mf_big = medfilt2(img, [5 5], 'symmetric');

mf = (mf_small .* 25 + mf_wide .* 15 + mf_tall .* 15 + mf_big .* 9) ./ 64;

im_sharpened = img + (img-mf).*amount;

end %function
```

xx_denoise_luma.m

```
%
% input image must be grayscale and of class double
% (we call rgb2ntsc in cylundistortfolder.m)
function [ im_denoised ] = xx_denoise_luma( img )
```

```

onestep    = cleanup_denoise_step(img);

twosteps   = cleanup_denoise_step((img+onestep)./2);

threesteps = cleanup_denoise_step((onestep+twosteps)./2);

im_denoised = (0.45*twosteps+0.55*threesteps);

end %function

```

cleanup_denoise_step.m

```

%
% input image must be grayscale and of class double
% (we call rgb2ntsc in cylundistortfolder.m)
function [ im_denoised ] = cleanup_denoise_step( img )

[imwie1, ~] = wiener2(img, [5 11]);
[imwie2, ~] = wiener2(img, [7 9]);
[imwie3, ~] = wiener2(img, [9 7]);
[imwie4, ~] = wiener2(img, [11 5]);

immed1 = medfilt2(img, [3 5], 'symmetric');
immed2 = medfilt2(img, [5 3], 'symmetric');

% subtle cleanup for fine specks
cross_nhood = [0 1 0; 1 1 1; 0 1 0];
weak_lo = ordfilt2(img, 2, cross_nhood, 'symmetric');
weak_hi = ordfilt2(img, 4, cross_nhood, 'symmetric');

im_denoised = ( imwie1 + imwie2 + imwie3 + imwie4 + immed1 + immed2 + weak_lo + weak_hi ) ./ 8;

end %function

```

custom_ahem

```

%
% input image must be grayscale and of class double
% (we call rgb2ntsc in cylundistortfolder.m)
function [ im_contrasted ] = custom_ahem( img )

simple_adjust = imadjust(img);
ahem_56_uni = adapthisteq(img, 'NumTiles', [5 6], 'ClipLimit', 0.02, 'Distribution', 'uniform');
ahem_56_exp = adapthisteq(img, 'NumTiles', [5 6], 'ClipLimit', 0.02, 'Distribution', 'exponential');
ahem_56_ray = adapthisteq(img, 'NumTiles', [5 6], 'ClipLimit', 0.02, 'Distribution', 'rayleigh');
ahem_65_uni = adapthisteq(img, 'NumTiles', [6 5], 'ClipLimit', 0.02, 'Distribution', 'uniform');
ahem_65_exp = adapthisteq(img, 'NumTiles', [6 5], 'ClipLimit', 0.02, 'Distribution', 'exponential');
ahem_65_ray = adapthisteq(img, 'NumTiles', [6 5], 'ClipLimit', 0.02, 'Distribution', 'rayleigh');

```

```

% We want to have an even number of inputs to the median, so that two of them
% will be averaged. Thus, get the above 7 things down to 6 by averaging the
% CLAHE results for each distribution and each grid size
ahe_avg_uni = (ahe_56_uni + ahe_65_uni) ./ 2;
ahe_avg_exp = (ahe_56_exp + ahe_65_exp) ./ 2;
ahe_avg_ray = (ahe_56_ray + ahe_65_ray) ./ 2;
ahe_avg_56 = (ahe_56_uni + ahe_56_exp + ahe_56_ray) ./ 3;
ahe_avg_65 = (ahe_65_uni + ahe_65_exp + ahe_65_ray) ./ 3;

im_contrasted = median(cat(3, simple_adjust, ahe_avg_uni, ahe_avg_exp, ahe_avg_ray, ahe_avg_56,
ahe_avg_65), 3);

end %function

```

xx_denoise_chroma.m

```

%
% input image must be grayscale and of class double
% (we call rgb2ntsc in cylundistortfolder.m)
function [ im_denoised ] = xx_denoise_chroma( img )

tempA = cleanup_denoise_step(img);
tempB = imgaussfilt(img, 0.6);
tempC = (tempA+tempB)./2;

tempAA = cleanup_denoise_step(tempC);
tempBB = imgaussfilt(tempC, 0.6);
tempCC = (tempAA+tempBB)./2;

tempA = cleanup_denoise_step(tempCC);
tempB = imgaussfilt(tempCC, 0.6);
im_denoised = (tempA + tempB) ./ 2;

end %function

```

simple_keystone.m

```

% Applies a keystone distortion effect
% This is NOT a homography; it is a naive line-by-line effect
% (similar to the keystone adjustment on a CRT monitor).
%
% For simplicity we assume that img is grayscale and of class double.
function [ im_persp ] = simple_keystone( img, pOffs )

width_i = size(img, 2);
hght = size(img, 1);

width_o = round(width_i+2*abs(pOffs));

```

```

im_persp = zeros(hght, wdth_o);
for i_row = 1:hght
    begOfRow = 1+(2*pOffs)*(i_row-1)/(hght-1);
    endOfRow = wdth_o-(2*pOffs)*(i_row-1)/(hght-1);
    if pOffs < 0
        begOfRow = begOfRow-2*pOffs; % i.e. +2*abs(pOffs)
        endOfRow = endOfRow+2*pOffs; % i.e. -2*abs(pOffs)
    end
    for i_col = 1:wdth_o
        if i_col >= begOfRow && i_col <= endOfRow
            % Pixel inside the valid image area
            srcCol = 1+(wdth_i-1)*(i_col-begOfRow)/(endOfRow-begOfRow);
            if srcCol == floor(srcCol)
                % Pixel read from an exact integer location
                im_persp(i_row, i_col) = img(i_row, srcCol);
            else
                % Naive linear interpolation within the row
                pixLft = img(i_row, floor(srcCol));
                pixRgt = img(i_row, ceil(srcCol));
                interp_factor = srcCol-floor(srcCol);
                im_persp(i_row, i_col) = interp_factor * pixRgt + (1-interp_factor) *
pixLft;
            end
        else
            % Pixel outside the valid image area - fill w/ medium gray
            im_persp(i_row, i_col) = 0.5;
        end
    end
end
end

end %function

```

performFolderMod.m (invoked from the MATLAB command line to start the actual stitching process for the feature-detection/linear-transformation approach)

```

% Make a panorama, using images from a folder
% Based very loosely on
http://www.tobw.net/index.php?cat\_id=2&project=Panorama+Stitching+Demo+in+Matlab (link now dead)
% filename lexicographical order MUST represent the linear progression in which the images were taken
function [ im_stitched ] = performFolderMod( folder )

% undistort script will read BMPs and write PNGs
images = dir(fullfile(folder, '*.png'));
n_images = size(images, 1);
fprintf('Found %i images\n', n_images);
if n_images < 3
    % not worthwhile to support the case of only 2 images
    error('Need at least three images (in PNG format)');
end

```



```

end

central_idx = floor((n_images+1)/2);

isFirstStitch = true;

min_pts_needed = 5; % manually adjust for chosen transformation method

% This approach is inefficient in that we read and feature-detect each image twice.
% However, the obvious solution would require logic currently in CVTBmatch to be duplicated here...
for idx = central_idx:n_images-1

    fprintf('About to begin stitching images %i and %i\n', idx, idx+1);

    im1 = imread(fullfile(folder, images(idx).name));
    if isFirstStitch
        % Ensure that im_stitched is initialized, so that if we fail to match the
        % image pairs on either side, the function will return something sane instead of
        erroring out
        im_stitched = im1;
    end
    im2 = imread(fullfile(folder, images(idx+1).name));

    try
        [pts1 pts2] = CVTBmatch( im1, im2 );
    catch siftExcpt
        % The name "sift" for the exception variable is a historical artifact;
        % this code is for SURF.
        disp('Failed to obtain matched pairs of feature points.');
```

disp('Giving up and switching to other side. Error description was:');

disp(siftExcpt.message);

break

end

n_pts_matched = length(pts1);

if n_pts_matched < min_pts_needed

fprintf('Insufficient matches between images %i and %i (have %i, need %i).\n', idx,

idx+1, n_pts_matched, min_pts_needed);

disp('Make sure files are named in proper sequence. Giving up and switching to other

side.');

break

end

try

[T_curr_im2, ~] = determ_search(pts2, pts1, 'proj_mix');

catch calcXformExcpt

```

disp('Failed to create transformation; this probably means that determ_search');
disp('or a transformation method has a fatal implementation mistake.');
```

```

disp('Giving up and switching to other side. Error description was:');
disp(calcXformExcpt.message);

break
end

% T_prev_pos tracks the extent to which the first (central) image has been pushed
% down and/or to the right by im_stitched growing at the top and/or left.
% Such information is kept separate from T_prev_im2, the composition of generated transforms.
if isFirstStitch
    [im_stitched, stitched_mask, ~, ~, T_prev_pos] = stitch_a(im1, im2, T_curr_im2);

    T_prev_im2 = T_curr_im2;

    isFirstStitch = false;
else
    T_im2_unshifted = maketform('composite', T_curr_im2, T_prev_im2);
    T_im2 = maketform('composite', T_prev_pos, T_im2_unshifted);

    [im_stitched, stitched_mask, ~, ~, T_next_pos] = stitch_a(im_stitched, im2, T_im2,
stitched_mask);

    T_prev_pos = maketform('composite', T_next_pos, T_prev_pos);
    T_prev_im2 = T_im2_unshifted;
end

warning off all % for size warning
    imshow(im_stitched);
warning on all
end %for (1st side)

isFirstStitchOn2ndSide = true;

for idx = central_idx:-1:2

    fprintf('About to begin stitching images %i and %i\n', idx-1, idx);

    im1 = imread(fullfile(folder, images(idx).name));
    im2 = imread(fullfile(folder, images(idx-1).name));

    try
        [pts1 pts2] = CVTbmatch( im1, im2 );
    catch siftExcpt
        disp('Failed to obtain matched pairs of feature points.');
```

```

        disp('Giving up. Error description was:');
```

```

        disp(siftExcpt.message);

        break
    end

    n_pts_matched = length(pts1);
    if n_pts_matched < min_pts_needed
        fprintf('Insufficient matches between images %i and %i (have %i, need %i).\n', idx-1,
idx, n_pts_matched, min_pts_needed);
        disp('Make sure files are named in proper sequence. Giving up.');
```

```

        break
    end

    try
        [T_curr_im2, ~] = determ_search( pts2, pts1, 'proj_mix' );
    catch calcXformExcpt
        disp('Failed to create transformation; this probably means that determ_search');
        disp('or a transformation method has a fatal implementation mistake.');
```

```

        disp('Giving up. Error description was:');
        disp(calcXformExcpt.message);

        break
    end

    if isFirstStitchOn2ndSide
        if isFirstStitch
            % Uh oh, the very first pair on the other side failed to match -
            % that is the only way we could have gotten here. We need this special
            % case because otherwise stitched_mask would not be defined.
            [im_stitched, stitched_mask, ~, ~, T_prev_pos] = stitch_a(im1, im2,
T_curr_im2);

            T_prev_im2 = T_curr_im2;

            isFirstStitch = false;
        else
            % we need to keep the previously-calculated T_prev_pos and stitched_mask,
            % but as we are starting again at the central image, we do not want to
            preserve T_prev_im2
            T_im2 = maketform('composite', T_prev_pos, T_curr_im2);

            [im_stitched, stitched_mask, ~, ~, T_next_pos] = stitch_a(im_stitched, im2,
T_im2, stitched_mask);

            T_prev_pos = maketform('composite', T_next_pos, T_prev_pos);
            T_prev_im2 = T_curr_im2;
        end
    end

```

```

        isFirstStitchOn2ndSide = false;
    else
        T_im2_unshifted = maketform('composite', T_curr_im2, T_prev_im2);
        T_im2 = maketform('composite', T_prev_pos, T_im2_unshifted);

        [im_stitched, stitched_mask, ~, ~, T_next_pos] = stitch_a(im_stitched, im2, T_im2,
stitched_mask);

        T_prev_pos = maketform('composite', T_next_pos, T_prev_pos);
        T_prev_im2 = T_im2_unshifted;
    end

    warning off all % for size warning
        imshow(im_stitched);
    warning on all
end %for (2nd side)

imwrite(im_stitched, strcat('stitchResult-', folder, '.png'));

% make sure we show the last state in case we broke out of the 2nd loop
warning off all % for size warning
    imshow(im_stitched);
warning on all

end %function

```

CVTBmatch.m

```

% Generate matched pairs of features between im1 and im2
% using MATLAB's (C)omputer (V)ision System (T)ool(b)ox.
% (should probably have called this SURFmatch...)
%
% Returns the lists of matched points for both images (always nonempty).
% Currently, error is thrown if no matches are found.
%
function [points1, points2] = CVTBmatch(im1, im2)

    % Config constants
    dsf_mt = 983; % lower = more lenient
    dsf_nsl = 5; % (don't touch it)
    mf_mt = 1.07; % higher = more lenient matching
    mf_mr = 0.56; % higher = more lenient matching

    im1red = im1(:,:,1);
    im1green = im1(:,:,2);
    im1blue = im1(:,:,3);
    im1gray = rgb2gray(im1);
    im2red = im2(:,:,1);

```

```

im2green = im2(:,:,2);
im2blue  = im2(:,:,3);
im2gray  = rgb2gray(im2);

disp('DEBUG: channel split done');

% Get lists of raw points
rawpts1red  = detectSURFFeatures(im1red, 'MetricThreshold', dsf_mt, 'NumScaleLevels',
dsf_nsl);
rawpts1green = detectSURFFeatures(im1green, 'MetricThreshold', dsf_mt, 'NumScaleLevels',
dsf_nsl);
rawpts1blue  = detectSURFFeatures(im1blue, 'MetricThreshold', dsf_mt, 'NumScaleLevels',
dsf_nsl);
rawpts1gray  = detectSURFFeatures(im1gray, 'MetricThreshold', dsf_mt, 'NumScaleLevels',
dsf_nsl);
rawpts2red  = detectSURFFeatures(im2red, 'MetricThreshold', dsf_mt, 'NumScaleLevels',
dsf_nsl);
rawpts2green = detectSURFFeatures(im2green, 'MetricThreshold', dsf_mt, 'NumScaleLevels',
dsf_nsl);
rawpts2blue  = detectSURFFeatures(im2blue, 'MetricThreshold', dsf_mt, 'NumScaleLevels',
dsf_nsl);
rawpts2gray  = detectSURFFeatures(im2gray, 'MetricThreshold', dsf_mt, 'NumScaleLevels',
dsf_nsl);

disp('DEBUG: detectSURFFeatures done');

% Get features corresponding to raw points
% SURFSize of 128 makes this really finicky (i.e. fewer matches)?
[feats1red, rawpts1red] = extractFeatures(im1red, rawpts1red);
[feats1green, rawpts1green] = extractFeatures(im1green, rawpts1green);
[feats1blue, rawpts1blue] = extractFeatures(im1blue, rawpts1blue);
[feats1gray, rawpts1gray] = extractFeatures(im1gray, rawpts1gray);
[feats2red, rawpts2red] = extractFeatures(im2red, rawpts2red);
[feats2green, rawpts2green] = extractFeatures(im2green, rawpts2green);
[feats2blue, rawpts2blue] = extractFeatures(im2blue, rawpts2blue);
[feats2gray, rawpts2gray] = extractFeatures(im2gray, rawpts2gray);

disp('DEBUG: extractFeatures done');

% Match up the features
idxsRed  = matchFeatures(feats1red, feats2red, 'Unique', true, 'MatchThreshold', mf_mt,
'MaxRatio', mf_mr);
idxsGreen = matchFeatures(feats1green, feats2green, 'Unique', true, 'MatchThreshold', mf_mt,
'MaxRatio', mf_mr);
idxsBlue  = matchFeatures(feats1blue, feats2blue, 'Unique', true, 'MatchThreshold', mf_mt,
'MaxRatio', mf_mr);
idxsGray  = matchFeatures(feats1gray, feats2gray, 'Unique', true, 'MatchThreshold', mf_mt,
'MaxRatio', mf_mr);

```

```

disp('DEBUG: matchFeatures done');

if isempty(idxsRed)
    anyRedMatches = false;
    disp('No matches on red!');
else
    mpts1red = rawpts1red( idxsRed(:,1), :);
    mpts2red = rawpts2red( idxsRed(:,2), :);
    anyRedMatches = true;
end
if isempty(idxsGreen)
    anyGreenMatches = false;
    disp('No matches on green!');
else
    mpts1green = rawpts1green(idxsGreen(:,1), :);
    mpts2green = rawpts2green(idxsGreen(:,2), :);
    anyGreenMatches = true;
end
if isempty(idxsBlue)
    anyBlueMatches = false;
    disp('No matches on blue!');
else
    mpts1blue = rawpts1blue( idxsBlue(:,1), :);
    mpts2blue = rawpts2blue( idxsBlue(:,2), :);
    anyBlueMatches = true;
end
if isempty(idxsGray)
    anyGrayMatches = false;
    disp('No matches on luma!');
else
    mpts1gray = rawpts1gray( idxsGray(:,1), :);
    mpts2gray = rawpts2gray( idxsGray(:,2), :);
    anyGrayMatches = true;
end

disp('DEBUG: indexing done');

% Pull the plain arrays out of the SURFPoints objects
if anyRedMatches
    ppts1red = mpts1red.Location;
    ppts2red = mpts2red.Location;
else
    ppts1red = [-1 -1]; % sentinel value
    ppts2red = [-1 -1]; % real points will always be nonnegative
end
if anyGreenMatches
    ppts1green = mpts1green.Location;

```

```

        ppts2green = mpts2green.Location;
    end
    if anyBlueMatches
        ppts1blue = mpts1blue.Location;
        ppts2blue = mpts2blue.Location;
    end
    if anyGrayMatches
        ppts1gray = mpts1gray.Location;
        ppts2gray = mpts2gray.Location;
    end

    disp('DEBUG: .Location done');

    % Pool together the point pairs from each color channel
    points1 = ppts1red;
    points2 = ppts2red;
    if anyGreenMatches
        points1 = [points1; ppts1green];
        points2 = [points2; ppts2green];
    end
    if anyBlueMatches
        points1 = [points1; ppts1blue];
        points2 = [points2; ppts2blue];
    end
    if anyGrayMatches
        points1 = [points1; ppts1gray];
        points2 = [points2; ppts2gray];
    end

    disp('DEBUG: ; done');

    % Although we used the 'Unique' parameter above, uniqueness is ensured only
    % within each channel; pooling the channels may have created duplication,
    % which we need to remove here
    if anyRedMatches || anyGreenMatches || anyBlueMatches || anyGrayMatches
        pts=unique([points1 points2], 'rows');

        % If red channel failed to match, we have to remove the sentinel value.
        % We can only get here if at least one other channel *has* matches, so
        % we know that 2:end should be valid.
        if pts(1,1)==-1
            pts=pts(2:end, :);
        end

        fprintf('Found %i matches in total\n',size(pts,1));

        % CV toolbox methods return single, but t'form methods require double
        points1 = double(pts(:, [1 2]));

```



```

        points2 = double(pts(:,[3 4]));
    else
        % XXX: does this need to be fatal, or should we return empty arrays
        % and let the caller handle the situation?
        error('No matches found');
    end %if
end %function

```

determ_search.m

```

% Calculates a transformation that aligns the points points1 and points2
% using a deterministic search (eliminating one point at a time) to avoid
% outliers.
% transMode: aff_lsq - Affine mapping
%             proj_svd - Homography
% n_pts:      size of point sample
% Return values:
% T_im1:      a tform object encapsulating the transformation from image1
%             (points1) onto image2 (points2)
% best_pts:   points used to estimate best transformation (Dim: arbitrary x 4)
%
% This function, and the subroutines for specific transformation modes, are
% based on the RANSAC implementation from
% http://www.tobw.net/index.php?cat\_id=2&project=Panorama+Stitching+Demo+in+Matlab
% (link now dead)
function [ T_im1, best_pts ] = determ_search( points1, points2, transMode )

disp('Performing deterministic search');
t = tic();

best_score = -99;
it_improv={}; % saves the inlier score per improvement

switch transMode
    case 'aff_lsq'
        min_n_pts = 4;
    case 'proj_svd'
        min_n_pts = 5;
    case 'proj_mix'
        min_n_pts = 5;
    otherwise
        error('transmode not known');
end

n_pts_provided = length(points2(:,1));
if n_pts_provided < min_n_pts
    error('You must provide enough points to overdetermine the chosen transformation type');
end

```

```

% Special case: There were just enough points provided, so no "search" is needed
if n_pts_provided == min_n_pts
    disp('Have just enough points - calculating transformation without searching subsets');
    disp('(poor results likely!)');

    best_pts = [points1 points2];

    warning off all
    switch transMode
        case 'aff_lsq'
            T_im1 = affine_leastsquare(points1, points2);
        case 'proj_svd'
            T_im1 = homography_svd(points1, points2);
        case 'proj_mix'
            T_im1 = weak_homography(points1, points2);
        otherwise
            % should not get here - should already have errored out above!
            error('transmode not known');
    end
    warning on all

    return
end

% Configuration constants
scoring_sigma = 2.35;
it_improv_max_print_len = 50;
%

% Choose some key points with which to check the sanity of the transformation.
% The more it looks like a rigid motion, the more reasonable we consider it to
% be. We draw a few lines across the bounding box of the input points; the more
% their lengths change, the lower our figure of merit (more logic in loops below)
ptslmins = min(points1);
ptslmaxs = max(points1);
ptslmids = (mean(points1) + median(points1) + ptslmins + ptslmaxs) ./ 4;

ptsl1ft = ptslmins(1);
ptslrgt = ptslmaxs(1);
ptsltop = ptslmins(2);
ptslbot = ptslmaxs(2);

ptslmdx = ptslmids(1);
ptslmdy = ptslmids(2);

bboxwidth = ptslrgt - ptsl1ft;
bboxhght = ptslbot - ptsltop;
bboxdiag = sqrt(bboxwidth*bboxwidth + bboxhght*bboxhght);

```

```

refpt1 = [pts1lft pts1top 1];
refpt2 = [pts1lft pts1mdy 1];
refpt3 = [pts1lft pts1bot 1];
refpt4 = [pts1mdx pts1bot 1];
refpt5 = [pts1rgt pts1bot 1];
refpt6 = [pts1rgt pts1mdy 1];
refpt7 = [pts1rgt pts1top 1];
refpt8 = [pts1mdx pts1top 1];

if n_pts_provided > 16
    disp('More than 16 matches; using median cut');
    [temppts1 temppts2] = medcut_to_16pts(points1, points2);
    n_pts_searchspace = 16;
else
    disp('No median cut needed');
    temppts1 = points1;
    temppts2 = points2;
    n_pts_searchspace = n_pts_provided;
end

% DIRTY HACK to work around class problems
all_idx = zeros(n_pts_searchspace, 1);
for i_setup = 1:n_pts_searchspace
    all_idx(i_setup)=i_setup;
end

bf_n_subsets = nchoosek(n_pts_searchspace, min_n_pts);
fprintf('Brute-force-testing %i subsets of size %i (please be patient)\n', bf_n_subsets, min_n_pts);

bf_subsets = nchoosek(all_idx, min_n_pts);
for i_bf = 1:bf_n_subsets
    bf_idx = bf_subsets(i_bf,:);

    warning off all % XXX: why? (this was in TobW's original code)
    switch transMode
        case 'aff_lsq'
            M = affine_leastsquare_mat(temppts1(bf_idx,:), temppts2(bf_idx,:));
            T = maketform('affine', M);
        case 'proj_svd'
            M = homography_svd_mat(temppts1(bf_idx,:), temppts2(bf_idx,:));
            T = maketform('projective', M);
        case 'proj_mix'
            M = weak_homography_mat(temppts1(bf_idx,:), temppts2(bf_idx,:));
            T = maketform('projective', M);
        otherwise
            % should not get here - should already have errored out above!
            error('transmode not known');
    end
end

```

```

end
warning on all
% transformation check
if max(max(isnan(T.tdata.T)))==1
    error('Unexpected NaN in transformation'); % XXX: does this really need to be fatal?
    % or can we simply "continue"? (can this ever happen in current MATLAB vers.?)
end

% NB: We *evaluate the quality* of the transformation against ALL points,
% not just the 16 returned by the median cut.

% Apply the transformation ...
[A_X A_Y] = tformfwd(T,points1(:,1),points1(:,2));
dXsq = (A_X - points2(:,1)).^2;
dYsq = (A_Y - points2(:,2)).^2;

% ... tally up the inlier score ...
inlier_score=0;
for i=1:length(dXsq)
    esq = dXsq(i)+dYsq(i);
    inlier_score = inlier_score + exp(-esq/(2*scoring_sigma*scoring_sigma));
end

% ... transform the reference points, check sanity ...
trnpt1 = M.' * refpt1.';
trnpt2 = M.' * refpt2.';
trnpt3 = M.' * refpt3.';
trnpt4 = M.' * refpt4.';
trnpt5 = M.' * refpt5.';
trnpt6 = M.' * refpt6.';
trnpt7 = M.' * refpt7.';
trnpt8 = M.' * refpt8.';

% - normalize the vectors...
trnpt1 = trnpt1 ./ trnpt1(3);
trnpt2 = trnpt2 ./ trnpt2(3);
trnpt3 = trnpt3 ./ trnpt3(3);
trnpt4 = trnpt4 ./ trnpt4(3);
trnpt5 = trnpt5 ./ trnpt5(3);
trnpt6 = trnpt6 ./ trnpt6(3);
trnpt7 = trnpt7 ./ trnpt7(3);
trnpt8 = trnpt8 ./ trnpt8(3);
% - ...and the 3rd component will subtract away
diag1dist = norm(trnpt5-trnpt1);
horizdist = norm(trnpt6-trnpt2);
diag2dist = norm(trnpt7-trnpt3);
vert_dist = norm(trnpt8-trnpt4);

```

```

diag1ratio = diag1dist/bboxdiag;
horizratio = horizdist/bboxwidth;
diag2ratio = diag2dist/bboxdiag;
vert_ratio = vert_dist/bboxhght;

diag1invrat = bboxdiag/diag1dist;
horizinvrat = bboxwidth/horizdist;
diag2invrat = bboxdiag/diag2dist;
vert_invrat = bboxhght/vert_dist;

diag1log = log(diag1ratio);
horizlog = log(horizratio);
diag2log = log(diag2ratio);
vert_log = log(vert_ratio);

% -empirical: geo.mean of Lorentzian-curve-of-log with SQRT of whichever ratio is <=1
diag1merit = sqrt( (1/(diag1log*diag1log+1)) * sqrt(min(diag1ratio, diag1invrat)) );
horizmerit = sqrt( (1/(horizlog*horizlog+1)) * sqrt(min(horizratio, horizinvrat)) );
diag2merit = sqrt( (1/(diag2log*diag2log+1)) * sqrt(min(diag2ratio, diag2invrat)) );
vert_merit = sqrt( (1/(vert_log*vert_log+1)) * sqrt(min(vert_ratio, vert_invrat)) );

rigidity_factor = min([diag1merit horizmerit diag2merit vert_merit]);

% ...and multiply to obtain the final score
adjusted_score = inlier_score * rigidity_factor;

% improvement check
if adjusted_score > best_score
    best_score = adjusted_score;
    T_im1 = T;
    it_improv(end+1) = best_score;
    best_idx = bf_idx;
end
end % for i_bf

if numel(it_improv) <= it_improv_max_print_len
    disp('High scores:');
    disp(it_improv);
else
    disp('Too many high-score updates to display');
end

best_pts=zeros(length(best_idx),4);
best_pts(:, [1 2])=points1(best_idx,:);
best_pts(:, [3 4])=points2(best_idx,:);

disp('done.')
```

```
toc(t)
```

```
end % function
```

affine_leastsquare.m

```
function [ T ] = affine_leastsquare( pts1, pts2 )
```

```
% ... create a object used by 'imtransform'
```

```
T = maketform('affine', affine_leastsquare_mat(pts1, pts2));
```

```
end
```

affine_leastsquare_mat.m

```
function [ X ] = affine_leastsquare_mat( pts1, pts2 )
```

```
% for more details, please see: Lecture 12, page 51,
```

```
http://www.vision.ee.ethz.ch/~bleibe/multimedia/teaching/cv-ws08/cv08-part12-local-features2.pdf
```

```
% prepare matrix A with pts1
```

```
A = zeros(size(pts1,1)*2,6);
```

```
A(1:2:end,5) = 1;
```

```
A(2:2:end,6) = 1;
```

```
A(1:2:end,1:2) = pts1;
```

```
A(2:2:end,3:4) = pts1;
```

```
% prepare matrix B with pts2
```

```
B = zeros(size(pts2,1)*2,1);
```

```
B(1:2:end)=pts2(:,1);
```

```
B(2:2:end)=pts2(:,2);
```

```
% solve  $A*x = B$  for x using least square error
```

```
x = A\B;
```

```
% reorder elements of x
```

```
X(1,1)=x(1);
```

```
X(1,2)=x(3);
```

```
X(2,1)=x(2);
```

```
X(2,2)=x(4);
```

```
X(3,1)=x(5);
```

```
X(3,2)=x(6);
```

```
X(1,3)=0;
```

```
X(2,3)=0;
```

```
X(3,3)=1;
```

```
end
```

homography_svd.m

```
% Calculate a homography that maps im1 to im2
function T = homography_svd(points1, points2)

T = maketform('projective', homography_svd_mat(points1, points2));

% The Matlab way:
% (only works with 4 points)
%T = maketform('projective', points2, points1);
```

homography_svd_mat.m

```
% Calculate a homography that maps im1 to im2
function x = homography_svd_mat(points1, points2)

% for more details, please see: Lecture 12, page 60,
http://www.vision.ee.ethz.ch/~bleibe/multimedia/teaching/cv-ws08/cv08-part12-local-features2.pdf
% Build matrix
xaxb = points2(:,1) .* points1(:,1);
xayb = points2(:,1) .* points1(:,2);
yaxb = points2(:,2) .* points1(:,1);
yayb = points2(:,2) .* points1(:,2);

A = zeros(size(points1, 1)*2, 9);
A(1:2:end,3) = 1;
A(2:2:end,6) = 1;
A(1:2:end,1:2) = points1;
A(2:2:end,4:5) = points1;
A(1:2:end,7) = -xaxb;
A(1:2:end,8) = -xayb;
A(2:2:end,7) = -yaxb;
A(2:2:end,8) = -yayb;
A(1:2:end,9) = -points2(:,1);
A(2:2:end,9) = -points2(:,2);

% Solve using smallest eigenvector
[U,S,V] = svd(A);
h = V(:,9) ./ V(9,9);
x = reshape(h,3,3);
```

weak_homography.m

```
% Calculate a homography that maps im1 to im2
% (biased toward an affine transform to limit perspective blowup)
function T = weak_homography(points1, points2)

T=maketform('projective', weak_homography_mat(points1, points2));
```


weak_homography_mat.m

```

% Calculate a homography that maps im1 to im2
% (biased toward an affine transform to limit perspective blowup)
function M = weak_homography_mat(points1, points2)

% compute the affine xform and homography the usual way
ma = affine_leastsquare_mat(points1, points2);
mh = homography_svd_mat(points1, points2);

% pick some reference points
ptslmins = min(points1);
ptslmids = (mean(points1)+median(points1))./2;
ptslmaxs = max(points1);

refpt1 = [ptslmins(1) ptslmins(2) 1];
refpt2 = [ptslmins(1) ptslmids(2) 1];
refpt3 = [ptslmins(1) ptslmaxs(2) 1];
refpt4 = [ptslmids(1) ptslmins(2) 1];
refpt5 = [ptslmids(1) ptslmids(2) 1];
refpt6 = [ptslmids(1) ptslmaxs(2) 1];
refpt7 = [ptslmaxs(1) ptslmins(2) 1];
refpt8 = [ptslmaxs(1) ptslmids(2) 1];
refpt9 = [ptslmaxs(1) ptslmaxs(2) 1];

% transform the reference points per the transformations calculated above
affpt1 = ma.' * refpt1.';
affpt2 = ma.' * refpt2.';
affpt3 = ma.' * refpt3.';
affpt4 = ma.' * refpt4.';
affpt5 = ma.' * refpt5.';
affpt6 = ma.' * refpt6.';
affpt7 = ma.' * refpt7.';
affpt8 = ma.' * refpt8.';
affpt9 = ma.' * refpt9.';

hompt1 = mh.' * refpt1.';
hompt2 = mh.' * refpt2.';
hompt3 = mh.' * refpt3.';
hompt4 = mh.' * refpt4.';
hompt5 = mh.' * refpt5.';
hompt6 = mh.' * refpt6.';
hompt7 = mh.' * refpt7.';
hompt8 = mh.' * refpt8.';
hompt9 = mh.' * refpt9.';

hompt1 = hompt1 ./ hompt1(3);

```

```

hompt2 = hompt2 ./ hompt2(3);
hompt3 = hompt3 ./ hompt3(3);
hompt4 = hompt4 ./ hompt4(3);
hompt5 = hompt5 ./ hompt5(3);
hompt6 = hompt6 ./ hompt6(3);
hompt7 = hompt7 ./ hompt7(3);
hompt8 = hompt8 ./ hompt8(3);
hompt9 = hompt9 ./ hompt9(3);

% average the transformed points and assemble lists
avgpt1 = (affpt1+hompt1) ./ 2;
avgpt2 = (affpt2+hompt2) ./ 2;
avgpt3 = (affpt3+hompt3) ./ 2;
avgpt4 = (affpt4+hompt4) ./ 2;
avgpt5 = (affpt5+hompt5) ./ 2;
avgpt6 = (affpt6+hompt6) ./ 2;
avgpt7 = (affpt7+hompt7) ./ 2;
avgpt8 = (affpt8+hompt8) ./ 2;
avgpt9 = (affpt9+hompt9) ./ 2;

refpts = [refpt1(1) refpt1(2); refpt2(1) refpt2(2); refpt3(1) refpt3(2); refpt4(1) refpt4(2);
refpt5(1) refpt5(2); refpt6(1) refpt6(2); refpt7(1) refpt7(2); refpt8(1) refpt8(2); refpt9(1)
refpt9(2)];
avgpts = [avgpt1(1) avgpt1(2); avgpt2(1) avgpt2(2); avgpt3(1) avgpt3(2); avgpt4(1) avgpt4(2);
avgpt5(1) avgpt5(2); avgpt6(1) avgpt6(2); avgpt7(1) avgpt7(2); avgpt8(1) avgpt8(2); avgpt9(1)
avgpt9(2)];

% finally compute the new, weakened homography
M = homography_svd_mat(refpts, avgpts);

```

medcut_to_16pts.m

```
function [simppts1, simppts2] = medcut_to_16pts(points1, points2)
```

```

n_pts_input = size(points1, 1);

rects = zeros(16, 4);
avgpts = (points1 + points2) ./ 2;
apmins = min(avgpts);
apmaxs = max(avgpts);
rects(1, 1) = apmins(1);
rects(1, 2) = apmins(2);
rects(1, 3) = apmaxs(1);
rects(1, 4) = apmaxs(2);

% Scan the rectangles and perform the cuts
for rect_bit_level = 0:3
    rect_bit_value = 2^rect_bit_level;

```

```

for i_rect = 1:rect_bit_value
    cur_rect_min_x = rects(i_rect, 1);
    cur_rect_min_y = rects(i_rect, 2);
    cur_rect_max_x = rects(i_rect, 3);
    cur_rect_max_y = rects(i_rect, 4);
    pt_in_cur_rect_flags = zeros(n_pts_input, 1);
    for i_point = 1:n_pts_input
        % We consider boundaries as inclusive, to avoid the situation of a
rectangle containing no points
        if cur_rect_min_x <= avgpts(i_point, 1) && avgpts(i_point, 1) <=
cur_rect_max_x && cur_rect_min_y <= avgpts(i_point, 2) && avgpts(i_point, 2) <= cur_rect_max_y
            pt_in_cur_rect_flags(i_point) = 1;
        end
    end
    idxs_in_cur_rect = find(pt_in_cur_rect_flags);
    pts_in_cur_rect = avgpts(idxs_in_cur_rect, :);
    cur_meds = median(pts_in_cur_rect);
    cut_x_axis_flag = ((cur_rect_max_x - cur_rect_min_x) > (cur_rect_max_y -
cur_rect_min_y));
    if (cur_rect_max_x - cur_rect_min_x) == (cur_rect_max_y - cur_rect_min_y)
        % alternate between horizontal and vertical cuts in the event of a tie
        cut_x_axis_flag = (mod(rect_bit_level, 2) == 1);
    end

    % The rectangle of lower coordinate value overwrites rects(i_rect, :),
    % while the rectangle of higher value is placed in
rects(i_rect+rect_bit_value, :).
    % Each pass doubles the number of rectangles stored in rects, until it is
full.

    if cut_x_axis_flag
        rects(i_rect,
                3) = cur_meds(1);

        rects(i_rect+rect_bit_value, 1) = cur_meds(1);
        rects(i_rect+rect_bit_value, 2) = cur_rect_min_y;
    else
        rects(i_rect,
                4) = cur_meds(2);

        rects(i_rect+rect_bit_value, 1) = cur_rect_min_x;
        rects(i_rect+rect_bit_value, 2) = cur_meds(2);
    end
    end
    rects(i_rect+rect_bit_value, 3) = cur_rect_max_x;
    rects(i_rect+rect_bit_value, 4) = cur_rect_max_y;
end
end

% Rescan the final rectangles and assign bucket flags
pt_in_bucket_flags = zeros(n_pts_input, 16);
for i_rect = 1:16

```

```

cur_rect_min_x = rects(i_rect, 1);
cur_rect_min_y = rects(i_rect, 2);
cur_rect_max_x = rects(i_rect, 3);
cur_rect_max_y = rects(i_rect, 4);

for i_point = 1:n_pts_input
    if cur_rect_min_x <= avgpts(i_point, 1) && avgpts(i_point, 1) <=
cur_rect_max_x && cur_rect_min_y <= avgpts(i_point, 2) && avgpts(i_point, 2) <= cur_rect_max_y
        pt_in_bucket_flags(i_point, i_rect) = 1;
    end
end

end

% If a point lies on a boundary between buckets, split its weight accordingly
dupchk = sum(pt_in_bucket_flags, 2);
for i_point = 1:n_pts_input
    pt_in_bucket_flags(i_point, :) = pt_in_bucket_flags(i_point, :) ./ dupchk(i_point);
end

end

% Average the original points within each bucket
simppts1 = zeros(16, 2);
simppts2 = zeros(16, 2);
for i_rect = 1:16
    simppts1(i_rect, 1) = sum(points1(:,1) .* pt_in_bucket_flags(:, i_rect)) /
sum(pt_in_bucket_flags(:, i_rect));
    simppts1(i_rect, 2) = sum(points1(:,2) .* pt_in_bucket_flags(:, i_rect)) /
sum(pt_in_bucket_flags(:, i_rect));
    simppts2(i_rect, 1) = sum(points2(:,1) .* pt_in_bucket_flags(:, i_rect)) /
sum(pt_in_bucket_flags(:, i_rect));
    simppts2(i_rect, 2) = sum(points2(:,2) .* pt_in_bucket_flags(:, i_rect)) /
sum(pt_in_bucket_flags(:, i_rect));
end
end % function

```

performFlowStitching.m (invoked from the MATLAB command line)

```

% Make a panorama, using images from a folder IMPORTANT:
% Image files must be named so that their lexicographical order
% represents a left-to-right progression.
function [ unwrapped_stitch ] = performFlowStitching( folder )
% - START OF FUNCTION - %

disp('** Part 0: Initial reading of images **');

imgfiles = dir(fullfile(folder, '*.png'));
n_images = size(imgfiles, 1);
fprintf('Found %i images\n', n_images);
if n_images < 3
    % XXX: copied from performFolderMod.m

```

```

    % This script could be made to work with only two images,
    % but we don't have a camera with sufficient FOV.
    error('Need at least three images (in PNG format)');
end

% Obtain the heights and widths, and create a cell array that holds the images
images = cell(n_images, 1); % could initialize empty and then use end+1 below...
heights = zeros(n_images,1);
widths = zeros(n_images,1);
for i = 1:n_images
    tmp_img = imread(fullfile(folder, imgfiles(i).name));

    if ndims(tmp_img) < 3
        fprintf('Problem with image no. %i: ', i);
        error('One or more images are grayscale; current implementation requires RGB');
    end

    heights(i) = size(tmp_img, 1);
    widths(i) = size(tmp_img, 2);

    images{i} = double(tmp_img)./255;
end

if max(heights)>min(heights)
    error('All images must have same height');
end
if max(widths)>min(widths)
    error('Current implementation requires all images to have same width');
end

hght = heights(1);
wdth = widths(1);

% High-pass-filter the images
disp('** Part 1: High-pass filtering temp copy **')
%
ims_hp = images;
midGray = 0.5;
for i = 1:n_images
    % custom_imgaussfilt computes a blur radius on its own
    % we add midGray to prevent potential domain problems with rgb2gray
    ims_hp{i} = ims_hp{i} - custom_imgaussfilt(ims_hp{i}) + midGray;

    fprintf('* finished with image %i\n', i);
end

% Compute brute-force horizontal alignment (on the high-passed images)
disp('** Part 2: Brute-force pre-alignment **');
```

```

% min_overlap needs to be larger than patchsize and
% (more restrictively) topwsz, below
min_overlap = 30;
max_overlap = floor(wdth/2);
best_overlaps = zeros(n_images,1);
for i = 1:n_images
    cur_best_overlap = min_overlap;
    cur_best_rmsdiff = inf;
    for ov = min_overlap:max_overlap
        curr_right = ims_hp{i};
        curr_right = curr_right(:,wdth-ov+1:wdth,:);
        if i == n_images
            next_left = ims_hp{1};
        else
            next_left = ims_hp{i+1};
        end
        next_left = next_left(:,1:ov,:);

        curr_right_red = curr_right(:, :, 1);
        curr_right_grn = curr_right(:, :, 2);
        curr_right_blu = curr_right(:, :, 3);
        curr_right_lum = rgb2gray(curr_right);
        next_left_red = next_left(:, :, 1);
        next_left_grn = next_left(:, :, 2);
        next_left_blu = next_left(:, :, 3);
        next_left_lum = rgb2gray(next_left);

        rmsdiff_red = norm(curr_right_red-next_left_red, 'fro')/sqrt(ov*hght);
        rmsdiff_grn = norm(curr_right_grn-next_left_grn, 'fro')/sqrt(ov*hght);
        rmsdiff_blu = norm(curr_right_blu-next_left_blu, 'fro')/sqrt(ov*hght);
        rmsdiff_lum = norm(curr_right_lum-next_left_lum, 'fro')/sqrt(ov*hght);

        cur_rmsdiff = rmsdiff_red + rmsdiff_grn + rmsdiff_blu + rmsdiff_lum;
        % XXX: should exclude blue from the max here? Or does incorporating
        % luma make this a non-issue? (also need to check RGB vs. BGR order)
        cur_rmsdiff = cur_rmsdiff + max([rmsdiff_red rmsdiff_grn rmsdiff_blu rmsdiff_lum]);

        if cur_rmsdiff < cur_best_rmsdiff
            cur_best_overlap = ov;
            cur_best_rmsdiff = cur_rmsdiff;
        end
    end
end

best_overlaps(i) = cur_best_overlap;

fprintf('* chose raw overlap of %ipx between image %i and next\n', cur_best_overlap, i);
end

```

```

disp('** Part 2.5: Harmonizing overlap values **');
best_overlaps_temp = best_overlaps;
% better to err in favor of more overlap rather than less
%fudge_target = max(best_overlaps);
fudge_target = (mean(best_overlaps)+median(best_overlaps)+2*max(best_overlaps)+max_overlap)/5;
for i = 1:n_images
    best_overlaps_temp(i) = round(0.435*best_overlaps_temp(i)+0.565*fudge_target);

    fprintf('* adjusted overlap of %ipx between image %i and next\n', best_overlaps_temp(i), i);
end
best_overlaps = best_overlaps_temp;

% Compute optical flow on the overlapping regions
disp('** Part 3: Optical flow and warping ** (be patient!) **');
%
overlap_regions = images;
for i = 1:n_images
    curr_right = images{i};
    curr_right = curr_right(:,width-best_overlaps(i)+1:width,:);
    if i == n_images
        next_left = images{1};
    else
        next_left = images{i+1};
    end
    next_left = next_left(:,1:best_overlaps(i),:);

    % iat_dense_sift internally converts RGB images to grayscale
    % by averaging the channels. We can do better...
    curr_right_red = curr_right(:,:,1);
    next_left_red = next_left(:,:,1);
    curr_right_grn = curr_right(:,:,2);
    next_left_grn = next_left(:,:,2);
    curr_right_blu = curr_right(:,:,3);
    next_left_blu = next_left(:,:,3);
    curr_right_gray = rgb2gray(curr_right);
    next_left_gray = rgb2gray(next_left);

    % actual I.A.T. stuff begins here
    % NB: patchsize must be less than min_overlap above
    % this patchsize matches that in the tutorial
    patchsize = 8;
    % full resolution
    gridspacing = 1;

    fprintf('(image %i) about to do iat_dense_sift...\n', i);
    disp('...on curr_right_red');
    curr_right_red_sift = iat_dense_sift(curr_right_red, patchsize, gridspacing);
    disp('...on next_left_red');

```



```

next_left_red_sift = iat_dense_sift(next_left_red, patchsize, gridspacing);
disp('...on curr_right_grn');
curr_right_grn_sift = iat_dense_sift(curr_right_grn, patchsize, gridspacing);
disp('...on next_left_grn');
next_left_grn_sift = iat_dense_sift(next_left_grn, patchsize, gridspacing);
disp('...on curr_right_blu');
curr_right_blu_sift = iat_dense_sift(curr_right_blu, patchsize, gridspacing);
disp('...on next_left_blu');
next_left_blu_sift = iat_dense_sift(next_left_blu, patchsize, gridspacing);
disp('...on curr_right_gray');
curr_right_lum_sift = iat_dense_sift(curr_right_gray, patchsize, gridspacing);
disp('...on next_left_gray');
next_left_lum_sift = iat_dense_sift(next_left_gray, patchsize, gridspacing);

%
SIFTflowpara.alpha = 2.23;
SIFTflowpara.d = 63;
SIFTflowpara.gamma = 0.00415;
SIFTflowpara.nlevels = 5;
SIFTflowpara.wsize = 7;
SIFTflowpara.topwsize = 26;
SIFTflowpara.nIterations = 56;
SIFTflowpara.nTopIterations = 150;
% XXX: Are both forward & reverse computations necessary (or is one flow
% guaranteed to be the inverse of the other? - in practice not true)
fprintf(' (image %i) iat_SIFTflow working on rev red:\n', i);
[vx_rev_red, vy_rev_red, ~] = iat_SIFTflow(curr_right_red_sift, next_left_red_sift,
SIFTflowpara);
fprintf(' (image %i) iat_SIFTflow working on fwd red:\n', i);
[vx_fwd_red, vy_fwd_red, ~] = iat_SIFTflow(next_left_red_sift, curr_right_red_sift,
SIFTflowpara);
fprintf(' (image %i) iat_SIFTflow working on rev green:\n', i);
[vx_rev_grn, vy_rev_grn, ~] = iat_SIFTflow(curr_right_grn_sift, next_left_grn_sift,
SIFTflowpara);
fprintf(' (image %i) iat_SIFTflow working on fwd green:\n', i);
[vx_fwd_grn, vy_fwd_grn, ~] = iat_SIFTflow(next_left_grn_sift, curr_right_grn_sift,
SIFTflowpara);
fprintf(' (image %i) iat_SIFTflow working on rev blue:\n', i);
[vx_rev_blu, vy_rev_blu, ~] = iat_SIFTflow(curr_right_blu_sift, next_left_blu_sift,
SIFTflowpara);
fprintf(' (image %i) iat_SIFTflow working on fwd blue:\n', i);
[vx_fwd_blu, vy_fwd_blu, ~] = iat_SIFTflow(next_left_blu_sift, curr_right_blu_sift,
SIFTflowpara);
fprintf(' (image %i) iat_SIFTflow working on rev luma:\n', i);
[vx_rev_lum, vy_rev_lum, ~] = iat_SIFTflow(curr_right_lum_sift, next_left_lum_sift,
SIFTflowpara);
fprintf(' (image %i) iat_SIFTflow working on fwd luma:\n', i);
[vx_fwd_lum, vy_fwd_lum, ~] = iat_SIFTflow(next_left_lum_sift, curr_right_lum_sift,

```

```

SIFTflowpara);

vx_rev = median(cat(3, vx_rev_red, vx_rev_grn, vx_rev_blu, vx_rev_lum), 3);
vx_fwd = median(cat(3, vx_fwd_red, vx_fwd_grn, vx_fwd_blu, vx_fwd_lum), 3);
vy_rev = median(cat(3, vy_rev_red, vy_rev_grn, vy_rev_blu, vy_rev_lum), 3);
vy_fwd = median(cat(3, vy_fwd_red, vy_fwd_grn, vy_fwd_blu, vy_fwd_lum), 3);

% Pad vector fields to the size of the original image. (todo explain more)
vx_rev_padded = padarray(vx_rev, [patchsize/2 patchsize/2], 'replicate');
vy_rev_padded = padarray(vy_rev, [patchsize/2 patchsize/2], 'replicate');
vx_fwd_padded = padarray(vx_fwd, [patchsize/2 patchsize/2], 'replicate');
vy_fwd_padded = padarray(vy_fwd, [patchsize/2 patchsize/2], 'replicate');

% At the left edge of each overlap, warp the next image all the way back to meet the current.
% At the right edge, warp the current image all the way forward to meet the next.
% Smooth transition in between.
blend_weight_rev = zeros(1, best_overlaps(i));
blend_weight_fwd = zeros(1, best_overlaps(i));
for j_col = 2:best_overlaps(i)
    cur_weight = (j_col-1)/(best_overlaps(i)-1);
    cur_weight = sin(cur_weight*pi/2);
    cur_weight = cur_weight * cur_weight;

    blend_weight_fwd(j_col) = cur_weight;
    blend_weight_rev(best_overlaps(i)-j_col+1) = cur_weight;
end
blend_weight_rev = repmat(blend_weight_rev, hght, 1);
blend_weight_fwd = repmat(blend_weight_fwd, hght, 1);

vx_rev_wtd = vx_rev_padded .* blend_weight_rev;
vy_rev_wtd = vy_rev_padded .* blend_weight_rev;
vx_fwd_wtd = vx_fwd_padded .* blend_weight_fwd;
vy_fwd_wtd = vy_fwd_padded .* blend_weight_fwd;

[warpedINL, suppINL] = iat_pixel_warping( next_left, vx_rev_wtd, vy_rev_wtd );
[warpedICR, suppICR] = iat_pixel_warping( curr_right, vx_fwd_wtd, vy_fwd_wtd );

overlap_regions{i} = average_by_support(warpedINL, suppINL, warpedICR, suppICR);

fprintf('* finished with image %i\n', i);
end

% Build a cell array that contains the non-overlapping regions of each image
disp('** Part 4: Selecting non-overlapping regions ** (almost done...) **');
%
nonoverlap_regions = images;
has_nonoverlapping_cols = ones(n_images,1);
for i=1:n_images

```

```

overlap_right = best_overlaps(i);
if i==1
    overlap_left = best_overlaps(n_images);
else
    overlap_left = best_overlaps(i-1);
end

if overlap_right+overlap_left < width
    im_cur = images{i};
    nonoverlap_regions{i} = im_cur(:,overlap_left+1:width-overlap_right,:);
else
    fprintf('Uh-oh, no non-overlapping columns from image %i!\n', i);
    has_nonoverlapping_cols(i) = 0;
end

fprintf('* finished with image %i\n', i);
end

% Assemble the overlap and non-overlap regions
disp('** Part 5: Assembling regions **');
%
unwrapped_stitch = overlap_regions{n_images};
for i=1:n_images
    if i > 1
        unwrapped_stitch = cat(2, unwrapped_stitch, overlap_regions{i-1});
    end
    if has_nonoverlapping_cols(i) == 1
        unwrapped_stitch = cat(2, unwrapped_stitch, nonoverlap_regions{i});
    end

    fprintf('* finished with image %i\n', i);
end

% swidth = size(unwrapped_stitch, 2);
% todo later: circle warp (in a separate function?)

disp('done: About to save image');
imwrite(unwrapped_stitch, strcat('flowStitchResult-', folder, '.png'));

% - END OF FUNCTION - %
end

```

custom_imgaussfilt.m

```

% Wrapper for the Gaussian blur from Image Processing Toolbox
% Automatically computes a blur radius that is useful for high-pass filtering
% and makes some compromises with respect to boundary conditions
%

```

```

% img must be of class double,
% but both grayscale and RGB are supported
function [ im_blurred ] = custom_imgaussfilt( img )

nd = ndims(img);

hght = size(img, 1);
wdth = size(img, 2);

% empirically-chosen formula: geometric-harmonic mean
hm = 8;
gm = max(hght,wdth);
if gm <= hm
    error('custom_imgaussfilt: Image must be larger than 8px');
end
for i_safety = 1:32
    if gm-hm < 0.7
        break;
    else
        newhm = 2/(1/hm+1/gm);
        newgm = sqrt(hm*gm);
        hm = newhm;
        gm = newgm;
    end
end

blur_rad = hm;

% XXX: In principle, the circular BC does not make sense for this application.
% However, it best satisfies the def'n of a freq-domain low-pass filter...
cblur = imgaussfilt(img, blur_rad, 'Padding', 'circular');
rblur = imgaussfilt(img, blur_rad, 'Padding', 'replicate');
sblur = imgaussfilt(img, blur_rad, 'Padding', 'symmetric');

medstk = median(cat(nd+1, cblur, rblur, sblur), nd+1);

im_blurred = (0.19*cblur + 0.38*rblur + 0.38*sblur + 0.05*medstk);

end %function

```