Senscape: modeling and presentation of uncertainty in fused sensor data live image streams

Henry Dietz and Paul Eberhart; University of Kentucky; Lexington, Kentucky

Abstract

Fusion of data from multiple sensors is a difficult problem. Most recent work centers on techniques that allow image data from multiple similar sources to be aligned and used to improve apparent image quality or field of view. In contrast, the current work centers on modeling and representation of uncertainty in real-time fusion of data from fundamentally dissimilar sensors. Where multiple sensors of differing type, resolution, field of view, and sample rate are providing scene data, the proposed scheme directly models uncertainty and provides an intuitive mechanism for visually representing the time-varying level of confidence in the correctness of fused sensor data producing a live image stream.

Introduction

Most often, sensor fusion to produce an image involves merging spatially-aligned and overlapping data from multiple imaging sensors. For example, video cameras often employed a beam splitter to send each of red, green, and blue light to a different sensor, in which case combining the data to produce a full-color image was trivial. It is nearly as straightforward to combine image data for the overlapping portion of a scene sensed by two or more separate image sensors. The difficulty lies in combining sensor data that is not of the same "shape."

An obvious distinction is between imaging sensors and sensors that sample point properties. Imaging sensors include conventional visible-light and multispectral CCD and CMOS cameras, micro-bolometers and other thermal imagers, time-of-flight depth cameras, etc. They report data for each sample in the form of an array of property values. In contrast, point property sensors report data that describes properties sampled at a single, often imprecisely defined, point in space. For example, ultrasonic and laser distance meters, infrared thermometers, accelerometers, gyroscopes, magnetometers, etc. report data that is not inherently structured as an image. More generally, sensor data "shape" can differ in at least:

- Dimensionality, from point sensors to imaging sensors returning data arrays with 1, 2, 3, or more dimensions
- Spatial field of view, resolution, and accuracy
- Tonal resolution and dynamic range
- Temporal properties including sample rate, measurement intervals, synchronization, and latency in reporting
- Point-of-view differences and misalignments, including parallax and occlusion

To create a real-time image stream that integrates data from a wide variety of different types of sensors is a complex problem. The approach taken in the current work involves use of an uncertainty-aware painting algorithm.

Painting algorithms

A painting algorithm is a method by which the next image in the output stream is created by incrementally "painting" new data into the appropriate portions of an image. Because updating the image to reflect the contribution from a sensor can be performed using an arbitrary function of the new data and the display history, painting algorithms easily handle integration of image and point sensor data, etc. However, spatio-temporal alignment of data from different sensors is critical in this painting process.

There are $O(2^n)$ alignments between n sensors, but this complexity can be reduced to $O(n)$ alignment problems by performing all alignments relative to one reference image, which one might think of as being the "canvas" for the painting. If the canvas is moved, all history of the painting moves with it. Typically, the reference canvas either will be defined using data from the imaging sensor with the widest field of view and highest resolution or it will be synthesized as the image from an idealized sensor, often with an even wider field of view. When data from a sensor is collected, it is transformed into the space of the canvas. It is in the space of the canvas that old and new data from a sensor are reconciled. Subsequently applying the updated canvas-space data from all sensors to construct the next output image is relatively straightforward.

As a simple example, imagine a cell phone with two cameras, one with a wide field of view, the other with a telephoto field of view. A good choice of reference canvas would be the view expected to be spanned by the wide camera as it moves over time, but interpolated to the angular pixel density of the telephoto. The higher-resolution, but narrower view, image data from the telephoto camera would be painted into its own history data structure shaped like the canvas: its **property map**. When the cell phone is pointed a bit to the right, the old data is shifted and new data is merged with the old, updating the property maps for both the wide and telephoto sensors. The rendered result is an aligned higher resolution patch within the wider stitched view. This imaginary sequence is depicted in Figure 1.

In a **Senscape**, each sensor will have its own property map. The painting algorithm is essentially:

1. Determine any transformations of the canvas and update the existing data in all property maps accordingly
2. Paint new data from each sensor into its property map
3. Paint data from the property maps into an image to render the next frame in the live image stream

Uncertainty (or confidence)

Figure 1's example treated all sensor data and the fusion process as perfect, but there was still complete uncertainty in areas (shown in gray) that each sensor had not yet captured. In gen-

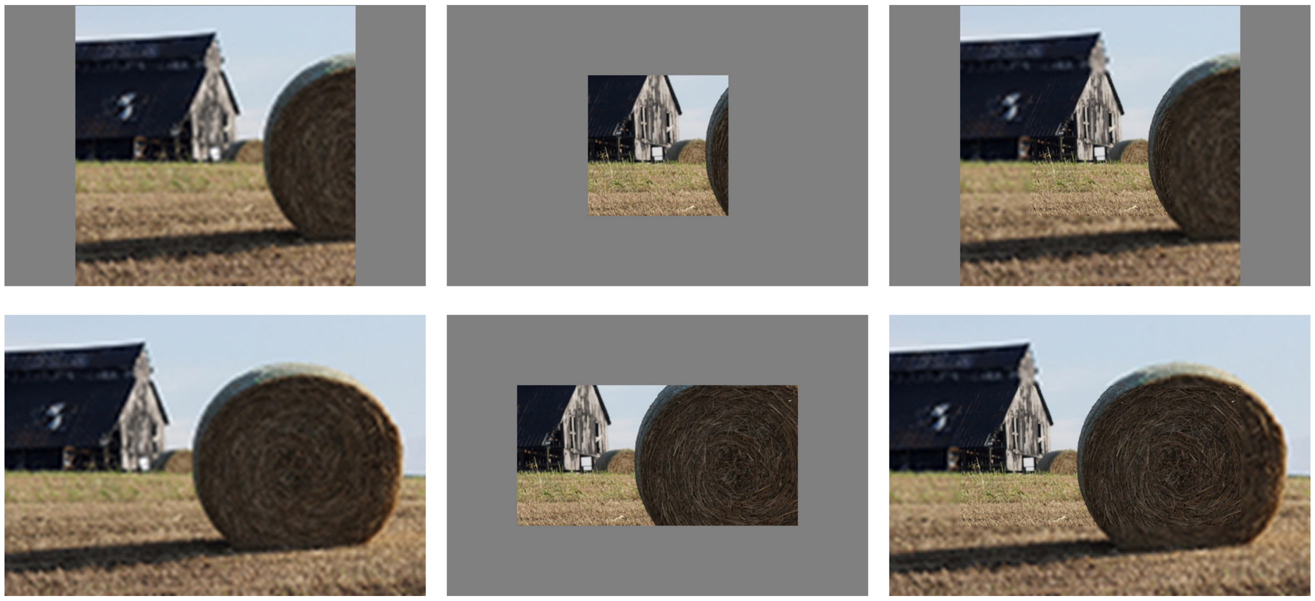

Figure 1. Painting sequence, left-to-right: wide lens view, tele view in canvas shape, combined; second row is result after cameras are pointed to the right

eral, **each sensor value read not only provides data about some property, but potentially alters our confidence about the scene appearance**.

Step one above is largely about alignment and tracking. Although motion sensors can be used, in most cases, alignment will need to be done based on matching image features in before and after versions of some sensor's property map – typically, the map that defines the canvas. Either way, this alignment is unlikely to be perfect. Aside from precision and accuracy issues, features used for alignment might become occluded by the camera's change in point of view or the scene itself might change such that the features disappear or move relative to the rest of the scene. If there is too much difference between the old and new canvas maps, the old data may no longer be valid. Ambiguity in alignment, and aging of the data, reduces confidence in the map's values.

The second step above, painting new sensor data into its property map, can alter both property values and confidence. Not all sensor data is equally trustworthy; each value reported by the sensor has some level of confidence that it is correct. In the current work, we argue that **confidence in a value is itself a potentially important property, and it also should be in a property map**. How can one combine disparate old and new values without tracking a value confidence metric and applying some type of Bayesian[1][2] weighting formula?

The final rendering, described in the third step, also is heavily dependent on confidence values. Confidence differences are needed to resolve rendering choices that pit values from one property map against those from another for priority in use of a shared display attribute. In Figure 1, the higher-resolution data is preferred in the fused image because it carries higher confidence.

Despite this, most sensor fusion attempts to hide missing or low-quality data in the images rendered. The goal is to produce a *credible* image rather than a *correct* one. For example, most cell phone cameras now have modes in which they will guess at depths in the scene in order to computationally render "bokeh" – smooth out-of-focus effects. It would spoil the effect to clearly indicate just how uncertain some of the depth estimates used were, but placing trust in incorrect, yet credible, data can be harmful.

Consider a sensor detecting a green traffic light. If the green light was actually for a different lane, but was incorrectly matched to the fixture hanging over your lane, knowing that the green light image had very low confidence could prevent an accident. The same is true if a sensor glitch caused no further images to be sent from the sensor that originally saw a green light for your lane; how confident are you that the light still displayed as green actually is still green 30 seconds after that image was captured? **Confidence itself should be a displayable property**, and conveying this additional information need not make the fused data more difficult to understand[3].

The objective for the current work is thus to investigate simple models for computationally tracking, and appropriately displaying in a meaningful way, the confidence of fused sensor data expressed as a real-time image stream. Four very different multisensor imaging systems are discussed: FireScape, NodeScape, KVIRP, and Wakam. The last two are open source hardware and software systems created as testbeds for the current work, and easily can be replicated by others at a cost under $200 each.

FireScape

The current notion of using a painting algorithm with explicit modeling of uncertainty in creating live image displays for fused sensor data was initially formulated as part of the 2006-2007 FireScape project. The project began when Bud Meyer approached Professors Bill Dieter and Henry Dietz with the challenge to invent an inexpensive device that could provide hands-free thermal imaging to guide firefighters as they navigate a burning building. Dieter led the project, with Dietz primarily working on the sensor fusion and display processing for a face-mask-mounted live display. Although various subsystems were prototyped, the complete FireScape system was never built and it was not discussed in formal publications until a survey paper about multicameras at EI2018[4].

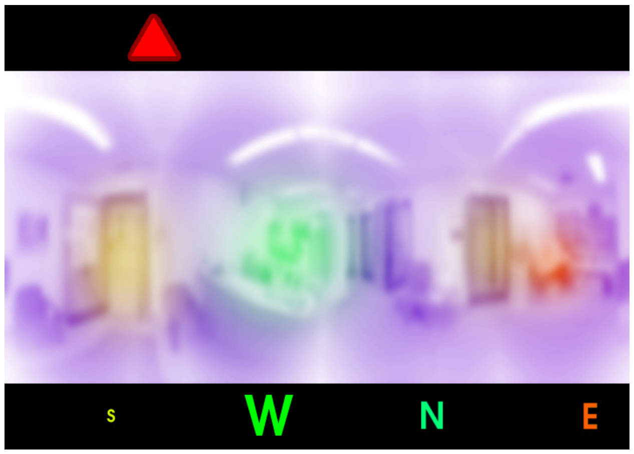

Figure 2. FireScape simulated image

The aspect of FireScape that is most relevant to the current work is the construction of the simulated display in Figure 2. This provides a 360° monochromatic base view made by fusing data from three cheap, 320×240 resolution, webcams fitted with door peepholes (used as fire-rated, easily replaceable, fish-eye lenses). The stitching was done using a fixed mapping table, which showed no significant misalignments at the low resolution needed for the mask-mounted display. Diffuse, relatively wide-angle IR data from multiple single-point sensors is painted on that gray scale image with confidence indicated by saturation and lack of a magenta tint. Temperatures run from blue to red, but are nonlinearly mapped to use green to emphasize where human body heat may have been detected. This simulated image assumed multiple fast point sensors, so confidence of thermal data decayed primarily by misalignment rather than aging. The S-W-N-E at the bottom indicate the compass directions, with approximate hottest temperature in each direction indicated by color and approximate distance to nearest object (from a sonar sensor) indicated by size of the letter. The red mark at the top indicates the oxygen supply is running low and points in the approximate direction from which the room was entered.

NodeScape

In 2012, NodeScape applied the concept of painting uncertainty-sensitive point sensor data onto a base image for the purpose of monitoring activity of nodes within Linux PC cluster supercomputers. An example NodeScape display image used for remote monitoring is shown in Figure 3; alternatively, the fused data can be projected onto the front panels of the physical nodes.

It should come as no surprise that there are a variety of physical and logical sensors within a typical PC node: ambient, processor, and GPU temperature; fan speeds; load average; network use; etc. One also can execute arbitrary code to synthesize property values that are more complex, such as ratios between temperature and load average. These standard or synthesized properties are monitored by a daemon process on each node called `epacsedon`, which collects sensor data updates and sends UDP messages to a machine running `nodescape`, which fuses and displays the complete status of the cluster. It is less obvious how this status data becomes an image.

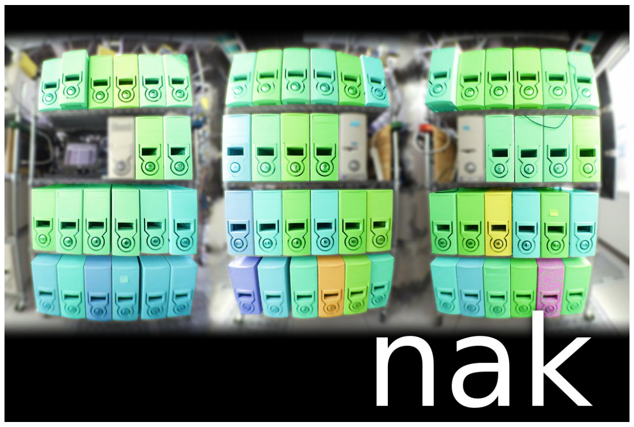

Figure 3. Nodescape status image for nak

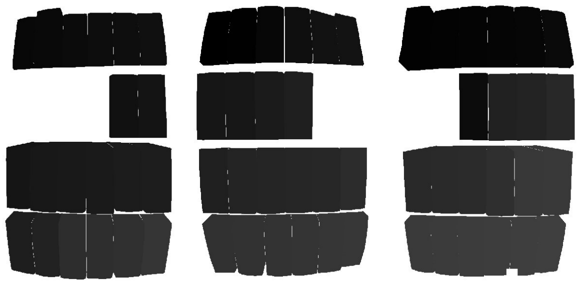

Figure 4. Nodescape painting key image for nak

Physical node placement often is guided by optimizing wiring complexity with various constraints on rack capacity, power distribution, network switches, etc. Finding the physical node that a sensor detected needs servicing often is not straightforward. Neither is it easy to recognize patterns, such as a row of nodes all running a little hot. Thus, sensor data is painted onto a reference image derived from a photograph of the machine.

The image configuration process begins by photographing the physical machine and editing the image to add labels, etc. If status will be projected on the nodes themselves, all areas that are not nodes should be black. Additionally, a key image must be constructed to identify which pixels correspond to which nodes, and shown in Figure 4. The key is completely static for NodeScape, but it is easy to imagine dynamic use of this mechanism. For example, a neural network could create a key identifying individuals in a scene so that point data from their personal electronic identity could be painted on their images.

The tinting of the reference image is by default done with colors ranging from blue to red so that higher values are mapped into longer wavelengths. This color orientation matches the common case that a high sensor reading indicates a problem. The minimum and maximum property values are learned by `nodescape` rather than specified a priori. Initially, the minimum and maximum values are the same, and a green tint is applied.

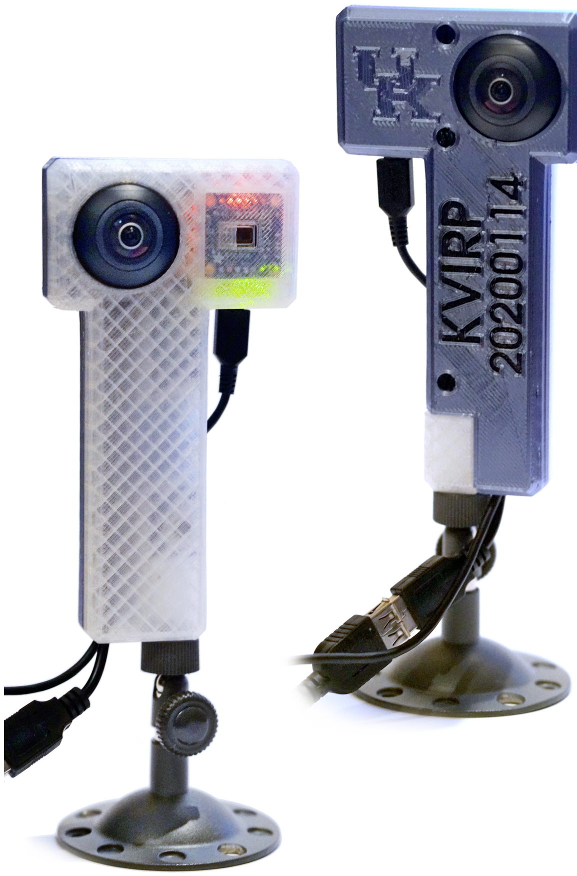

Figure 5. *KVIRP 20200114, front (left) and rear (right) views*

NodeScape does not maintain a property map for each point sensor, but as `nodescape` receives each UDP packet, it appropriately updates the current value of the property for that node and records when the update occurred. In addition to problems like overheating, cluster nodes often fail because their network connection is interrupted; however, UDP messages also could be dropped due to heavy traffic, so a small period without updates does not necessarily indicate a significant problem. Thus, instead of a subtle indication of data aging, the node image is dithered with pixels tinted magenta, with the percentage of tinted pixels slowly increasing with age of the latest update from that node. The dither is kept to less than 100% and magenta is a color outside the spectrum used to represent normal values, so both the age and the last known value are easily understood in a glance. For example, in Figure 3, it is still apparent that the mostly-magenta node was formerly green.

KVIRP

KVIRP, pronounced kay-verp, stands for Kentucky's Visual / Infra Red Painter. Front and rear views of KVIRP 20200114 are shown in Figure 5. It is an open-source hand-held camera system we created to collect 360° visible-light color video along with low-resolution, narrower-angle, thermal images. The goal is to use a confidence-driven painting algorithm to impose thermal data on the 360° degree view.

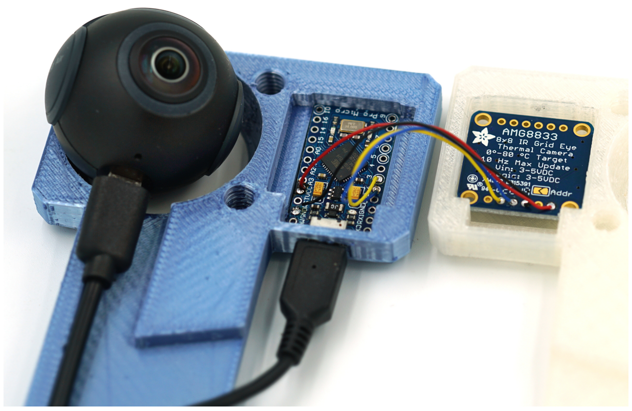

Figure 6. *KVIRP's Insta360 Air, controller, and thermal imager*

The full-color canvas for sensor fusion is defined as the result of fusing image data from a pair of visible-light sensors. An Insta360 Air[5] dual-camera module is mounted in KVIRP so that the component cameras are front and rear facing. Each component camera has an $f/2.4$ fisheye lens with a field of view significantly exceeding 180° degrees. A pair of circular images, one per component camera, are captured and transmitted via USB at up to 30FPS as a single JPEG image consisting of up to 3008×1504 pixels. The image quality and data rate are impressive for such a small and inexpensive (under \$100) camera module.

The Insta360 Air module is less than 40mm deep, which allows even objects placed fairly close to the side of the module to be recorded by both front and rear cameras, but this also makes it difficult to mount the camera without the mount obscuring some portion of the field of view. The 3D-printed housing for KVIRP (and Wakam, described below) allows an unobstructed overlap between the front and rear fisheye views.

The current version of `kvirp.cpp` uses OpenCV to decode the fisheye video stream, as well as to display video in real time. Algorithms for high-quality stitching of fisheye images have been extensively studied. In the late 1990s, Panorama Tools[6] provided open-source code for transformation of fisheye projections and basic image stitching; as early as 1999[4], those algorithms were applied in multiple systems stitching image pairs captured by fisheye cameras rigidly attached back-to-back. There is even open-source code using OpenCV for the particular case of stitching compound images captured by a two-camera module[7]. However, `kvirp.cpp` uses its own very simple stitching; the stitching is not of particularly high quality, but also **produces a confidence map**. This map is used to reduce contrast in the portions of the stitched image that are most subject to alignment errors and potentially missing image content due to parallax.

The other sensor component in KVIRP is an infra-red (thermal) imager, the hardware implementation of which is shown in Figure 6. Photons of visible light have wavelengths between roughly 350nm and 750nm. Longer wavelengths have less energy per photon – and come from objects with a lower average temperature. Near infra-red wavelengths that can be detected by CCD or CMOS cameras span from about 750nm to 1200nm, which roughly corresponds to temperatures above 300°C. To image temperatures more typical of human environments, the sensor must be

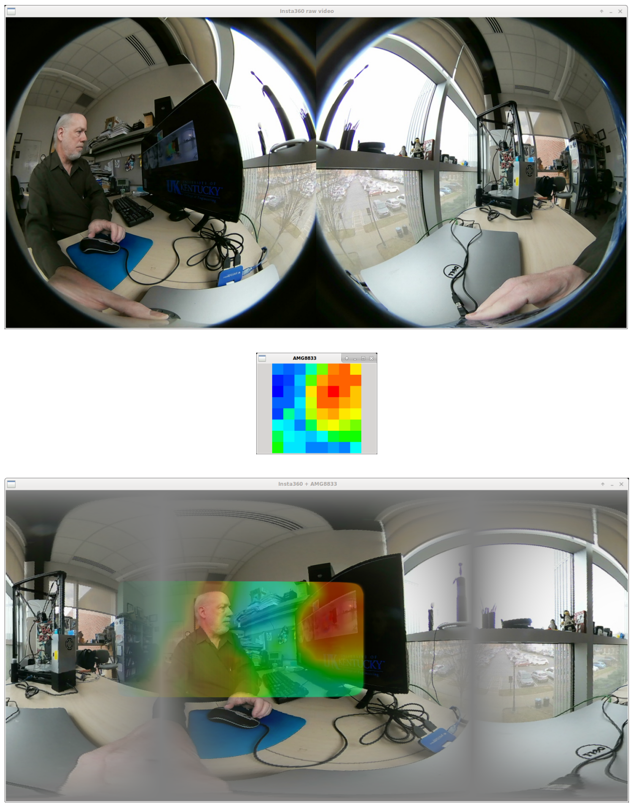

Figure 7. *KVIRP raw image data and fused result*

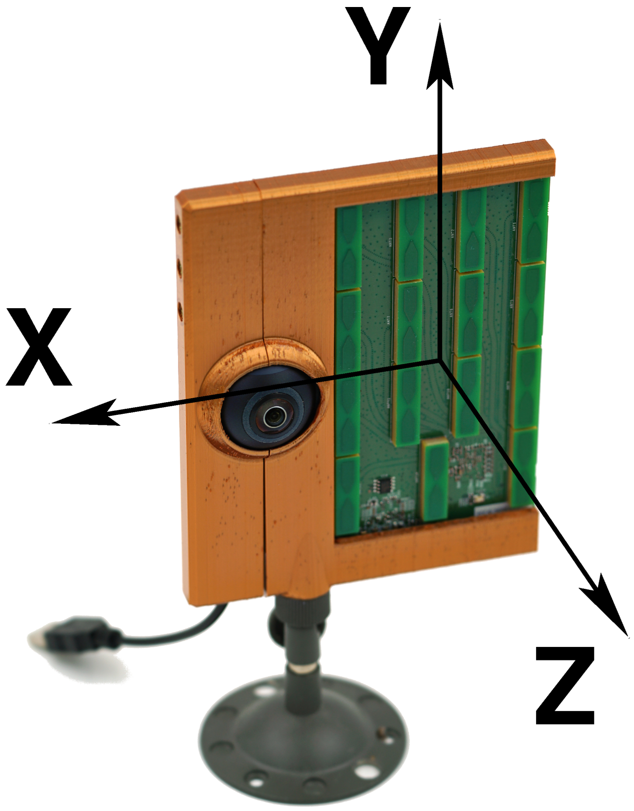

Figure 8. *Wakam: WAlabot (creator) Kentucky cAMera*

able to detect correspondingly longer wavelengths, which requires a different sensor technology and cannot use a conventional lens because glass significantly blocks IR.

Although IR imagers generally have been expensive, Adafruit's AMG8833 8x8 IR Grid Eye Thermal Camera[8] costs just $40 and can sense temperatures from 0°C to 80°C with a thermal resolution of approximately 0.25°C and absolute accuracy of +/-2.5°C. The catch is that this thermal imager contains just 64 pixels covering a field of view of approximately 60° in each of horizontal and vertical (i.e., about 7.5° per pixel) and cannot provide new sample data at a rate above 10Hz (i.e., 10FPS). As shown in Figure 6, KVIRP uses a $3 ATmega32U4 Pro Micro[9] to interface the Panasonic AMG8833 to USB and power for both boards is provided by the USB host. Using library routines to control and access the AMG8833[8], the program running on the Pro Micro simply initializes the system and then loops sampling an image and transmitting the pixel values to the host via USB.

As seen in Figure 7, the fused result is similar to a much higher-quality version of what was planned for FireScape. However, KVIRP is using the property mapping approach described in the introduction of this paper – and confidence is given its own maps. The stitched 360° image is the reference canvas, and custom-written code uses simple feature matching to determine the motion transformation to be applied at each update. The search for the best transformation could allow for motion in six dimensions – X, Y, and Z translations and roll, pitch, and yaw rotations – but is instead limited to just pitch and yaw. Just as the fisheye stitching used in KVIRP is crude, but augmented by a confidence metric, this perhaps overly-simple transformation is accompanied by computing a confidence metric to adjust the confidences in the map for the old thermal data. Very low confidence in determining the motion alignment transformation essentially causes old thermal data to be discarded.

KVIRP's visible and thermal sensors are handled by two separate processes that communicate through a shared-memory data structure. This approach reconciles different, even dynamically variable, sample rates for the Insta360 Air and thermal imager. The fused display stream is created by confidence-weighted tinting of the (color) stitched image, with the temperatures present automatically scaled to span the spectrum from blue to red.

Wakam

Wakam, pronounced wah-cam, stands for WAlabot (creator) Kentucky cAMera. The front of the unit is shown in Figure 8. It is designed to combine 360° visible-light capture, using the same Insta360 Air[5] dual-camera module used in KVIRP, with 3D radar imaging.

The Walabot Creator[10] is a radar imaging array. It is sold in the form of a 72x140mm populated circuit board with fifteen linearly-polarized broadband antennas mounted on the front. The unit operates in the 3.3-10GHz range with an average transmit power below -41dBm/MHz, which is considered to be harmless to humans and animals – although it could potentially interfere with 5GHz WiFi unless configured to use either the 3.3-4.8GHz or 6.3-8GHz sub-band. Control of the array and transmission of minimally-processed data is accomplished via USB. Walabot provides a low-level API[11] and SDK that can be used with C++ or

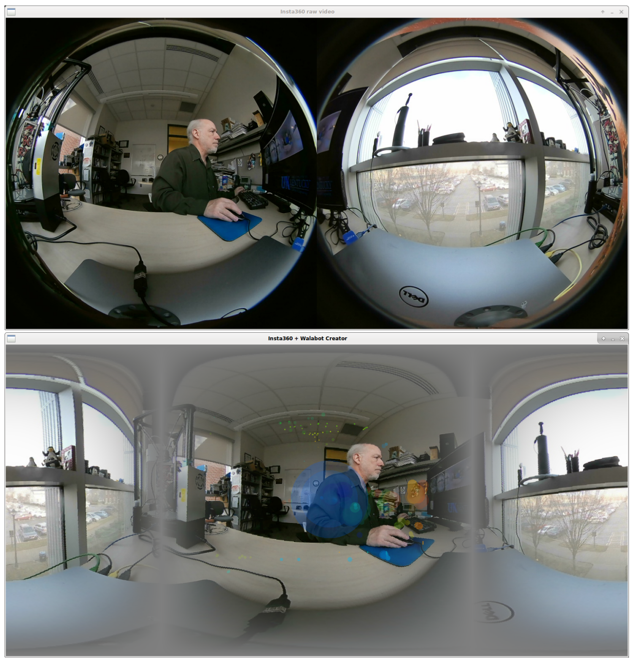

Figure 9. *Wakam raw image data and fused result*

Python. Scanning profiles are provided for short-range scanning within walls (the primary use of their commercial product) and for high or low resolution sensing at a distance.

Radar sensor data is of very low resolution compared to the visual data from the Insta360 Air. Despite inherently measuring in a spherical three-dimensional space, the Walabot typically is configured to report the spherical coordinates of (just a few) "tracked" objects. Thus, it behaves as a point sensor. In Wakam, the Walabot Creator is held in its reference orientation, with X, Y, and Z as shown in Figure 8. In addition to reporting signal amplitude, the radial distance (R, in centimeters), Theta (θ, in degrees), and Phi (ϕ, in degrees) map into X, Y, and Z coordinates in the scene space as:

$$X = R \times sin(\theta)$$
$$Y = R \times cos(\theta) \times sin(\phi)$$
$$Z = R \times cos(\theta) \times cos(\phi)$$

Despite the point nature of the radar target sampling, `wakam.cpp` uses a full property map for both the radar data and its confidence. Each target is painted into the radar value property map at an aligned location using Z as the value. The initial confidence property values are derived from the amplitude. Each target is painted as a spot with width also a function of the amplitude – higher amplitude paints a larger spot. The initial confidence within a spot drops off further from the center, approximating a sphere. The painting of a target's data also applies confidence-weighting to merge with any previous values in the property map.

The raw Insta360 frame and corresponding fused image are shown in Figure 9. The radar data is automatically scaled to the distances present, showing the closest regions as blue and more distant ones as red.

Conclusion

The one-sentence summary of this work is:

Make confidence as obvious as it is important.

Adding indications of confidence in presentation of data can provide benefits without making the display harder to understand[3]. Here, several examples were used to demonstrate the basic concepts behind **Senscape**, a painting-based scheme for modeling and presenting uncertainty in real-time image streams created by sensor fusion. Four features of the processing are key:

1. There is a base, reference, image canvas for alignment
2. Each property has an aligned history
3. Each property maps values *and* confidence
4. Rules govern confidence update and display

This type of integration of Bayesian modeling of confidence with a real-time painting algorithm can be applied to fuse data from a wide range of both imaging and point sensors.

Two open-source testbeds were developed in support of this work, KVIRP and Wakam. KVIRP fuses visible and thermal images, while Wakam fuses visible images and point radar data. The hardware designs and software for both are freely available via `Aggregate.Org/DIT/SENSCAPE` so that they may serve as inexpensive platforms for further research.

References

[1] Ravi K. Sharma, Todd K. Leen, and Misha Pavel, "Probabilistic image sensor fusion," Advances in Neural Information Processing Systems, pp. 824-830. 1999.

[2] Huadong Wu, M. Siegel, and S. Ablay, "Sensor Fusion Using Dempster-Shafer Theory 11: Static Weighting and Kalman Filter-like Dynamic Weighting," Proceedings of the 20th IEEE Instrumentation Technology Conference (Cat. No.03CH37412), vol. 2, pp. 907-912, July 2003, DOI: 10.1109/IMTC.2003.1207885

[3] Michael Leitner and Barbara P. Buttenfield, "Guidelines for the Display of Attribute Certainty," Cartography and Geographic Information Science, 27:1, 3-14, DOI: 10.1559/152304000783548037 (2013)

[4] Henry Dietz, Clark Demaree, Paul Eberhart, Chelsea Kuball, and Jong Yeu Wu, "Lessons from design, construction, and use of various multicameras," Electronic Imaging, Photography, Mobile, and Immersive Imaging 2018, pp. 182-1-182-10(10), DOI:10.2352/ISSN.2470-1173.2018.05.PMII-182

[5] https://www.insta360.com/product/insta360-air (accessed 1/14/2020)

[6] Helmut Dersch, "Panorama Tools," http://panotools.org/dersch/ (2002 archive accessed 1/14/2020)

[7] Tuan Ho and Madhukar Budagavi, "Dual-fisheye lens stitching for 360-degree imaging," 2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), ISSN 2379-190X, pp. 2172-2176, March 2017, DOI:10.1109/ICASSP.2017.7952541

[8] Dean Miller, "Adafruit AMG8833 8x8 Thermal Camera Sensor," August 22, 2018, https://cdn-learn.adafruit.com/downloads/pdf/adafruit-amg8833-8x8-thermal-camera-sensor.pdf (accessed 1/14/2020)

[9] Sparkfun, "ATmega32U4 Arduino board," https://github.com/sparkfun/Pro_Micro (accessed 1/14/2020)

[10] "WALABOT - Technical Brief," https://walabot.com/docs/walabot-tech-brief-416?type=pdf (accessed 1/14/2020)

[11] "Walabot API Beta," https://api.walabot.com/ (accessed 1/14/2020)