VoXED: an online XED file extractor

Amadeo Tato Cota Neto, Bruno Cardoso Dantas, Joao Marcelo Xavier Natario Teixeira, Veronica Teichrieb
Universidade Federal de Pernambuco
Recife, Brazil
atcn@cin.ufpe.br, bcd@cin.ufpe.br, jmxnt@cin.ufpe.br, vt@cin.ufpe.br

***Abstract*—Kinect devices are used up to this day for image capturing and video recording in many activities. These devices make it possible to capture color and depth data, crucial to many research works. Despite that, these data are stored in a xed file, in the case of the Kinect V1, that cannot be opened directly. Existing tools to extract data from these files include the Kinect Studio application and the xed extractor developed by Daniel Jackson. Although valuable tools, they require a higher computational knowledge from the user since they require either the compilation of a source code or to develop an application that uses the stored data like it was received from a Kinect device while a Kinect device is physically connected to the user's computer. In this paper, we propose a web service that allows users to retrieve color data from xed files in an easier way than the existing tools, by retrieving a compressed file with the images stored on xed files. The obtained results show that we successfully retrieved the images stored on these files, making it possible to reuse old xed files containing important information such as body posture datasets.**

***Keywords*—Kinect studio; xed file extraction; body pose information.**

I. Introduction

Capturing pose and depth of people is a challenge in computer vision that has relevance in areas such as gesture recognition, augmented reality, robotics, and security. Its application requires precision in obtaining the positions of the joints and estimating the distance between the camera and the objects. To this end, it is essential to create datasets with pose and depth capture sensors to train computer vision and machine learning algorithms to perform tasks such as gesture recognition and human movement analysis. An example of such efforts is Kinder-Gator [1], a dataset that comprises 58 movements of children and adults.

Kinect is a sensor developed by Microsoft, and its first version was considered a disruptive innovation that has transformed a number of applications, exploiting its benefits in motion capture and gesture recognition. Although it has withdrawn from the market, several areas took advantage of its technology, benefiting from using its sensors to this day. Among the application sectors identified are Education and Research [2], Augmented Reality [3], Industry and Manufacturing [4] and Medical Rehabilitation [5] [6], the latter being the most emphasized in the scientific literature.

Although Kinect sensors are now obsolete, it is still possible to use datasets based on their technology. The challenge, however, lies in extracting the information from the originally recorded .xed files. Based on the original project from Daniel Jackson[1], this work aims to create a free web tool to allow data extraction from this file type. We have carried out tests and hope that this tool will help the scientific community in carrying out research with this type of data.

The remainder of this paper is organized as follows. Section II provides more details regarding how body pose information can be extracted from RGBD sensors and what hardware and software tools may be employed for this purpose. Section III describes the implementation of the proposed solution and its main differences from the original project. Section IV shows the results obtained and the validation of the proposed tool. At last, section V concludes the work, providing a few directions for future work to improve the solution.

II. Body pose from RGBD cameras

Body pose estimation from RGBD cameras involves capturing and analyzing depth and color information to determine the position and orientation of a person's body in a three-dimensional space. Several hardware options and associated APIs have emerged to facilitate this task. Devices like the ASUS Xtion[2], Intel RealSense[3], and Microsoft Kinect[4] have gained prominence in this domain. The ASUS Xtion and Intel RealSense cameras offer compact and affordable solutions with depth-sensing capabilities, making them suitable for various applications. On the other hand, Microsoft Kinect has been a pioneer in this field, providing robust hardware and a well-documented API that allows developers to extract detailed information about the body's pose. In the subsequent paragraphs, we will further describe the capabilities and features of the Microsoft Kinect, shedding light on its extensive potential for body pose estimation and its impact on various industries.

[1]https://github.com/danielgjackson/xed
[2]http://xtionprolive.com/asus-3d-depth-camera
[3]https://www.intelrealsense.com/
[4]https://learn.microsoft.com/pt-br/windows/apps/design/devices/kinect-for-windows

A. Microsoft Kinect

Launched by Microsoft in 2010, Kinect emerged as a motion sensor for the Xbox 360 console that allowed the movement of the human body tracking without the need for physical controllers. This device contains RGB and infrared cameras, as well as microphones, enabling body tracking and voice recognition [7]. Its technology aroused the interest of developers who, with the version for Windows released later by Microsoft, were able to create non-gaming applications for Kinect [8]. In 2013, it received an updated version (Kinect 2.0), but in 2017, Microsoft announced the end of its production. In addition to being used by gamers to control characters using gestures and voice commands, the Kinect, as already mentioned, can be used for various applications, such as medicine, education, augmented reality and research and development, which means that the device continues to be used in various applications today [7] [8].

B. Kinect Studio

Various RGBD cameras have enabled information extraction from users' joints, such as Asus Xtion, Intel Realsense, and Microsoft Kinect [9]. Using these devices, APIs have emerged for different types of applications, such as Registered Relief Depth (RRD) [10], NITCAD [11], and Eldo-care [6].

The Kinect Studio was a valuable tool that came bundled with the Kinect SDK, designed for recording the data captured by the Kinect sensor. This tool was instrumental in enabling developers to record and later analyze depth, color, and skeletal tracking data for various applications. However, it did have some limitations that impacted its usability. One notable limitation was that the Kinect Studio didn't allow for the direct export of recorded data from a .xed file, which made sharing and using the recorded data in other applications more challenging.

To utilize the data captured with Kinect Studio, developers needed to create a custom application that could access and play back the recorded data as if it were coming directly from a physical Kinect sensor. This added an extra layer of complexity to the development process, especially for those who wanted to work with the captured data in different environments or scenarios.

Another significant limitation was that the physical Kinect sensor needed to be connected to the computer when using Kinect Studio. This requirement restricted the portability of recorded sessions and limited the flexibility of developers who wanted to work with the data independently of the physical device.

Despite these limitations, the Kinect Studio remained a valuable resource for researchers and developers in fields such as computer vision, robotics, and game development, providing essential tools for capturing and analyzing motion and depth data from the Kinect sensor.

III. The Proposed Solution

The proposed solution is an image extractor that retrieves color images from the video's frames stored on xed files by Kinect V1 devices. We based our work on the xed file extractor developed by Daniel Jackson in 2013[5]. The code, written in the C language, allows the extraction of images stored in xed files.

Jackson's code is publicly available on the GitHub platform under the BSD 2-Clause, a permissive license that allows modifications and distribution of the source code as well as its personal or commercial usage. The only imposed limitation is the preservation of the license content in the redistributions of the source code and its binaries, regardless the code was modified or not.

A. Original Xed Extractor

Xed files have a basic structure that is exploited by the code to realize the image extraction. The file can be divided into sections as described in Jackson's code. The file sections, in this order, are the header, a set of event packets, closing stream indexes, and the end of the file information.

When running the code, the file's metadata are read to obtain useful data for the image extraction process. Firstly, the extractor reads the file's header, obtaining information about the number of streams and the position of the end of the file information. After that, the code reads, from the end of the file information section, metadata of the packets' indexes and of the frames, such as the width and height of the frames and its payload lengths.

Using the obtained metadata, the code iterates through the packets to start the image extraction. The frame's images are either stored in RGB or Bayer GRBG color models. The code can distinguish in which color model the image is stored by the length of the frame's payload and by the number of pixels in that frame, calculated by multiplying the frame's width by its height. If the frame's payload length is twice the number of pixels in that frame, the image is in the RGB color model. Otherwise, if the frame's payload length is the same as the number of pixels of the frame, then the image is in the GRBG color model. The obtained images are stored in their respective color model in a BMP file.

GRBG images are codified in a way that each pixel contains information about only one color of that pixel from the color image. Due to that, these images are presented in grayscale. Despite that, a color image can be obtained from the GRBG one after additional processing.

[5]https://github.com/danielgjackson/xed

B. VoXED: Extractor Script and Web Service

Since the original extractor code was written in C, it needs to be compiled so an executable file can be generated. However, at the time of writing, an executable file was not distributed to download. This fact makes the extractor usage difficult since it makes necessary the download of both the source code and the tools to compile it to use the extractor. Another factor that may affect the extractor usage is the need to use command line to execute it. These factors combined make the tool not friendly to be used by individuals with limited computing knowledge who need to extract images from xed files to use it on researches or for other purposes.

Our solution, named "VoXED", addresses these difficulties by providing an easy-to-use web service that extracts images from xed files. To develop it, we translated the original extractor code to the Python language and used the Azure Functions to publish the developed code as a web service. To make use of the web service, users may either make an HTTP request to the service[6] using the POST method with the xed file in its body, with the "file" key, or make use of the developed web interface[7] (Figure 1), that simplifies the extraction process to users with low computing knowledge allowing xed files to be submitted to the service through a web form. The max body size of the requisition was set to roughly 500 MB due to Azure limitations. The extractor script, the web interface and the web service source codes are available at GitHub[8] under MIT license.

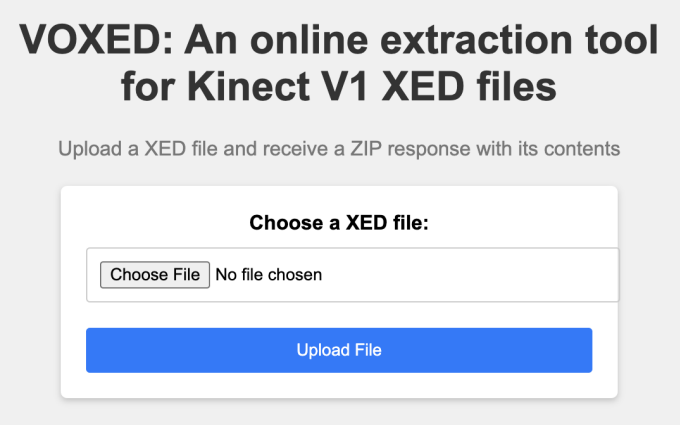

Figure 1. VoXED's web site interface. Users may use the service by providing a xed file to the form field and receive a zip with the extracted images.

When the service receives an HTTP POST request to the "XedDecode" endpoint with a valid xed file, the Azure function temporarily stores it with a random name. The process of image extraction is then executed, being the extracted images stored temporarily. The next step is to compress the extracted images into a zip file that will be sent as a response to the requisition. All the temporarily stored files are removed before the zip content is sent as a response to the requisition.

[6]https://voxed.azurewebsites.net/api/xeddecode
[7]https://voxedteam.github.io/voxed/
[8]https://github.com/VoxedTeam/voxed

C. Differences from Proposed and Original Extractors

Due to the inherent differences between the C and Python languages, some modifications were conducted in the translation process. Below we list what we consider to be the most notable changes:

- The number of code files was reduced from three C code files and two header ones (Figure 2) to a single Python script;

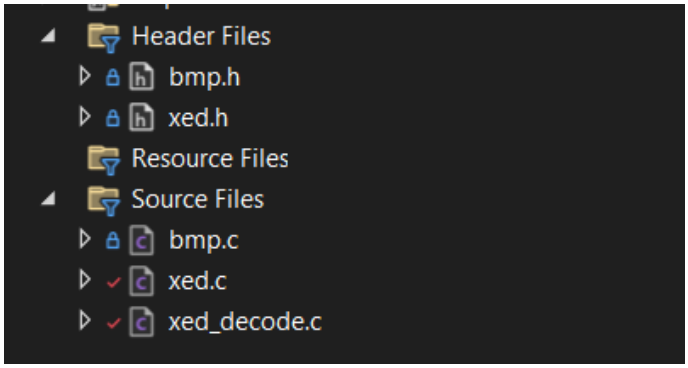

Figure 2. Code and header files used by the original xed extractor to extract the images. All these files were translated to a single python script on our solution.

- External Python libraries were used to generate the image files whereas dedicated code was written to generate these files in the original work. A comparison of the original and translated code is presented in Figure 3;
- Functions used to assign values to fields of structures were moved to the classes' constructors in the Python code. An example of this is shown in Figure 4;
- Pointer variables and low-level memory management are available to be used in C code but not in Python code. We used lists and references to achieve the same results as in the original extractor as shown in Figure 5.

A key difference between the proposed solution and the original extractor is that the original code stores the extracted GRBG images as they are. Our solution, on the other hand, realizes the conversion of the image from GRBG to RGB, returning to the user a color image.

```c
int BitmapWrite(const char *filename, const void *buffer, int bitsPerPixel, int width, int inputStride, int height)
{
    FILE *fp;
    int p;
    int stride;
    int paletteEntries;
    int y;

    if (bitsPerPixel <= 8) { paletteEntries = (1 << bitsPerPixel); } else { paletteEntries = 0; }
    bitsPerPixel = ((bitsPerPixel + 7) / 8) * 8;
    stride = ((((width * bitsPerPixel) + 31) & ~31) >> 3);

    if (filename == NULL || filename[0] == '\0') { return -1; }

    fp = fopen(filename, "wb");
    if (fp == NULL) { return -1; }

    fputc('B', fp); fputc('M', fp); // bfType

    fputlong((height * stride) + 54 + (paletteEntries * 4), fp);     // bfSize
    fputshort(0, fp);              // bfReserved1
    fputshort(0, fp);              // bfReserved2
    fputlong(54 + (paletteEntries * 4), fp);  // bfOffBits

    fputlong(40, fp);              // biSize
    fputlong(width, fp);           // biWidth
    fputlong(height, fp);          // biHeight
    fputshort(1, fp);              // biPlanes
    fputshort(bitsPerPixel, fp);   // biBitCount
    fputlong(0, fp);               // biCompression
    fputlong(height * stride, fp); // biSizeImage
    fputlong(5000, fp);            // biXPelsPerMeter
    fputlong(5000, fp);            // biYPelsPerMeter
```

```python
def extract_image_from_bytes(buffer, width, height, filename):
    img = Image.frombytes("L", (width,height), buffer)         # Open image as grayscale from bytes
    img_array = np.array(img)
    img_rgb = cv2.cvtColor(img_array, cv2.COLOR_BayerGRBG2BGR) # Conversion to RGB
    cv2.imwrite(filename, img_rgb)
    return img_rgb
```

Figure 3. Comparison of the code used to store the BMP images on the original C code (above) and the translated Python code (bellow). Only a piece of the original BMP generator code code is shown in the figure. It is worth to mention that the shown lines of the Python code are also reading the image from its bytes.

IV. Results and validation

We tested our solution using three different xed files with GRBG images stored in them. The disk size of the files ranged from 273 MB to 293 MB, approximately. We conduced three tests to each file: a script test, a local test, and a service test.

In the script test, only the extractor was evaluated. Our objective in this test was to obtain the same images as the ones extracted by the original code, but colored as result of our post-processing. In the local test, we evaluated both the script and the web service, but the service was executed locally. Our objective then was to check if the service returns the zip file with the extracted images. Finally, in the web service we evaluated both the script and the web service after being published on Azure platform, using the interface to make requests to the service. Our objective was to check if the code was executing as expected on the Azure platform.

We were successful in all the script and local tests, obtaining the expected images as shown in Figure 6 in the script tests and obtaining these images in a zip file in the local tests. In the service tests, in the other hand, the function returned the zip file just as in the local tests but sometimes an unhandled server error was sent as response to the requisitions. We discovered that the error was due to the high memory consumption of the developed function. Due to that, the service may raise an “OutOfMemory” exception sometimes. We believe that the chances of this error occur are higher when two or more

```c
// Read an index entry (xed_index_entry_t)
static int XedReadIndexEntry(xed_reader_t* reader, xed_index_entry_t* indexEntry)
{
    // If reading to nowhere, skip entry
    if (indexEntry == NULL)
    {
        fseeko64(reader->fp, 24, SEEK_CUR);
        return XED_OK;
    }

    indexEntry->frameFileOffset = fget_uint64(reader->fp);  // @ 0 (e.g. 0x000000004c002bfc, can point to first frame)
    indexEntry->frameTimestamp = fget_uint64(reader->fp);   // @ 8 (e.g. 0x000000038f84d534, or 0 if none)
    indexEntry->dataSize = fget_uint32(reader->fp);          // @16 (e.g. 614400)
    indexEntry->dataSize2 = fget_uint32(reader->fp);         // @20 (e.g. 614400)
    return XED_OK;
}
```

```python
class xed_index_entry_t:
    def __init__(self, xed_file):
        self.frame_file_offset = read_int(xed_file,8) # @ 0 (e.g. 0x000000004c002bfc, can point to first frame)
        self.frame_timestamp = read_int(xed_file,8)   # @ 8 (e.g. 0x000000038f84d534, or 0 if none)
        self.data_size = read_int(xed_file,4)          # @16 (e.g. 614400)
        self.data_size2 = read_int(xed_file,4)        # @20 (e.g. 614400)
```

Figure 4. Comparison between the code related to assign values to the C structures' fields on the original extractor (above) and to the Python classes on the translated code (bellow).

```c
// Allocate global index
reader->globalIndex = (xed_index_t **)malloc(sizeof(xed_index_t *) * maxEvents);
if (reader->globalIndex == NULL)
{
    fprintf(stderr, "ERROR: Problem allocating global index entries (%d)\n", maxEvents);
    return XED_E_OUT_OF_MEMORY;
}

// Create global index
reader->totalEvents = 0;
```

```python
# Allocate global index
self.global_index = []

# Create global index
self.total_events = 0
```

Figure 5. Pointers and memory management present from the original extractor were replaced by references and lists.

requisitions are made in a close time interval (or at the same time) to the service.

We also collected metrics regarding the time consumption. We sent the xed files via our web interface and measured the files' upload time and server processing time. These metrics are present in Table I.

We also measured the execution time of our extractor at Azure platform to evaluate the extractor script time perfor-

Table I
VoXED Web Serive Time Metrics

File	Size (MB)	Upload (s)	Server Processing (s)
1	293	49.244	38.417
2	288	48.162	36.219
3	273	55.194	35.416

mance. The results are present in Table II.

Table II
IMAGES EXTRACTION TIME

File	Size (MB)	Extraction (s)
1	293	1.492
2	288	1.652
3	273	0.754

Analyzing the results of Table I and II, we noticed that the extractor script produces an almost irrelevant impact on the overall processing time of the server. The most time-consuming operations are probably internal Azure operations and the creation and read of the zip file.

Xed files that store RGB images were not available for us during the conducing of this research. Thus, tests were not conduced with files of that type.

V. CONCLUSION

In this paper, we proposed an easy-to-use web service capable of extracting images from xed files. This tool improves the previous ones by facilitating the usage by individuals with low expertise in programming. We hope that the developed tool helps future research that uses data previously captured from the Kinect V1 device, making it easier to extract them.

Although the web service hosted on Azure fails in retrieving the images due to memory limitations sometimes, it still responds with the extracted images in other cases. Furthermore, the developed extractor in the Python language can be used locally or be integrated with other applications to extract the images from xed files regardless of the web service.

Future works include fixing the memory errors of the web service and expanding the tool with a local application that can be executed locally without command line. The extractor can also be further improved by the addition of the capability to extract depth information and by adding support to xef files, generated by Kinect V2 devices using Kinect Studio 2.0 version.

ACKNOWLEDGMENTS

The authors would like to thank Daniel Jackson, for developing the original version of the extractor in the C programming language.

REFERENCES

[1] A. Aloba, G. Flores, J. Woodward, A. Shaw, A. Castonguay, I. Cuba, Y. Dong, E. Jain, and L. Anthony, "Kinder-gator: The uf kinect database of child and adult motion." in *Eurographics (Short Papers)*, 2018, pp. 13–16.

[2] Y. Zhou, Y. Zhe, X.-d. Xu, J.-s. Zhai, and W. He, "Practice research of classroom teaching system based on kinect," in *2020 15th International Conference on Computer Science & Education (ICCSE)*. IEEE, 2020, pp. 572–576.

[3] Y. Tokuyama, R. J. Rajapakse, S. Yamabe, K. Konno, and Y.-P. Hung, "A kinect-based augmented reality game for lower limb exercise," in *2019 International Conference on Cyberworlds (CW)*. IEEE, 2019, pp. 399–402.

[4] Y. Luo, T. Wang, A. Zhu, Z. Wang, G. Shan, and H. Snoussi, "Unmanned trolley control based on kinect," in *2018 Chinese Automation Congress (CAC)*. IEEE, 2018, pp. 2218–2222.

[5] R. Hong, Z. Wu, T. Zhang, Z. Zhang, A. Lin, X. Su, Y. Jin, Y. Gao, K. Peng, L. Li *et al.*, "Preliminary verification of a kinect-based system for evaluating postural abnormalities in patients with parkinson's disease," *Parkinsonism & Related Disorders*, vol. 113, 2023.

[6] S. Das, A. Adhikary, A. A. Laghari, and S. Mitra, "Eldo-care: Eeg with kinect sensor based telehealthcare for the disabled and the elderly," *Neuroscience Informatics*, p. 100130, 2023.

[7] J. C. Chow and D. D. Lichti, "Photogrammetric bundle adjustment with self-calibration of the primesense 3d camera technology: Microsoft kinect," *IEEE Access*, vol. 1, pp. 465–474, 2013.

[8] M. J. Landau, B. Y. Choo, and P. A. Beling, "Simulating kinect infrared and depth images," *IEEE transactions on cybernetics*, vol. 46, no. 12, pp. 3018–3031, 2015.

[9] T. N. Syed, L. Jizhan, Z. Xin, Z. Shengyi, Y. Yan, S. H. A. Mohamed, and I. A. Lakhiar, "Seedling-lump integrated non-destructive monitoring for automatic transplanting with intel realsense depth camera," *Artificial Intelligence in Agriculture*, vol. 3, pp. 18–32, 2019.

[10] A. Z. K. Frisky, A. Harjoko, L. Awaludin, A. Dharmawan, N. G. Augoestien, I. Candradewi, R. M. Hujja, A. Putranto, T. Hartono, Y. Suhartono *et al.*, "Registered relief depth (rrd) borobudur dataset for single-frame depth prediction on one-side artifacts," *Data in Brief*, vol. 35, p. 106853, 2021.

[11] N. G. S. S. Srinath, A. Z. Joseph, S. Umamaheswaran, C. L. Priyanka, M. Nair, and P. Sankaran, "Nitcad-developing an object detection, classification and stereo vision dataset for autonomous navigation in indian roads," *Procedia Computer Science*, vol. 171, pp. 207–216, 2020.

Figure 6. Sample images extracted from the tested xed files. The results from the original code we based on are shown in the left. In the right are presented the images extracted by our solution.