panoMosaics: Improving Object Detection Model Debugging with Panoramic Mosaics

Erin McGowan, Parikshit Solunke, and Kora Stewart Hughes

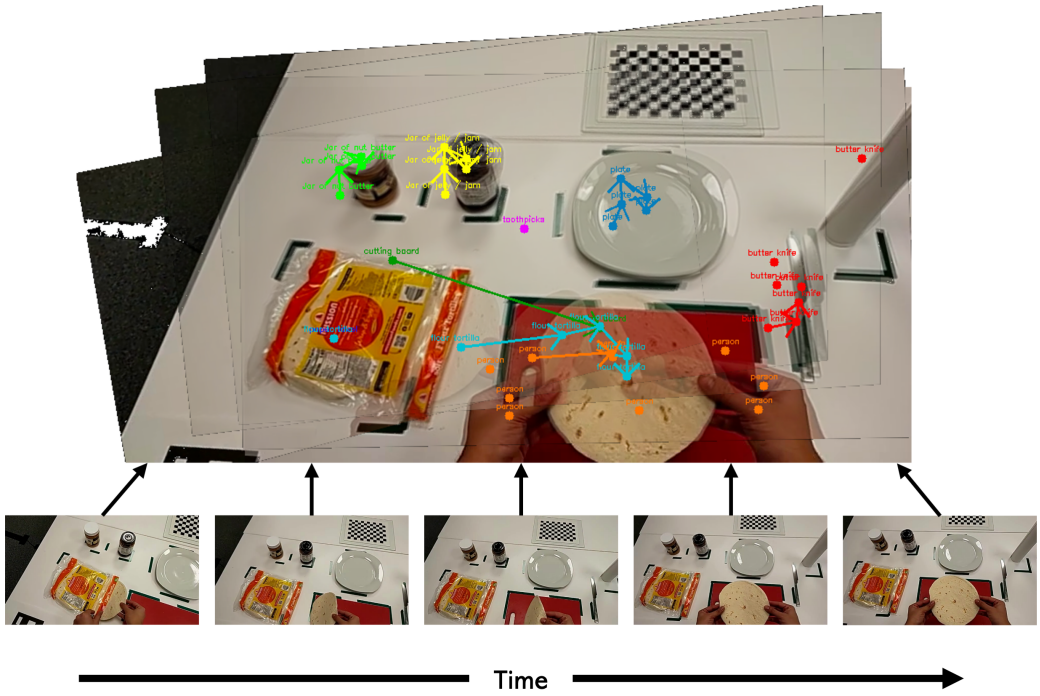

Fig. 1: *panoMosaics* is a Python library for creating panoramic mosaic displays of object detection model outputs.

Abstract— Traditional methods of visualizing and evaluating the output of object detection models are limited to capturing objects within a single frame and timestep, and thus fail to capture the temporal and spatial context that is often necessary for various domain applications. We propose *panoMosaics*, an enhanced approach for visualizing object detection that maximizes user understanding of model performance. The proposed visualization harmonizes time-variant features in a panoramic mosaic display that expands the field of view and facilitates object detection model debugging through augmented bounding boxes. We enhance the standard bounding box visualization by leveraging object trails with different colors and gradients to convey object positions over multiple timesteps. We also present a faster algorithm for panorama stitching than previous work. We demonstrate the effectiveness of *panoMosaics* via a use case by contrasting our approach against multiple baselines. The *panoMosaics* library is pip-installable and available at `https://github.com/egm68/panoramic-mosaics`.

Index Terms—Data visualization, Object detection, Augmented reality.

1 Introduction

Traditional methods of evaluating dynamic object detection models face an issue of generalizability over time. Standard bounding box methods are intuitive and contain expressive power, but are limited to capturing objects within a single frame and time-step, sacrificing descriptive capabilities.

- *Erin McGowan is with New York University. E-mail: erin.mcgowan@nyu.edu*
- *Parikshit Solunke is with New York University. E-mail: pss442@nyu.edu*
- *Kora Stewart Hughes is with New York University. E-mail: khughes@nyu.edu*

Alternative visualization techniques using invariant features can make up for these shortcomings by adding more detailed information, yet lack the intuitive presentation and digestibility of standard image-based models. Our proposed visualization harmonizes time-variant features in a panorama-centered visualization to maximize user understanding and information conveyed.

This issue arose during the development of ARGUS [4], an interactive visual analytics system that allows developers of intelligent assistive AR systems to conduct seamless analysis of complex datasets by synthesizing multiple data streams at different scales, dimensions, and formats acquired during performance time. Through the use of spatiotemporal visualization widgets, ARGUS allows for retrospective analysis and debugging of historical data generated by ML models for AR assistants. Specifically, ARGUS employs an object detection model to recognize all objects in an image and their positions. To inspect and debug these outputs, ARGUS contains a video player component that identifies the spatial location of detected objects over time. This component allows the user to toggle between two views: 1) the raw main camera video stream and 2) a sequence of panoramic mosaics generated from this main camera stream. The first view of the video player component displays the raw main camera video stream collected by an AR headset (in our case, a Microsoft HoloLens 2 [1]). This allows the user to note specific frames where object detection failed or yielded unexpected results (we note that we hereafter use the term "user" to refer to a user of the ARGUS system and "performer" to refer to a person wearing an AR headset while performing a task).

The main camera of the HoloLens has a limited field of view. Often objects that the performer sees at a given timestep cannot be seen in the frame of the raw main camera video at that timestep. The goal of the second view in ARGUS's video player is to aggregate frames into a series of panoramic mosaics, capturing a broader scope of what the performer sees at each timestep. A preliminary investigation (see Fig. **??**) has shown that these panoramic mosaics expand the view of the scene significantly and reveal objects within the field of view of the performer that were missed by the main camera of the HoloLens.

However, the panoramic mosaic stitching algorithm outlined in [4] is neither efficient nor the most intuitive way to display this data. We propose an updated design with additional features that both expands upon the scope of the main camera view as well as facilitates object detection model debugging through the innovative use of augmented bounding boxes. Our contributions are as follows:

- An improved panoramic mosaic display that is updated at each timestep, with each new frame being stitched into its respective position on the larger backdrop of the entire scene (of what the performer has viewed thus far) rather than displaying consecutive panoramas stitched from different windows of the original main camera video.
- Implementation of a faster algorithm for panorama stitching than the one used in [4].
- Enhancement of the standard bounding box visualization, which conveys object positions at a single moment in time, to a visualization that is able to convey object positions over multiple timesteps in a single view using trail and color encodings and various shapes in combination with panorama stitching.

2 Related Work

Panoramic image stitching is an active research area in which a variety of approaches have been proposed over the years. Early approaches to panoramic image stitching were based on feature matching and transformation estimation. Szeliski and Shum [19] proposed a method for automatic alignment and stitching of overlapping images using feature detection and matching techniques. Lowe [14] developed SIFT, an algorithm that computes a feature vector that is invariant with respect to scale, orientation, and illumination, making it robust to changes in the image due to camera motion, viewpoint changes, or lighting conditions. Brown and Lowe [3] introduced an automatic image stitching algorithm that uses SIFT to detect image features, computes a homography matrix for each pair of images by employing the RANSAC algorithm [7], and uses this matrix to stitch images together. Many applications, including the cross-platform computer vision library OpenCV [2], make use of this technique to perform image stitching.

Subsequent works have focused on the problem of video stitching. Specifically, there have been various works dealing with the problem of stitching videos from multi-camera surveillance systems together [10] [9]. Additionally, there have been efforts to deal with scenes with dynamic elements [15], large exposure differences and considerable motion [6], and parallax handling of moving foregrounds [12]. Video stitching is a computationally costly task and there have been efforts to speed up this process and achieve stitching of multi-camera footage in real-time [21].

Our task poses several unique challenges that diverge from earlier applications. Firstly, we are dealing with egocentric video data where the user performs a complex task, resulting in a scene crowded with moving objects and haphazard camera panning motions. Earlier work on panoramic mosaics has either excluded moving objects in the scene [11] or only dealt with simple and slow camera panning [17]. Furthermore, the HoloLens collects various multimodal sensor data streams, including eye gaze direction, eye gaze origin, and inertial measurement unit data, which can help comprehend the relative position of each frame in space and its motion. However, effectively stitching a mosaic from an egocentric camera stream while accounting for scene and camera motion, and integrating the multimodal data streams requires significant computational resources and intelligent solutions.

3 Background

3.1 Perceptually-enabled Task Guidance

The Perceptually-enabled Task Guidance (PTG) program [5] is a Defense Advanced Research Projects Agency (DARPA) initiative that focuses on developing AI technologies for task guidance. It aims to expand the skillset of users and reduce their errors while performing complex physical tasks. Wearable sensors, such as head-mounted cameras and microphones, enable an Augmented reality (AR) assistant to see and hear what the user does, thus allowing the headsets

to provide feedback through speech and aligned graphics that aide in task execution.

3.2 ARGUS

ARGUS [4] is a system created by researchers on the PTG team at New York University (NYU). It allows developers of assistive AR systems to debug the ML models associated with those systems. The user interface of the offline mode of ARGUS contains three components: the Data Manager, the Spatial View, and the Temporal View. The Data Manager allows the user to apply filters and select a session recorded by a performer using the HoloLens. The Spatial View contains a point cloud representing the physical environment, 3D points for eye and hand positions, and gaze projections and heatmaps. We will be using the Temporal View, which includes features for object detection model debugging (see Figure 3), as a baseline for our work.

The Temporal View contains the Model Output Viewer, which displays the output of the machine learning models (reasoning and perception) necessary for an AR assistant to function. It also contains a video player component that identifies the spatial location of detected objects over time. This component allows the user to toggle between two views: 1) the raw main camera video stream and 2) a sequence of panoramic mosaics generated from this main camera stream.

Fig. 2: The ARGUS temporal viewer, which includes (A) a video player component that displays the RGB camera video stream with bounding boxes output by the DETIC object detection model and (B) a heatmap of model detection confidence for each object over time.

The first view of the video player component displays the raw main camera video stream collected by HoloLens. This allows the user to note specific frames where object detection failed or yielded unexpected results. Our work enhances the panoramic mosaic view to incorporate more of the scene and more clearly articulate the movement of objects within it over time.

3.3 Data

We obtained recordings of AR assistants system users performing a task (in this case following a recipe) from the PTG team at NYU. This data is stored in an NYU server and is accessible via *ptgctl*, a Python Library and Command Line tool for interacting with the PTG API and data streams [18].

The HoloLens 2 has a primary RGB camera, 4 grayscale peripheral cameras, an infrared depth camera, and an IMU containing an accelerometer, gyroscope, and magnetometer. With research mode enabled [20], it is possible to stream data from all these sensors. Hand-tracking and eye-tracking data are also streamed. The eye tracking data consists of 3D gaze origin positions and directions. The hand tracking data consists of 26 joint points for each hand. Each joint point contains a 3D position and a quaternion that indicates the orientation. Performer sessions can vary in size, for instance, the recording of a simple recipe (preparing pinwheels) usually takes about 6 minutes and results in around 600 MB data.

The object detection model currently integrated with ARGUS, DETIC [22], generates bounding boxes based on the main RGB camera stream. Therefore, we only incorporate the main camera stream into our stitched panoramic mosaic. While it is possible to stream data from the primary camera at higher framerates and resolutions, due to resource constraints, our stream has a framerate of 7.5 fps at a resolution of 760×428.

4 Methods

We propose *panoMosaics*, a Python library that allows users to evaluate object detection models by generating panoramic mosaics of videos. We build upon the panoramic mosaic view described in [4] by integrating additional visual features into the existing ARGUS Temporal View that aid in identifying frame dropouts and object-detection accuracy between time steps.

4.1 Panoramic Mosaic Construction

For each frame we intend to stitch together, we compute SIFT keypoints and descriptors [13]. We then choose a "base frame;" we will warp all other frames to the coordinate plane of this frame. We match the SIFT descriptors between each frame and the base frame using a Fast Library for Approximate Nearest Neighbors (FLANN)-based matching [16] function from OpenCV [2]. We subsequently use Lowe's ratio test [13] to filter for valid matches, or matches which actually correspond to the same real world location in each frame. After this, we use these matching features to compute a homography matrix for each non-base frame. We note that it is only mathematically possible to calculate this homography matrix if we have at least four matching features between a given frame and the base frame. If a frame does not have at least four features that match the features of the base frame, said frame is discarded. We then translate the base frame to the center of a transparent background panel. We apply

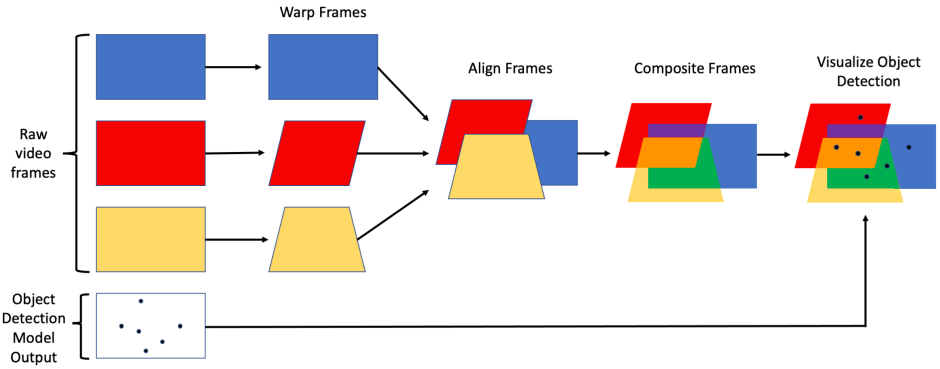

Fig. 3: Overview of *panoMosaics* architecture.

this same translation to the homography matrices for each non-base frame so that all of the frames will be aligned on the larger background panel. If a translated non-base frame will exceed the bounds of this background panel, we expand the width and height of the background panel. We are then able to multiply each non-base frame by its respective homography matrix, warping them (one at a time) into their correct positions on the background panel.

With each new frame N that is warped into position on the background panel, we perform alpha (α) compositing to blend the new frame into the existing panoramic mosaic of the previous $N-1$ frames. We apply equal weight ($\alpha = 0.5$) to the new frame N and the existing panoramic mosaic of the previous $N-1$ frames, meaning frame N (which represents the current timestep if we choose to save all intermediate panoramic mosaics as a video during construction) is the most visible in the resulting panorama. The further a frame is (in time) from frame N, the more faded it will appear in the final panorama. We also only perform alpha compositing on the portion of frame N which overlaps with the existing panoramic mosaic of the previous $N-1$ frames; the rest of the image remains opaque (i.e. the alpha channel of each pixel in the remainder of the image is set to 255). We compute the bounds of this overlapping region using Shapely [8], a BSD-licensed Python package for manipulation and analysis of planar geometric objects. We do so by creating one Shapely polygon (which is represented by its corners) for each new frame N and another polygon for the shape of the existing panoramic mosaic of the previous $N-1$ frames. We then use the Shapely *intersection* function to create a third polygon consisting of the overlap between the first two polygons; our alpha compositing function is applied to any pixels contained within this third overlap polygon. We then take the union of the polygon for frame N and the polygon for the previous $N-1$ frames using the Shapely *unary_union* function. The resulting polygon becomes the new polygon for the existing panorama once a new frame N is introduced. Performing alpha compositing on the overlapping region alone allows us to significantly decrease the stitching time for each panorama.

The panorama mode of the ARGUS video player component displays a series of panoramic mosaics that are resized down to the original resolution of the main camera video. These panoramic mosaics are generated by dividing the raw main camera video stream into windows of equal length and sampling a small number of frames from each window. Our updated design consists of a larger background (i.e. larger than the original resolution of the raw main camera video) onto which each new frame is stitched. Instead of displaying a series of panoramic mosaics, this larger mosaic is updated to include the current frame at each timestep. This not only provides a more comprehensive view of the scene, but also a more intuitive view of how the entire scene changes over time. Our updated implementation of the alpha compositing function is also much faster than the alpha compositing function used in ARGUS, and in turn vastly improves the overall stitching time for a each panorama.

4.2 Visualizing Object Detection

To visualize the results of the Object Detection model, we allow the user to pick from a range of mark options that indicate the position and size of a detected object (see Figure 4). These mark options include classic "Boxes," "Brackets," "Frames" (which only draws the corners of each bounding box region), and "Center Dots" (which draws only a dot at the center of each bounding box region). The "Brackets," "Frames," and "Center Dots" mark options each mitigate the clutter present in standard bounding box visualizations (especially those with many objects) to varying degrees. We also implemented the mark options "Center Dots Lined" and "Arrow," which each draw a dot at the center of the bounding box region and then connect them, to better indicate the direction of object movement over time.

If duplicate objects (i.e. two objects with the same label that appear in the same timestep) are present and a mark option which draws connects between timesteps (e.g. arrows) is specified, we connect the duplicate objects by spatial proximity. For instance, if the model detects "flour tortilla" once at timestep 1 and twice at timestep 2, we draw the arrow from the specified location in timestep one to the nearest of the two locations from timestep 2.

Additionally, the user can also choose from two "color schemes," which determine which attribute of the scene color denotes, for clarity of analysis. The first implementation is an object identification option where the color of the chosen mark indicates the object name or class. The second implementation is a time-variant option where the color and opacity of the mark indicates how far in the past the object was identified.

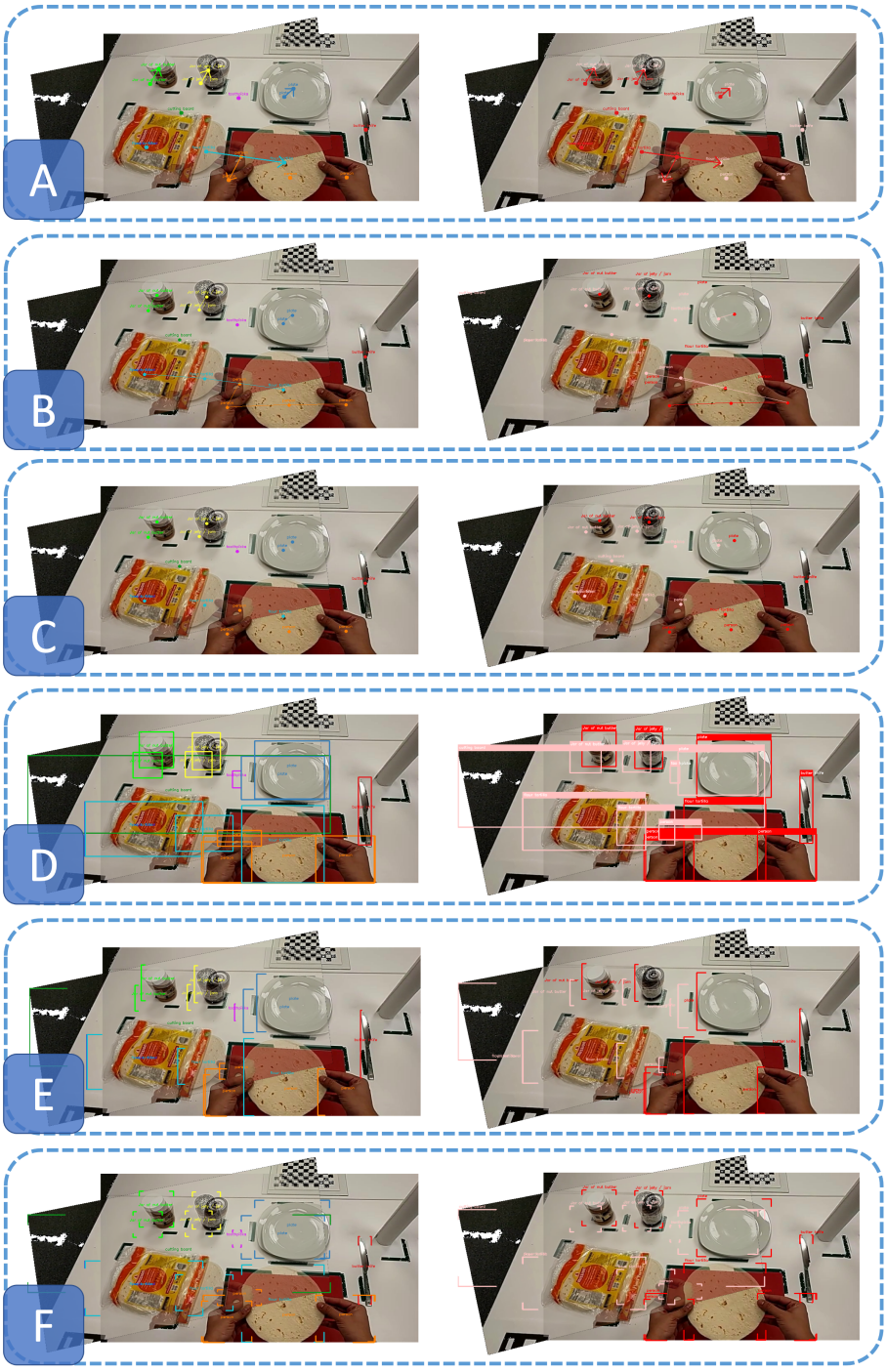

Fig. 4: Examples of each object detection visualization style in the *panoMosaics* library, including (A) Arrows, (B) Center Dots Lined, (C) Center Dots, (D) Boxes, (E) Brackets, and (F) Frames. Examples in the left column use color to denote object label, while examples in the right column use color to denote timestep.

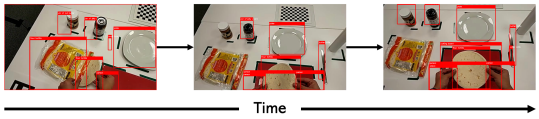

Fig. 5: Three frames from our clip of interest annotated with labeled bounding boxes generated by the DETIC object detection model.

These enhancements over the standard bounding boxes, as well as the ability to fine-tune exactly what is visualized, allow for finer-grained analysis of the object detection model outputs.

5 Use Case

To demonstrate the effectiveness of *panoMosaics*, we will compare the process of object detection model debugging using a video with standard bounding boxes, using the ARGUS Temporal View, and using our tool. Suppose we are ML model developers who want to debug an object detection model. In our case we will use DETIC [22], as this is the object detection model currently integrated with ARGUS. We want to find moments where the object detection model did not detect an object that we know to be in the scene at that time and determine why. We also want to pinpoint any other instances of unexpected model behavior and determine their causes.

Debugging an object detection model using standard bounding boxes. The current standard method of evaluating the performance of an object detection model involves drawing bounding boxes denoting the detected location of each object on an image or each frame of a video (see Figure 5). However this method only shows the location of each object one timestep at a time. In many cases, particularly when the camera is part of an AR headset, this approach fails to capture the full scope of what the user wearing the headset can see at that same timestep. This is certainly true for the HoloLens 2 central RGB camera, which has a limited field of view. This view can also be cluttered and confusing if there are too many objects.

Debugging an object detection model using ARGUS Temporal View. The ARGUS Temporal View contains a video player component that also utilizes standard bounding boxes to visualize object detection model output, and is therefore subject to many of the same limitations discussed above. To provide additional temporal context, the ARGUS Temporal View also includes a heatmap which displays model detection confidence for each object at each frame of a given recording (see Figure 3). However, on a general level this approach still fails to capture the spatial context of the environment surrounding a given frame. Moreover, with respect to our domain of interest (object detection applied to egocentric video data recorded with a camera that has a limited field of view), this spatial context is especially crucial as it has broader implications for user behavior. This spatial context allows developers to not only determine which objects are visible to the camera at a given timestep, but also to infer which objects are visible to the user at that timestep (and, most importantly, any differences between those two groups of objects).

Debugging an object detection model using *panoMosaics*. Instead, we can accomplish this task using our tool *panoMosaics*. We see that our clip of interest encompasses five timestamps from the object detection output, so we select the five frames that are closest to these timestamps and stitch them together. Since we are analyzing several frames at once and want to avoid unnecessary clutter that may obscure parts of the scene, we choose the relatively unobtrusive arrow mode for the visualization style (see Figure 4). We also use color to denote object label. The resulting panoramic mosaic can be seen in Figure 6. We observe immediately that it is much easier to tell object trajectories apart with this view than it is in a standard bounding box view.

We can also make several observations about the performation of the obect detection model using this panoramic mosaic. We see that the model detected an object that is not actually present in the scene: “toothpicks” in pink. We can guess that this happened because of the tape on the table. The model also incorrectly identifies the location of objects that are actually present at some timesteps, like the “cutting board” in green. This prompts us to inspect the upper left and lower right object location bounds output by the DETIC model, and we find that the model classified the entire first frame of the scene as the cutting board. Moreover, the model failed to identify an object entirely: paper towels. We can observe that this happened because the portion of the paper towels in any one frame was not large enough to be detected - we can only see a significant portion of the object in the panorama.

With respect to duplicate object handing, we see that the model correctly identifies duplicates of some objects, like “flour tortilla” in teal. However, the model also incorrectly detects duplicates of some objects. We can determine this is the case with “butter knife” in red, since the number of red dots is greater than the number of frames we stitched together, implying a duplicate must be present in at least one frame. We see a combination of correct and incorrect model behavior when it comes to the detection of "person" (shown in orange). In most frames, the model correctly identifies two separate locations for the two hands of the user. However, we also see that person is incorrectly detected in three locations for one frame.

Baseline Comparison. Compared to standard bounding boxes and the ARGUS temporal view, we see that *panoMosaics* provides the spatial and temporal context that one-frame-at-a-time visualizations lack (see Figure 1). Our tool also allows the user to customize the visualization based on the focus of their task.

6 Conclusion

The proposed library, *panoMosaics*, provides a versatile visualization that add time-variant insights that aid in the analysis of object detection algorithms. By combining panoramic stitching and innovative object identification marks, the user is given insight into the detection algorithm’s accuracy while retaining spatial context.

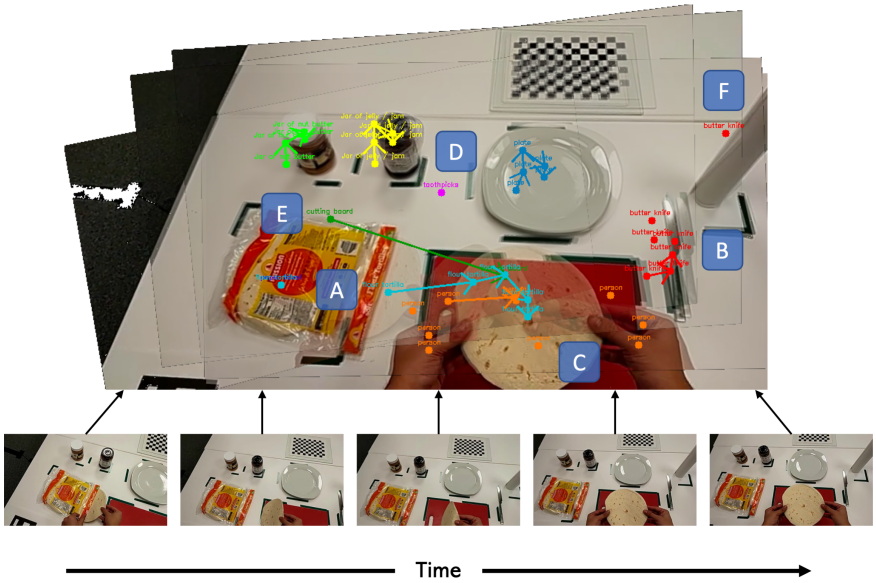

Fig. 6: A visualization generated using *panoMosaics* that shows (A) Correctly detected duplicate objects (flour tortillas). (B) An object that was incorrectly detected as a duplicate (butter knife). (C) An object that was correctly detected as a duplicate at some timesteps, but the number of duplicates was incorrect (person). (D) An object that was detected but is not actually present in the scene (toothpicks). (E) An object detected at the wrong location (cutting board). (F) An object the model failed to detect (paper towels).

	Standard Bounding Boxes	**ARGUS Temporal View**	***panoMosaics***
Field of View	Single timestep, limited by HoloLens camera	Single timestep, limited by HoloLens camera	Up to entire scene
Context of current frame	None	Temporal	Temporal and spatial
Object Detection Visual - ization	Single color, box overlap can be cluttered or confusing	Augments classic bounding boxes with heatmap for object detection confidence	Specified object outputs and trajectories, color scheme denotes objects or time

Table 1: Comparison of standard bounding boxes, ARGUS Temporal View, and *panoMosaics* for object detection output visualization.

7 Limitations and Future Directions

Though our visualization is capable of traversing through a large range of time steps the space we have to accurately convey the position, scale, and movement direction of an object is limited. This means that ultimately, clutter scales with the number of detected objects as well as the number of time steps incorporated into the pano scene. Depending on the identification mark used, this issue is also exacerbate.

Additionally, one of the major challenges we face is handling motion within recordings while stitching images into a panorama. The faster the motion, the more blurry an image may be and the less cleanly it will integrate into other displayed panoramic images. This distortion can be somewhat mitigated with computer vision techniques, however the solution that allows for the most comfortable solution is to only stitch frames taken at low velocities, which provides another dimension of challenge in data collection (or to increase the framerate of the HoloLens RGB camera, which introduces hardware temperature issues).

A similar challenge is connecting objects between timesteps when duplicates are present. Limitations of our system in its current state include high processing times and lack of interactivity. These limitations mean that our system may not generate panoramas quick enough to be able to provide real-time feedback.

We intend to improve upon the performance time in order to allow our tool to generate results on the fly. To do so, we plan to migrate from using OpenCV in Python to using

OpenGL fully leveraging parallel processing on the GPU. Improving the speed will also enable us to add interactive elements to our tool allowing for better analysis. Our tool is currently available as a pip-installable python library. However, to integrate it better with the workflow in ARGUS, we plan to package it as an npm-installable module.

REFERENCES

[1] Hololens 2-overview, features, and specs: Microsoft hololens. 2

[2] G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000. 2, 3

[3] M. Brown and D. G. Lowe. Automatic panoramic image stitching using invariant features. *International journal of computer vision*, 74:59–73, 2007. 2

[4] S. Castelo, J. Rulff, E. McGowan, B. Steers, G. Wu, S. Chen, I. Roman, R. Lopez, E. Brewer, C. Zhao, and et al. Argus: Assistive visualization of human-ai collaboration for task guidance in augmented reality. *In Submission.*, 2023. 2, 3

[5] B. Draper. Perceptually-enabled task guidance (ptg), 2022. 2

[6] A. Eden, M. Uyttendaele, and R. Szeliski. Seamless image stitching of scenes with large motions and exposure differences. In *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*, vol. 2, pp. 2498–2505, 2006. doi: 10.1109/CVPR.2006.268 2

[7] M. A. Fischler and R. C. Bolles. Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Commun. ACM*, 24(6):381–395, jun 1981. doi: 10.1145/358669.358692 2

[8] S. Gillies, C. van der Wel, J. Van den Bossche, M. W. Taves, J. Arnott, B. C. Ward, and others. Shapely, Jan. 2023. doi: 10.5281/zenodo.5597138 4

[9] B. He and S. Yu. Parallax-robust surveillance video stitching. *Sensors*, 16(1), 2016. doi: 10.3390/s16010007 2

[10] B. He, G. Zhao, and Q. Liu. Panoramic video stitching in multi-camera surveillance system. In *2010 25th International Conference of Image and Vision Computing New Zealand*, pp. 1–6, 2010. doi: 10.1109/IVCNZ.2010.6148851 2

[11] C.-T. Hsu and Y.-C. Tsan. Mosaics of video sequences with moving objects. *Signal Processing: Image Communication*, 19(1):81–98, 2004. doi: 10.1016/j.image.2003.10.001 2

[12] M. U. Kakli, Y. Cho, and J. Seo. Minimization of parallax artifacts in video stitching for moving foregrounds. *IEEE Access*, 6:57763–57777, 2018. doi: 10.1109/ACCESS.2018.2871685 2

[13] D. G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2):91–110, Nov. 2004. doi: 10.1023/b:visi.0000029664.99615.94 3

[14] G. Lowe. Sift-the scale invariant feature transform. *Int. J*, 2(91-110):2, 2004. 2

[15] A. Mills and G. Dudek. Image stitching with dynamic elements. *Image and Vision Computing*, 27(10):1593–1602, 2009. Special Section: Computer Vision Methods for Ambient Intelligence. doi: 10.1016/j.imavis.2009.03.004 2

[16] M. Muja and D. G. Lowe. Fast approximate nearest neighbors with automatic algorithm configuration. In *International Conference on Computer Vision Theory and Applications*, 2009. 3

[17] D. Steedly, C. Pal, and R. Szeliski. Efficiently registering video into panoramic mosaics. In *Tenth IEEE International Conference on Computer Vision (ICCV'05) Volume 1*, vol. 2, pp. 1300–1307. IEEE, 2005. 2

[18] B. Steers, S. Chen, I. Roman, J. Lin, H. Vo, and J. Rulff. ptgctl. `https://github.com/VIDA-NYU/ptgctl`, 2023. 3

[19] R. Szeliski and H.-Y. Shum. Creating full view panoramic image mosaics and environment maps. In *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pp. 251–258, 1997. 2

[20] D. Ungureanu, F. Bogo, S. Galliani, P. Sama, X. Duan, C. Meekhof, J. Stühmer, T. J. Cashman, B. Tekin, J. L. Schönberger, P. Olszta, and M. Pollefeys. Hololens 2 research mode as a tool for computer vision research, 2020. 3

[21] S.-H. Yeh and S.-H. Lai. Real-time video stitching. In *2017 IEEE International Conference on Image Processing (ICIP)*, pp. 1482–1486, 2017. doi: 10.1109/ICIP.2017.8296528 2

[22] X. Zhou, R. Girdhar, A. Joulin, P. Krähenbühl, and I. Misra. Detecting twenty-thousand classes using image-level supervision, 2022. 3, 6