Image Segmentation Using Hardware Forest Classifiers

Neil Pittman, Alessandro Forin
Microsoft Research
Redmond, WA USA
pittman@microsoft.com,
sandrof@microsoft.com

Antonio Criminisi, Jamie Shotton
Microsoft Research
Cambridge, United Kingdom
antcrim@microsoft.com,
jamiesho@microsoft.com

Atabak Mahram
Boston University
Boston, MA USA
mahram@bu.edu

Abstract—Image segmentation is the process of partitioning an image into segments or subsets of pixels for purposes of further analysis, such as separating the interesting objects in the foreground from the un-interesting objects in the background. In many image processing applications, the process requires a sequence of computational steps on a per pixel basis, thereby binding the performance to the size and resolution of the image. As applications require greater resolution and larger images the computational resources of this step can quickly exceed those of available CPUs, especially in the power and thermal constrained areas of consumer electronics and mobile.

In this work, we use a hardware tree-based classifier to solve the image segmentation problem. The application is background removal (BGR) from depth-maps obtained from the Microsoft Kinect sensor. After the image is segmented, subsequent steps then classify the objects in the scene. The approach is flexible: to address different application domains we only need to change the trees used by the classifiers. We describe two distinct approaches and evaluate their performance using the commercial-grade testing environment used for the Microsoft Xbox gaming console.

Keywords-NUI; FPGA; Smart Cameras; Computer Vision; Kinect;

I. INTRODUCTION

The Microsoft Kinect is a Natural User Interface (NUI) device that allows users to interact with computer systems using their bodies. This technology involves depth sensing cameras producing images at a rate of 30 frames per second, and software to process those images. The software tracks multiple human participants in a scene and identifies their poses, down to their individual body-parts. This is a computationally complex process that requires a tremendous amount of compute resources on the host system, even with GPU acceleration. The computational requirements limit the applications and form factors where we can use NUI, currently ruling out interesting cases such as mobile phones and tablet computers. All these and more require a lower power profile, and using dedicated hardware is one way to handle this load with acceptable power usage.

The existing Microsoft Kinect software pipeline illustrated in Figure 1 has four stages: Background Removal (BGR), Body Part Classification, Centroid Calculation, and Model Fitting.

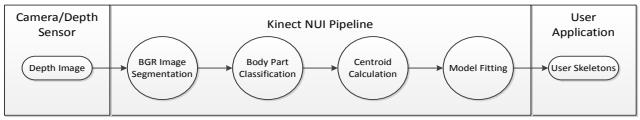

Figure 1. Microsoft Kinect NUI Pipeline.

The BGR step tags each pixel in the depth map as belonging to a player or the background. Body Part Classification further refines the player pixel classification with the probability of belonging to one of 31 body parts (head, neck, hands, etc.). This step has previously been demonstrated in hardware [1]. The third step, Centroid Calculation, aggregates the probability maps into one or more centroids, e.g. the specific location of the center of each body part, for each player. The last step, Model Fitting, aggregates the centroids into human skeletons, dealing with noise and occlusions. The Model Fitting step currently is not computationally intensive and it is performed on the CPU. Figure 2a shows how on the Microsoft Xbox console, the first step is performed on the CPU, the second and third on the GPU, and the last step again on the CPU. The new, alternative pipeline proposed here is illustrated in Figure 2b.

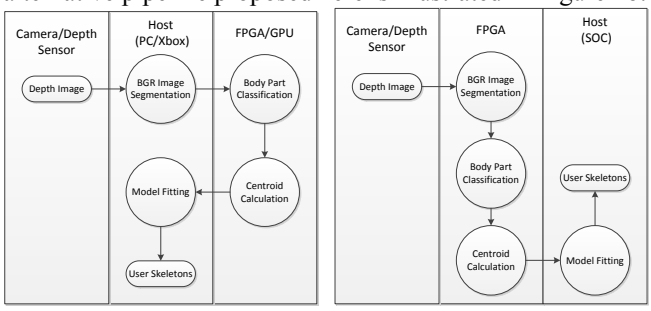

Figure 2. Comparison of *a)* the Split Software-hardware Pipeline and *b)* the Mostly-hardware Pipeline.

The BGR stage simplifies the body part classification problem by identifying the islands or subsets of pixels that likely belong to players. This process is highly sequential, involving comparison of each active pixel to its neighbors. Effectively, the BGR job is to decide if each pixel is part of a human player or not. If a tree based classifier [8] can learn to identify parts of a body, it stands to reason that it might also be able to learn to separate a human body from other objects in the scene. This is the basic hypothesis that we successfully investigated in this work.

To use NUI in a power-constrained environment we would prefer to connect the camera directly to the pipeline (instead of via USB), and to compute as much as possible within the device before sending the result to the host CPU. This demands a hardware implementation of BGR. The existing Microsoft Xbox software implementation is complex and has proven challenging to port to hardware. In this work we propose the alternative, novel method of using our algorithm ("*Forest Fire*") [8] to perform the BGR step. The input depth image is sent to a first classifier that separates the interesting portions (e.g. the humans) from the background. The regular second stage then operates on the filtered image. Both stages are implemented in hardware, using two replicas of the classifier each trained on different data sets.

We also evaluate a second approach, whereby we fuse the first and second phases together, and apply the body part classifier directly to an un-segmented image. The single classifier performs BGR and body part labeling at the same time. This second approach is much simpler, but results in lower quality and worse performance.

Section II presents some background material, including related work. Section III describes the systems we realized, and expands on specific problems like floor detection, player tagging, and forest training. Section IV reports our results, and Section V concludes.

II. BACKGROUND

A. Previous Work on Image Segmentation

Uses of hardware acceleration for the image segmentation problem have been limited, with most of the work surveyed focused on the acceleration of software algorithms. Application domains include image quality, general vision applications, player identification and body tracking, medical imaging, and 3D world reconstruction.

Both our work and the work of Oberg et al. [1] are based on Criminisi & Shotton [8], who use decision forest classifiers in a range of computer vision applications, including identifying body parts in the Microsoft Kinect. We now further extend it to perform image segmentation in hardware. Yin et al. [7] used a software classification forest to perform image segmentation on non-depth sensing webcams. This technique compares well to stereo camera approaches but with a peak of 7.7 fps, lacks the performance to be useful for our real-time application. Kinsella [2] improves the image quality of webcams, realizing several image segmentation algorithms on a Digilent Spartan 3 Evaluation board (Grey-Scale Histogram, Contrast Stretching, Histogram Equalized Stretch, Bimodal Distribution and Thresholding). Segmentation for object classification requires a different approach; the most successful so far is using a statistically trained tree classifier. Yang & Welch [3] accelerates image segmentation using NVIDIA GPUs, for background removal in general computer vision applications. The register combiners of the GPU are used to compute the squared distance of all pixels, and thresholding then separates objects in the scene. Some additional image morphology is done using blending, such as erosion and dilation of the edges of objects. Yang reports a 30% improvement over software. While this work could conceivably be used for our purposes, the power requirements and limited speedup over CPU are a concern. MacLean [6] provides an overview of the field, and motivates the suitability of FPGAs for computer vision applications in general.

B. Software BGR

Using motion between frames, the software version of BGR used in the Microsoft Kinect product identifies pixels as candidates for active player tagging. Using a Connected Components algorithm, these active pixels are combined with other nearby pixels into pixel islands, using a gradient descent approach. In the ideal case, a player mask will then emerge as a single island. However, it is often the case that a player mask must be assembled from multiple islands. This is accomplished using motion, history from previous frames, and a fairly complex set of manually designed rules for combining and splitting islands. This results in a complex and entirely sequential software, making it undesirable for implementation in FPGA. In this work, we explore alternative approaches that can solve the problem with acceptable quality and better performance.

Note that BGR does not simply separate players from the background. Players are individually tagged, consistently from frame to frame, and objects that only vaguely resemble human shapes are rejected. Additionally, the Model Fitting stage requires the identification of the floor plane for a precise definition of the player's position in the world space. A hardware BGR implementation must provide the same information with similar or better quality.

We have tested the possibility of running BGR in software on an embedded processor. In a first test, we used an Intel Atom processor at 1.6 GHz, which resulted in a frame rate of approximately 14.3 fps. In a second test, we used the ARM processor on the Microsoft Surface tablet running at 1.2 GHz, resulting in a frame rate of approximately 7 fps. The ARM on the Xilinx Zynq is similar but runs at half that speed. While neither implementation was well tuned, the distance to the minimal acceptable frame rate of 30 fps is quite large, and does not even take into account the remaining phases of the pipeline. Note that in practice, the frame rate must be even higher, possibly at 90+ fps, to allow end-user applications to run concurrently.

C. FPGA Forest Fire

Forest Fire is a random tree based classification algorithm used by the Microsoft Kinect to classify pixels of a depth image as human body parts. Each active pixel traverses several binary trees. Starting at the root, a decision is made to proceed to either left or right child based on an evaluation function. Eventually, this traversal will reach a leaf node where the probabilities the current pixel belongs to a particular body part are stored. The results from each tree are then aggregated together [8].

Oberg [1] produced a high performance hardware implementation of the Forest Fire classifier, for body part classification. Memory accesses are the primary bottleneck for this type of system, and the tree traversal and the Sorting FIFO produce the optimal memory access sequences. In this work we reuse this core, with some modifications. The system described in [1] is similar to the Microsoft Xbox platform in that it splits the four steps of the pipeline between hardware and software in the same manner, as shown in Figure 2a.

The implementations described in this paper are shown in Figure 2b. Our intended target is a SoC system similar to the Xilinx Zynq. We timed the Model Fitting stage on the Microsoft Surface tablet at less than 1 ms per frame, confirming our estimates that this stage does not seriously affect performance. Given the sequential nature of the Model Fitting code, a hardware implementation will not give any performance benefit, while the area cost could be quite noticeable. The system in Figure 2b is therefore preferable for a low-power, embedded realization.

III. HARDWARE BGR

Figure 3 is a composite block diagram of the three solutions we evaluate in this paper. *Baseline* uses the BGR software step, using the Connected Components algorithm to create the foreground map, and RANSAC [4] to compute the floor. In *One-Stage* we feed the input depth image directly into a single classifier trained with an augmented forest labeled with all the original 31 body parts, plus the floor. An additional element handles the floor data to detect the exact equation of the floor plane. *Two-Stage* uses two instantiations of the Forest Fire classifier. The first instance (left side) separates floors, humans, and 'anything else'. We then feed the 'humans' foreground map to the original Microsoft Xbox body-part classifier for further identification of the various body parts. Note that only the Baseline solves the problem of *player separation*, e.g. consistently over time assigning a given body part to a specific player. The other two solutions require an additional Pre-Model Fit module.

A. One Stage Classifier

The One-Stage forest classifier is simple and requires no additional hardware other than what is being used for the body part classification. The only difference is that we have augmented the set of labels of interest with background related ones. Furthermore, the use of a single classifier minimizes the external memory requirement. The chief disadvantage is that the classifier must operate on every pixel of the image, without filtering. In contrast, the image segmentation step (BGR) filters out background pixels and the classifier only operates on the active foreground map. In practice, the ratio of background to foreground pixels is 4:1 or more. Even if the classifier does not change, this at least triples the processing time. In other words, the classifier operates constantly in what [1] describes as "stress performance", leading to 56fps. And clearly the classifier trees must now grow to accommodate the floor and still produce the same original accurate results. Nonetheless, to make for accurate comparisons we trained the forest to mirror the original body part forest. The new forest also has three trees, each 20 levels deep. The total size of the forest is about 24 MB. The only difference is the additional class label for the floor pixels. Performance is 56fps, for all inputs.

B. Two Stage Classifier

A number of considerations led us to investigate the Two-Stage approach, also represented in Figure 3, where first we perform a coarse-grained classification into foreground, background and floor, followed by a second, more fine-grained classification just on the foreground (player) pixels. Performance is one reason, e.g. we can use a small and faster forest for the first step filtering and only apply the heavy duty body part classification step to $1/4^{th}$ – $1/5^{th}$ of the entire image. This raises the average performance back to over 200 fps. A second reason is that the hardware resources required for implementing the classifier on the FPGA is only a small percentage of those available; replicating the entire unit is therefore quite feasible. A third consideration is that we can reuse in the second step the existing production Microsoft Xbox forest, which was trained on millions of images. As it turns out, the first forest does not require quite as much training data. Moreover, the small number of classes can be progressively augmented in the future with additional classes, e.g. representing other scene features such as walls, sofas, etc.

In practice, the separation of background and foreground can be done at a lower resolution than body parts: it is possible to achieve reasonable results by subsampling the images in the BGR step and then up-sample the resulting mask for the body part classification step. The most important consideration is performance. Producing good

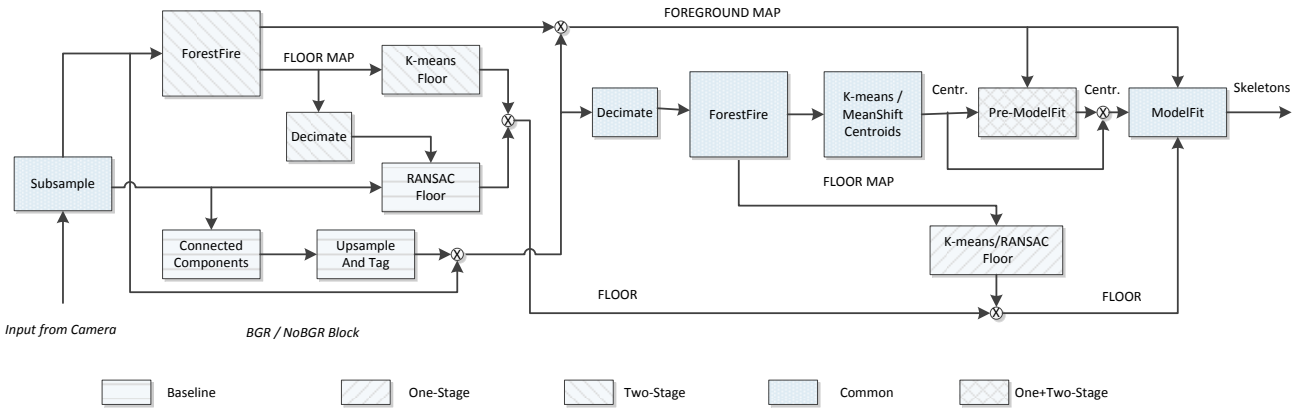

Figure 3. Skeletal Tracking Hardware Pipeline Block Diagrams.

classification accuracy on fewer classes is achievable with smaller forest depth. If the forest were small enough, it would be possible to store it on chip rather than in external memory. We evaluate a small forest of 24 KB (3 trees, 10 depth levels) and a larger 1.5 MB forest (3 trees, 16 levels). Both will fit in the internal memory of many modern FPGAs. Eliminating the external memory accesses reduces the computational cost of this first step considerably.

C. *Floor Computation*

The Baseline system uses RANSAC [4] to find the equation of the plane with the minimal distance from all the pixels in the bottom 10% of the image. An additional step refines the floor candidates to the one with the minimum eigenvalues across all floor pixels. The algorithm checks that the Microsoft Kinect sensor is placed to within ± 20 degrees of the floor normal. If we apply some other form of filtering, such as a classifier, we can relax this vertical orientation requirement and perform RANSAC on all floor pixels. Note that on a mobile device the orientation restriction cannot be enforced, floor pixels must be classified regardless of the camera orientation. Both of the hardware approaches described in this work can do this.

A RANSAC algorithm can provide a high degree of noise reduction, e.g. mathematically any three 'noise-less' floor points would lead to the same plane equation. Classification and centroid computations are an alternative form of noise filtering. This leads us to consider a simpler and more efficient way to compute the floor plane equation. Just like for body part pixels, we compute centroids for the floor pixels using our hardware streaming *k*-means algorithm. This generates a relatively large number of floor 'centroids', but a much smaller number than the total number of floor pixels (in practice, a hundred or so against a few thousands). We then identify a bounding cube for the floor that is defined by the six centroids on the faces of the bounding box. From the 21 combinations of these points (taken 3 at a time) we have 21 candidate planes with a normal pointing out of the floor and up into the scene. Each candidate floor is tested against all the remaining centroids to find the plane with the best fit. Note that, by construction, each candidate has a well-known orientation with respect to the coordinate system. With RANSAC, points are selected randomly and therefore both of the up/down planes are equally valid and indistinguishable.

Both modules can be used in our system, as shown in Figure 3. RANSAC offers better noise reduction; *k*-means is more efficient and eliminates any up/down uncertainties. Note that in a hardware implementation, RANSAC can compensate for the performance advantage of *k*-means by computing in parallel and asynchronously to the frame stream. The floor orientation in practice will not change very frequently, likely not at every frame.

D. *Player Tagging and Model Fitting*

Software BGR performs the necessary functions of player labeling and player tracking. Different players must be labeled differently in a frame, and the same player must be assigned the same ID from frame to frame. The connected components based approach naturally leads to player separation, and remembering the center of body mass of a player from frame to frame accomplishes the required consistency in tagging. Since the Forest Fire classifier does not provide this function, it must be implemented elsewhere to maintain backward compatibility. This module is identified as Pre-ModelFit in Figure 3. Through experimentation we verified that the original Microsoft Xbox body part forest and the Forest Fire hardware can accurately classify pixels into body parts whether they are pre-partitioned by player or not. The probability maps simply indicate multiple hot spots corresponding to the various instances of e.g. a left hand. Therefore in the two stage classifier, *all* the foreground pixels are just labeled 'player 1' and passed to the second stage.

Unfortunately, the problem has only moved and the Microsoft Xbox Model Fitting algorithm still needs the centroids partitioned by players in order to properly assemble skeletons. The Pre-Model Fit step (in software) takes a single list of centroid candidates and splits it into multiple per-player lists. The fine details of this step are out of scope, but intuitively the problem must be solvable more easily when handling 4-8 candidate centroids per body part than when handling many thousands of raw, untagged pixels. One approach is to use the same island-based gradient descent algorithm used by software BGR, only at the centroid level. A simpler approach (used in the evaluation in Section IV) is as follows:

First partition the head centroids. Then looking at the clusters of head centroids, estimate the number and location of each head in the scene. Use the neck and left/right torso candidates to verify the estimate of the number of players. Double-check the estimate against the previous frame, assuming the addition and deletion of a player is an infrequent event. Use a combination of proximity and connectivity tests to connect heads to necks, shoulders and torsos. Finally, add limbs to each player's centroid set. Note that we should err on the side of caution; we can add the same weak centroid to all player sets and let Model Fitting sort it out. But we must assign a strong candidate to the correct list. Consistent tagging works the same as in software BGR, and we can track the head or torso centroids from frame to frame. As for observed performance, the two combined model fitting steps are still about 1ms per frame.

E. Forest Training

The Microsoft Xbox product is routinely tested against a very large repository of clips, e.g. short depth-image movies of about 200-500 frames each. We found a way to leverage this data and generate the training sets for the two new forests required by our experiment. We use the baseline Microsoft Xbox pipeline to identify and eliminate the players from the scenes, leaving only the background depth-images. We then use computer generated human models of various sizes, body types, and different poses and carefully insert them into the 'holes' left by the original players.

This approach is automatic, can generate a large number of training images, and lets us create the required variety in the player's poses which was somewhat missing in the original clips. Since the CG players are computer-generated, we know in advance where their body parts are and we can automatically generate the ground-truth labels. Floor pixels are labeled using the computed floor equations.

Additional work is necessary to refine this forest training process and improve the size and quality of the training sets beyond our current research prototype. We used small training sets of 4000 images for the 32 class forest used by the One-Stage version, due to the long time required to perform training. We trained the 3 class forest used by first step of the Two-Stage version on 10000 images. With these set sizes we can produce a new forest in about 24 hours using an Intel i7 hex core PC at 3.2 GHz with 64 GB of memory. The current quality of this training data could be improved: the technique of inserting computer generated human figures into existing scenes produces artifacts, especially for small body parts like hands, feet, wrists and knees.

IV. RESULTS

We used the following setup for testing. A host PC provides feeds from live cameras or clips saved on disk, using the Simple Interface for Reconfigurable Computing (SIRC) [5]. The FPGA returns centroids and the plane coefficients of the floor to the PC, who then performs the final Model Fit stage(s). We used a Xilinx Virtex6 240t ML605 Evaluation Board (xv6vlx240t-1ff1156). We evaluated the two hardware BGR approaches against three Microsoft Xbox suites of test clips and compared them with the original baseline system and with the ground truth. Each suite is dominated by a different type of clips: Suite 94 is mostly standing humans with a variety of backgrounds, Suite 97 is standing humans with very similar backgrounds and Suite 111 is dominated by seated humans with furniture. The suites contain 193, 533, and 155 clips respectively. Each clip includes hundreds of frames. We run each clip through the system under test. The *Baseline* is the existing Microsoft Kinect for Windows SDK, *one-stage* is the single classifier system, and *two-stage* is the two-step classifier system. For each frame we log and compare the final skeleton's joints positions from Model Fitting.

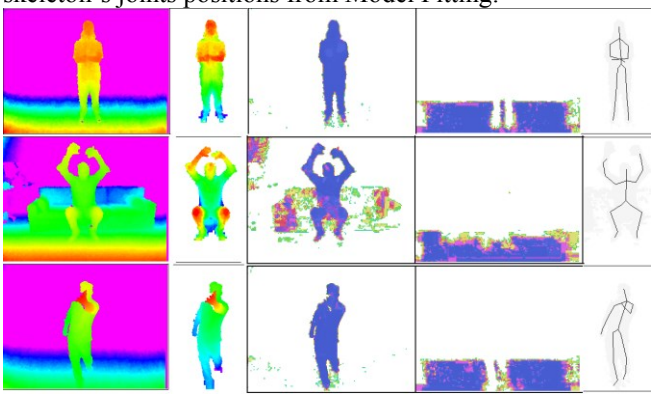

Figure 4. Results For Three Selected Images.

Figure 4 shows the results for one individual frame per suite. The first column is the depth map. The second column is the player tagged by the Baseline software BGR. The third column is the foreground probability map from the two-stage system. The fourth column is the floor probability map. The fifth column is the final skeleton, superimposed onto the player's silhouette. This visual comparison is impractical for millions of frames, and does not provide any quantitative result.

Table 1. Results for Suite 94*, 97 and 111***.**

Suite 94						Suite 97						Suite 111					
RANSAC			k-means			RANSAC			k-means			RANSAC			k-means		
Diff. Joints and Limbs (%)																	
Base line	one stage	two stage	Base line	one stage	two stage	Base line	one stage	two stage	Base line	one stage	two stage	Base line	one stage	two stage	Base line	one stage	two stage
95.07	97.85	97.08	95.07	97.96	97.22	46.79	68.77	54.58	**46.79**	**68.76**	**54.62**	79.14	94.70	79.02	**78.93**	94.61	**78.54**
60.23	62.53	62.94	**60.23**	**64.14**	**63.38**	9.51	21.40	10.40	**9.51**	21.65	**10.69**	41.41	77.63	40.63	**41.32**	77.45	**41.45**
Average and Maximum Joint Difference X, Y, & Z (m)																	
Base line	one stage	two stage	Base line	one stage	two stage	Base line	one stage	two stage	Base line	one stage	two stage	Base line	one stage	two stage	Base line	one stage	two stage
0.13	0.14	0.13	0.13	0.15	0.13	0.02	0.04	0.02	0.02	0.04	0.02	0.07	0.26	0.06	0.07	0.26	0.06
0.75	0.80	0.77	0.75	0.81	0.80	0.18	0.40	0.20	0.21	0.40	0.22	0.41	1.51	0.47	0.41	1.51	0.48
Average Length (m), Inclination and Azimuth (degrees) Difference																	
Base line	one stage	two stage	Base line	one stage	two stage	Base line	one stage	two stage	Base line	one stage	two stage	Base line	one stage	two stage	Base line	one stage	two stage
0.05	0.07	0.05	0.05	0.07	0.06	0.02	0.03	0.03	0.02	0.03	0.03	0.05	0.35	0.05	0.05	0.34	0.05
41.6	44.9	42.1	41.8	45.8	42.5	10.0	16.4	10.3	10.0	16.5	10.4	24.4	58.5	26.4	25.2	58.4	27.2
17.9	18.6	17.7	17.9	19.1	17.7	9.0	12.1	8.9	9.0	12.1	8.9	16.5	28.9	16.7	16.4	28.7	16.8

*standing with varied scenes, **standing with similar scenes, ***seated and furniture.

A. Skeletal Tracking Results

One quantitative approach is to compare each joint location in space, then average and aggregate the results. This is a probabilistic task and 100% accuracy is impossible. Consequently, a large location error will not correspond to a large visual error. A more consistent and effective measure of accuracy is to look at the lengths and especially the orientations of the skeleton's limbs. Consider the case of two small deltas in e.g. elbow and wrist position. The resulting axis of the lower arm can be off by as much as the sum of the two (opposite) deltas, or not at all. Visually, we are much more affected by the angular differences.

In Table 1, we report the number of *joints* locations different from the ground truth and their average and maximum distance along each dimension (X, Y, Z). Lower values are better. We calculate the number of *limbs* different from the ground truth and the average differences in length, inclination and azimuth. These statistics are then aggregated per suite. Since we have two alternate algorithms for floor calculation, we report statistics for both of *RANSAC* and *k-means*, to evaluate the effects they have on the rest of the pipeline. For the suites tested, both algorithms are effective and their differences are small.

As Table 1 shows, most of the joints differ widely from the ground truth, baseline included. For suite 94, the versions using the hardware BGR instead of the software BGR had between 2 and 3 percentage points more different joints and between 3 and 4 percent more different limbs. Overall, the two-stage classifier did slightly better than the one-stage classifier. This is to be expected given the much smaller training set for the forest. The two-stage classifier was also much closer to the baseline in the average difference along each dimension. In suite 97, the baseline performed much better and thus the difference against hardware BGR is more striking. Here the joint difference from the baseline ranged from 12 percent for the one-stage classifier to 8 percent for the two forest version. However, in the case of the two-stage classifier the average limb deviation from the ground truth is still very small. More surprising are the results of Suite 111, the seated clips. Here the baseline shows differences in over three fourths of the joints and almost half the limbs. The one-stage classifier came nowhere close missing 95 percent of the joints and 75 percent of the limbs. But again the two-stage classifier showed close to the same number of correct joints and limbs. The average difference was just a little larger than in suites 94 and 97. This result is surprising since the training data we used for our forests were standing humans only, no seated positions at all.

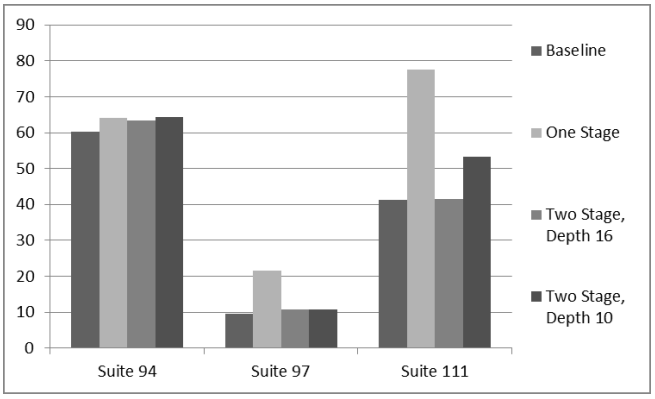

Figure 5. Percent limbs that differ from ground-truth.

Figure 5 summarizes the limb comparisons, which is better correlated to the visual outputs of Figure 4. Since the choice of floor detection algorithm has a marginal effect on the final accuracy, we only show the *k*-means results. In addition to the systems of Table 1, we evaluate the Two-Stage system using a smaller forest of three trees of depth 10 rather than 16, using the same test suites. Reducing the depth from 16 to 10 levels reduces the size of the forest from 1.5 MB to 25 KB, and with fewer levels to evaluate each frame can be processed 50% faster in this stage.

Overall, Figure 5 shows that the Two-Stage/16 system is almost identical to the baseline, for all test suites. The

reduced tree depth of Two-Stage/10 has a small negative effect on suites 94 and 97, and a larger impact (+14%) on suite 111. Given that suite 111 is dominated by seated positions and there is additional noise from objects in the background, this is not unexpected. The One-Stage system is the worst performer in all cases, with differences against baseline of 4%, 12%, and 36%.

B. *Floor Computation Results*

The effect on the skeleton pipeline is small, but the floor computation accuracy (and latency) can be more relevant in other scenarios, such as with mobile devices. We compare the RANSAC and *k*-means solutions in the following way. We use one frame each from the test clips, plus a number of frames from other test clips. We establish the ground-truth by running an unlimited number of RANSAC iterations on each frame, stopping only when we have a stable result. We discard 12% of images for which we cannot generate a floor, resulting in a total of 2,922 test floors. Computations are performed in double-precision floating point for the ground truth. The RANSAC test case is then split into an IntegerRansac and FloatRansac case, to reflect the hardware and software implementations, respectively. These cases differ from the ground truth RANSAC because of the limited number of iterations allowed (within a single frame time). *K-means* only has one (integer) test case.

Table 2. Floor detectors performance & accuracy.

Algorithm	Floors Detected	Percent of Total (%)	Inclination (degrees)	Azimuth (degrees)
FloatRansac	2,366	80.9	20.5	9.9
IntegerRansac	2,483	84.9	2.5	6.3
k-means	2,912	99.6	11.0	11.8

As shown in Table 2, *k*-means can find a valid floor in the allotted time in a higher number of cases. As for accuracy, the angular deviations from ground-truth indicate that these results are close, but not as accurate as with IntegerRansac. Interestingly, FloatRansac produces the worst results both in performance and accuracy. This version generates a larger number of invalid point selections, resulting in less iteration in the allotted time.

C. *Device Utilization*

We implemented two hardware prototypes of the system. For the one stage classifier, the system is quite similar to the one in [1], and therefore worth comparing to. For the two stage classifier, the Forest Fire module is instantiated twice, and to save memory the foreground image data is written back into the same input buffer after the first stage completes. As mentioned, it is possible to store the first stage image segmentation forest on chip, but for simplicity we stored both forests in DDR3 memory. We did not investigate pipelining the two classifiers.

Table 3 shows the utilization of the Virtex 6 240t (xv6vlx240t-1ff1156) for the One-Stage system. The utilization reported in [1] for the same chip was 7.5% LUTs and 30% BRAMs. Our implementation now returns the centroids of the body parts instead of pointers to the forest leaves for each pixel. In other words, each Forest Fire core now includes an additional *k*-means centroid computation unit that was previously absent. This has tripled the LUTs while cutting the block rams to a fifth. This also significantly reduces the output bandwidth requirement. The size of the Sorting FIFO is reduced to 2 block rams rather than 33. The Forest Fire core now performs processing in batches of 1024 pixels rather than loading the entire foreground image, but the effect of this batching on the frame rate is negligible. The implementation of the DDR3 Controller and PC Interface are largely unchanged, they only appear to have changed because synthesized with the 14.2 release of the Xilinx IDE.

Table 3. FPGA Utilization of the One-Stage Prototype.

	LUTs	*FF*	*BRAM*
Full System	37470 (24.86%)	31796 (10.55%)	27 (6.49%)
Forest Fire Core	30129 (19.99%)	23198 (7.69%)	5 (1.2%)
Sorting FIFO	425 (0.28%)	428 (0.14%)	2 (0.48)
DDR3 Controller	5488 (3.64%)	7496 (2.49%)	0 (0%)
PC Interface	1852 (1.23%)	1101 (0.37%)	22 (5.29%)
Input Buffer	76 (0.05%)	42 (0.01%)	11 (2.64%)
Output Buffer	0 (0%)	0 (0%)	8 (1.92%)

The operation of the One-Stage system is as follows. The depth image is downloaded to the FPGA over Ethernet using SIRC. The pixels are classified by the random decision tree classifier using the forest stored in the DDR3. The centroids of each body part type are calculated and written to the output buffer. The results are transmitted back to the host using Ethernet.

Table 4. FPGA Utilization of the Two-Stage Prototype.

	LUTs	*FF*	*BRAM*
Full System	67236 (44.61%)	55405 (18.38%)	32 (7.69%)
Forest Fire Two Instantiations	60611 (40.21%)	46614 (15.46%)	10 (2.4%)
Forest Fire Core0	29888 (19.83%)	23275 (7.72%)	5 (1.2%)
Forest Fire Core1	29773 (19.75%)	23337 (7.74%)	5 (1.2%)
DDR3 Controller	4746 (3.15%)	7686 (2.55%)	0 (0%)
PC Interface	1876 (1.24%)	1101 (0.37%)	22 (5.29%)
Input Buffer	76 (0.05%)	42 (0.01%)	11 (2.64%)
Output Buffer	0 (0%)	0 (0%)	8 (1.92%)

Table 4 reports the utilization for the Two-Stage system, after adding the second instantiation of the Forest Fire core. While the first stage forest could be stored on chip, for simplicity we chose to store it in the DDR3 along with the other forest. Some additional control logic was required to

chain the two classifier cores, to store and reload the data from DDR3, and to route the data accordingly. With two instances of the classifier core we double area use, but the entire design is well under half of this midrange FPGA. In this scenario Core0 performs the BGR image segmentation and Core1 performs the body part classification on the foreground pixels (see Figure 3). From Core0 the classified pixels are dumped to the DDR3 and those tagged as floor are passed to the centroid and floor calculation modules. The foreground pixels are loaded from DDR3 to Core1 for body part classification.

D. Power Measurements

In order to provide an idea to the potential power advantages of a hardware solution, in Table 5 we present a series of power measurements for consideration. All values are power measurements taken from the wall socket, thus they are power measurements for the entire system.

Table 5. System Power of NUI Implementations.

Platform	Power (W)
PC w/Kinect	162.6
Xbox w/Kinect	92.1
Xilinx ML605 w/Kinect	25.7
Digilent ZedBoard w/Kinect	9.0
Kinect Alone	3.5

In the case of the Xbox, the Kinect sensor is powered through the USB by the host. For the other platforms, the Kinect sensor powered by a separate power supply and is added to that of the host. The average power utilization of the Kinect sensor alone is 3.5 watts.

These measurements demonstrate that all of the processing required can be implemented in hardware with significant power savings using a high end FPGA. This also suggests the Xilinx Zynq, a SOC with a microprocessor and FPGA, could run NUI interface and the application code within a much lower power budget than the Xbox or PC.

V. CONCLUSIONS

The main contribution of this work is a complete, fully-embedded realization of the Microsoft Kinect pipeline, without using powerful CPUs or GPUs, at a very low power, using commercially available FPGAs. To realize the system in hardware, we tested the novel idea of using a classification forest for image segmentation. Rather than segmenting connected objects and tagging pixels, we directly classify them into human body parts and floors.

A straightforward realization with a single classifier does not perform well compared to the commercial grade Xbox baseline. This system is compact, but the excessive noise and limited forest training are negative factors. Performance is also negatively affected by the lack of filtering. A two-stage approach instead performs acceptably. A first classifier detects the foreground and floor data, and a second classifier performs the actual body part extraction. This system is no worse than 7% of the baseline on the entire Xbox production test suites, and only 1% worse on average. Additionally, we can leverage the shipping main forest and augment it with a second, much smaller forest acting as filter. The hardware implementation reduced the overall latency of the system, with a frame rate in excess of 200 fps.

We have considered a number of alternative algorithms for the segmentation task. None meets all of the requirements, especially the separation of players from each other. We reduce the separation problem to the assigning of the identified body parts to the correct players, which is computationally easier. Training of the forests requires some creativity: we use a number of scalable techniques to leverage a large amount of unreliable data in effective ways. For evaluation, the best metric is to use the final skeleton's limb orientations. In addition to the players, the system must correctly identify the floor location, e.g. the world-space coordinates. We have described and evaluated both a RANSAC-based and a novel *k*-means based approach.

VI. REFERENCES

[1] J. Oberg, K. Eguro, R. Bittner and A. Forin. "Random Decision Tree Body Part Recognition Using FPGAs." The 22nd International Conference on Field Programmable Logic and Applications (FPL 2012). August 2012.

[2] B. Kinsella. "Embedded Image Segmentation on an FPGA". National University of Ireland. Galway, Ireland. April 2008.

[3] R. Yang and G. Welch. "Fast Image Segmentation and Smoothing Using Commodity Graphics Hardware." Journal of Graphics Tools – Special Issue on Hardware Accelerated Rendering Techniques. Volume 7. Issue 4. December 2002.

[4] M. A. Fischler and R. C. Bolles. "Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography". Communications of the ACM 1981, 24:381-395.

[5] K. Eguro. "SIRC: An extensible reconfigurable computing communication API." Field-Programmable Custom Computing Machines (FCCM), 2010 18th IEEE Annual International Symposium on. IEEE, 2010.

[6] W. J. MacLean. "An Evaluation of the Suitability of FPGAs for embedded Vision Systems." IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshop. June 2005.

[7] P. Yin, A. Criminisi, J. Winn and I. Essa. "Bilayer Segmentation of Webcam Videos Using Tree-based Classifiers." IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI), IEEE, 2010.

[8] A. Criminisi and J. Shotton. "Decision Forests for Computer Vision and Medical Image Analysis". Springer. 2013. ISBN 978-1-4471-4928-6.