ChatStitch: Visualizing Through Structures via Surround-View Unsupervised Deep Image Stitching with Collaborative LLM-Agents

Hao Liang, Zhipeng Dong, Kaixin Chen, Jiyuan Guo, Yufeng Yue, Mengyin Fu, Yi Yang*

A. Contents

In this document, we provide the following supplementary contents:

- Discussion of Multi-image Stitching Tasks (section B).
- More Details about Agents (section C)
- More Details about Experimental settings (section D).

B. Discussion of Multi-image Stitching Tasks

Most stitching algorithms focus on stitching two images. Theoretically, multi-image stitching can be achieved by performing multiple pairwise stitching. However, the final stitching effect is influenced by various factors such as the overlapping rate of the input images and the design of the stitching algorithm. In this section, we discuss the problems that may arise when extending a two-image stitching algorithm to multi-image stitching.

SoTA two-image stitching methods (such as APAP [1], AANAP [2], LPC [3], UDIS [4], UDIS2 [5], etc.) typically select one image as the reference image and another image as the target image, then warp the target image to achieve alignment. If the multiple images to be stitched satisfy the global-overlapping condition, it is only necessary to select a common reference image, and then take turns to warp all other images as the target image, to achieve a good multi-image stitching result. In this case, the multi-image stitching problem can be transformed into a two-image stitching problem under no cumulative distortion.

However, the surround-view images in autonomous driving scenarios usually satisfy the non-global-overlapping condition. In this case, it is necessary to first stitch two adjacent images together, and then stitch them with the next image, and so on. This recursive stitching approach will cause the image distortion (especially projective distortion) to accumulate gradually during the stitching process, resulting in significant distortion in the final result. In addition, images that satisfy the non-global-overlapping condition usually have a lower overlap rate. Existing SoTA methods already show serious distortion when stitching two images, and may even fail to complete multi-image stitching, as shown in fig. 8 and fig. 7.

In this paper, our goal is to achieve multi-image stitching under the non-global-overlapping condition in autonomous driving scenarios. We maintain spatial consistency under a large field of view through cylindrical projection and combine it with rectangular warping constraints to reduce projective distortion. Finally, we calculate the global control point motion through our motion propagation strategy to achieve globally consistent multi-image stitching.

*This work was supported by the National Natural Science Foundation of China under Grant 62233002, Grant 92370203 and Development Program of China under Grant 2022YFC2603600.(Corresponding author: Yi Yang)

The authors are with School of Automation, Beijing Institute of Technology, Beijing, 100081, China.

C. More Details about Agents

In ChatStitch, agents are composed of specially designed LLM modules paired with corresponding functions. They have equal access to system data and the ability to manage their own parameters. This section provides a detailed overview of agent design and typical methods.

A. Example of agent prompt

We employ the GPT-4o model as the language interface for the agent modules. Each agent's prompt includes its role definition, a basic description of its corresponding function, and descriptions of both input and output data. To optimize the performance of the LLM and its corresponding function, the LLM is tasked with translating complex instructions into parameters stored in a JSON file, while the function reads this JSON file to execute tasks under the LLM's parameter control. Additionally, examples are included in the design to clarify specific input and output formats.

It is important to note that, to enhance the reference to external data assets, we have pre-loaded content from the 3D Asset Bank into the LLM module as selectable parameters. This approach allows ChatStitch to adjust its functions according to the actual needs of the scene, thereby enhancing the system's adaptability. An example of a perspective measurement prompt is shown in fig. 1.

B. More Details about 3D Asset Bank

To enhance the utilization of external data assets, we have provided a library of existing vehicle models. The construction of the 3D Asset Bank follows the process outlined below.

1) Photograph the target vehicle while ensuring the following requirements are met: Use a smartphone or camera to capture images by circling around the object. The angle between consecutive shots should not exceed 3°, and ensure there is an overlap between images.
2) Import the collected images into the COLMAP to compute the intrinsic and extrinsic parameters for each image,
3) Utilize the SAM2 model to extract the mask of the target vehicle. By multiplying the original images with the mask images, obtain a set of images featuring only the target vehicle with the background removed.

Profile

Role: "Given the relative poses, box my partner."

Design

#Agent description

"I will give you a description about my partner, which may include its length, width and height, you need to output the box coordinates of my partner."

#Details

"For a box, there are eight coordinates. Each coordinate will be a combination of the three-dimensional measurements: length, width, and height. "

#Return format

"Given my description, return a dictionary in JSON format, with coordinates. "

#Example

"I will give you an example:
<user>: Box the E300 which length is 4, width is 2, and height is 1.8
<agent>: {"coordinates": [
{"x": 0, "y": 0, "z": 0},{"x": 4, "y": 0, "z": 0},
{"x": 4, "y": 2, "z": 0},{"x": 0, "y": 2, "z": 0},
{"x": 0, "y": 0, "z": 1.8},{"x": 4, "y": 0, "z": 1.8},
{"x": 4, "y": 2, "z": 1.8},{"x": 0, "y": 2, "z": 1.8}]}"

Fig. 1: Prompt example of perspective measurement.

Fig. 2: Some part of 3D Asset Bank.

4) Input the processed images along with the parameters calculated by COLMAP into InstantNGP. After training, this process yields a NeRF model for the specific target vehicle.

Following the guidance of the above process, we can flexibly expand and utilize external data assets. Some example models are shown in fig. 2.

C. More Details about Relative Pose Measurement

The LLM of the Relative Pose Measurement Agent receives language instructions from the Task Management Agent and outputs the corresponding vehicle ID parameters. The corresponding function reads the vehicle ID parameters and GPS location information, then calculates the relative pose between the vehicles.

To compute the relative pose between vehicles, the system utilizes a hierarchical coordinate transformation framework, as illustrated in Fig. 3. Several key coordinate systems are defined:

- **Observer Camera Coordinate System (oc)**: Originating from the observer's camera, this coordinate system represents the relative pose of observed objects.
- **Observer Body Coordinate System (ob)** and **Observer Horizontal Coordinate System (ol)**: These systems bridge the observer's camera coordinates with the global frame.
- **Target Vehicle Body Coordinate System (tb)** and **Target Vehicle Horizontal Coordinate System (tl)**: These systems represent the pose of the target vehicle in its local reference frame.
- **Target Vehicle NeRF Space Coordinate System (tn)**: This system is used to describe the target vehicle's pose in the NeRF (Neural Radiance Field) space for rendering or reconstruction purposes.
- **Panoramic Image Pixel Coordinate System (p)**: This system links the target vehicle's pixel position (u, v) in the panoramic image to its corresponding real-world coordinates.

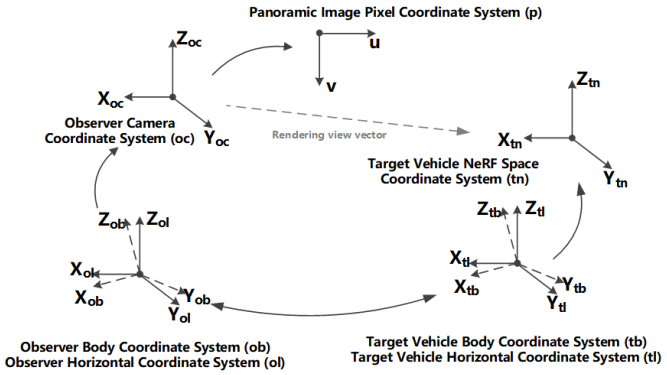

Fig. 3: Illustration of hierarchical coordinate systems used for relative pose computation.

To calculate the relative pose and rendering vector of the target vehicle in the observer's camera coordinate system, the system uses the following known inputs:

- GPS coordinates of both the observer vehicle and the target vehicle.
- The transformation between the Observer Body Coordinate System (ob) and the Observer Camera Coordinate System (oc), provided by the camera's intrinsic and extrinsic parameters.
- The transformation between the Target Vehicle Body Coordinate System (tb) and the Target Vehicle NeRF Space Coordinate System (tn).

The output is the (u, v) pixel coordinates of the target vehicle in the observer's camera image plane and the corresponding rendering direction vector.

1) GPS-Based Global Transformation: Using the GPS coordinates of the observer vehicle and the target vehicle, the relative position of the target vehicle in the Observer Body Coordinate System (ob) is calculated. Denoting the GPS coordinates as $(\text{lat}_o, \text{lon}_o, \text{alt}_o)$ for the observer and $(\text{lat}_t, \text{lon}_t, \text{alt}_t)$ for the target, their relative position $\mathbf{p}_{ob}$ in the observer's body frame is computed via:

$$\mathbf{p}_{ob} = \mathbf{R}_{GPS}(\mathbf{p}_t^{GPS} - \mathbf{p}_o^{GPS}),$$

where $\mathbf{R}_{GPS}$ accounts for the Earth's curvature and local alignment.

2) Transformation to Observer Camera Coordinate System: The position of the target vehicle in the observer's camera coordinate system (X_{oc}, Y_{oc}, Z_{oc}) is obtained using the known extrinsic transformation matrix $T_{oc \leftarrow ob}$ (camera pose with respect to the observer body):

$$\mathbf{p}_{oc} = T_{oc \leftarrow ob} \cdot \mathbf{p}_{ob}.$$

3) Projection to Image Plane: The 3D point (X_{oc}, Y_{oc}, Z_{oc}) is projected onto the image plane using the camera's intrinsic matrix K. The pixel coordinates (u, v) are given by:

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = K \cdot \begin{bmatrix} X_{oc} \\ Y_{oc} \\ Z_{oc} \end{bmatrix},$$

where $K = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}$, and f_x, f_y are focal lengths, while c_x, c_y are the principal point offsets.

4) Rendering Vector Calculation: The rendering vector $\mathbf{d}_{render}$ in the Target Vehicle NeRF Space Coordinate System (tn) is calculated by first transforming the position in the observer's camera coordinate system (X_{oc}, Y_{oc}, Z_{oc}) to the target vehicle's NeRF space using:

$$\mathbf{p}_{tn} = T_{tn \leftarrow tb} \cdot T_{tb \leftarrow oc} \cdot \mathbf{p}_{oc},$$

where $T_{tn \leftarrow tb}$ is the transformation from the target body to the NeRF space, and $T_{tb \leftarrow oc}$ is derived from the relative pose between the two vehicles.
The rendering vector is then normalized as:

$$\mathbf{d}_{render} = \frac{\mathbf{p}_{tn}}{\|\mathbf{p}_{tn}\|}.$$

By following this hierarchical process, the system accurately computes the pixel coordinates (u, v) of the target vehicle in the observer's camera image and determines the corresponding rendering direction vector in the target vehicle's NeRF space.

D. More Details about Experimental Settings

A. System-level Perception Experimental Details

As shown in fig. 5, the target vehicle of the system uses a Baojun E100, while the chase vehicle employs a Baojun E300. The distance between the two vehicles is 5 meters, and the moving speed during the experiment is 10 km/h. The chase vehicle is equipped with a camera model of insta360 Pro 2, which has a diameter of 143 mm and utilizes 6 x 200° F2.4 fisheye lenses. Pose estimation is achieved using integrated navigation, and the photos obtained have a resolution of 8k, with the image format being JPEG.

B. Visual Programming Comparison Experimental Details

We have configured the experimental environment for Visual Programming on a server and successfully run the natural language image editing test. The experimental platform is based on Ubuntu 20.04.2 LTS, with the CUDA 12.2 graphics acceleration library. The system is equipped with an NVIDIA GeForce RTX 3090 Ti GPU and an Intel(R) Xeon(R) Gold 5215 CPU @ 2.50GHz. The experimental environment was created and deployed using Anaconda, with Python 3.10 as the programming language. The dependencies installed are listed in table I. We replaced the original text-davinci-003 model with GPT-4o and modified the API call format to adapt to GPT-4o, as detailed in fig. 6.

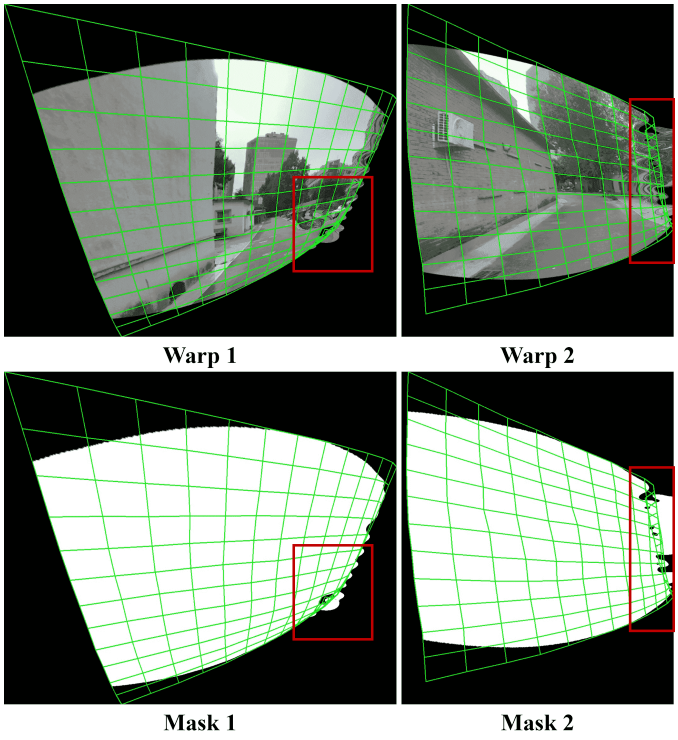

Fig. 4: Example diagrams of the grid folding phenomenon.

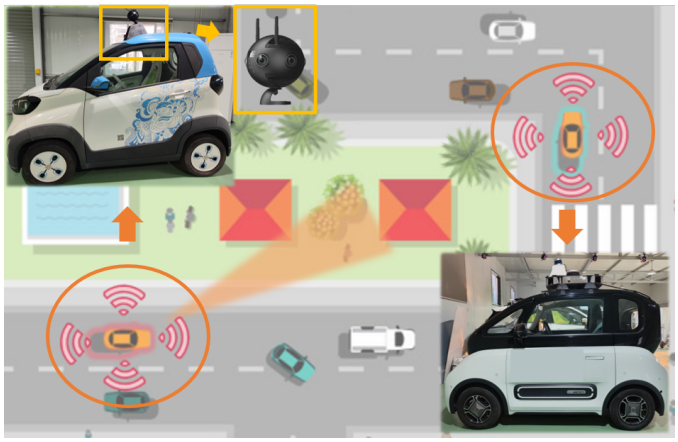

Fig. 5: Experimental layout diagram

C. SV-UDIS Experimental Details

To verify the performance of the proposed SV-UDIS algorithm, we train and test our model on the UDIS-D [4] dataset and the vehicle surround perception dataset released by MCOV-SLAM [6]. For the UDIS-D dataset, when performing two-image stitching, we use the original dataset released by Nie et al. [4]; when performing multi-image stitching, we extract a portion of data from it and reorganize it into the input

Open-source library name	Version
openai	0.28.0
ipdb	0.13.9
Pillow	9.2.0
transformers	4.27.2
pytest	7.1.3
opencv-python	4.6.0.66
scipy	1.9.2
face-detection	0.2.2
augly	1.0.0
diffusers	0.14.0
accelerate	0.17.1
ipykernel	6.15.2

TABLE I: Some open-source libraries and their corresponding versions.

```
prompt_text = [
    {
        "role": "system",
        "content": "You are an assistant that \
        generates image manipulation programs \
        based on the user's instruction. \
        Please respond with only the program code, \
        no additional explanations or markdown formatting."
    },
    {
        "role": "user",
        "content": self.prompter(inputs)
    }
]
response = openai.ChatCompletion.create(
    model="gpt-4o",
    messages=prompt_text,
    temperature=self.temperature,
    max_tokens=512,
    top_p=self.top_p,
    frequency_penalty=0,
    presence_penalty=0,
    n=1,
    stop=None,
    logprobs=True
)
```

Fig. 6: Using the API to call GPT-4o in Python.

form of multi-image stitching. For the MCOV-SLAM dataset, we extract a set of surround image data every 50 frames from the original image sequence released by Pan et al [6]., and randomly divide them into training and testing sets at a ratio of 0.85 for the training set. Additionally, when using the UDIS2 [5] method for multi-image stitching, the first step is to stitch two adjacent images together, then stitch them with the next image, and so on.

We train our multi-warp network on these two datasets for 150 epochs, with settings for the optimizer, learning rate, and number of control points consistent with UDIS2 [5]. For the multi-warp stage, γ_1, γ_2, and γ_3 are set to 0.5, 0.2, and 10, respectively. For the multi-composition stage, we use the pretrained model provided by UDIS2 [5]. All implementations are based on PyTorch and the NVIDIA RTX 4090 GPU.

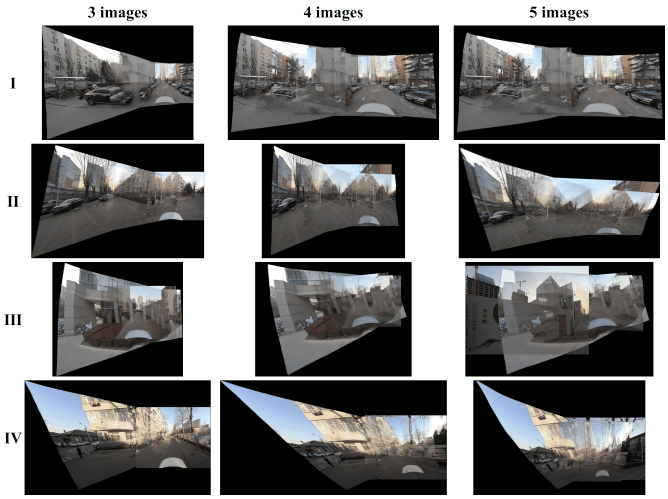

Fig. 7: The results of multi-image stitching by UDIS2 [5] on the MCOV-SLAM [6] dataset.

REFERENCES

[1] J. Zaragoza, T.-J. Chin, Q.-H. Tran, M. S. Brown, and D. Suter, "As-projective-as-possible image stitching with moving dlt," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 36, no. 7, pp. 1285–1298, 2014.

[2] C.-C. Lin, S. U. Pankanti, K. N. Ramamurthy, and A. Y. Aravkin, "Adaptive as-natural-as-possible image stitching," in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015, pp. 1155–1163.

[3] Q. Jia, Z. Li, X. Fan, H. Zhao, S. Teng, X. Ye, and L. J. Latecki, "Leveraging line-point consistence to preserve structures for wide parallax image stitching," in *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021, pp. 12 181–12 190.

[4] L. Nie, C. Lin, K. Liao, S. Liu, and Y. Zhao, "Unsupervised deep image stitching: Reconstructing stitched features to images," *IEEE Transactions on Image Processing*, vol. 30, pp. 6184–6197, 2021.

[5] ——, "Parallax-tolerant unsupervised deep image stitching," in *2023 IEEE/CVF International Conference on Computer Vision (ICCV)*, 2023, pp. 7365–7374.

[6] Y. Yang, M. Pan, D. Tang, T. Wang, Y. Yue, T. Liu, and M. Fu, "Mcov-slam: A multicamera omnidirectional visual slam system," *IEEE/ASME Transactions on Mechatronics*, vol. 29, no. 5, pp. 3556–3567, 2024.

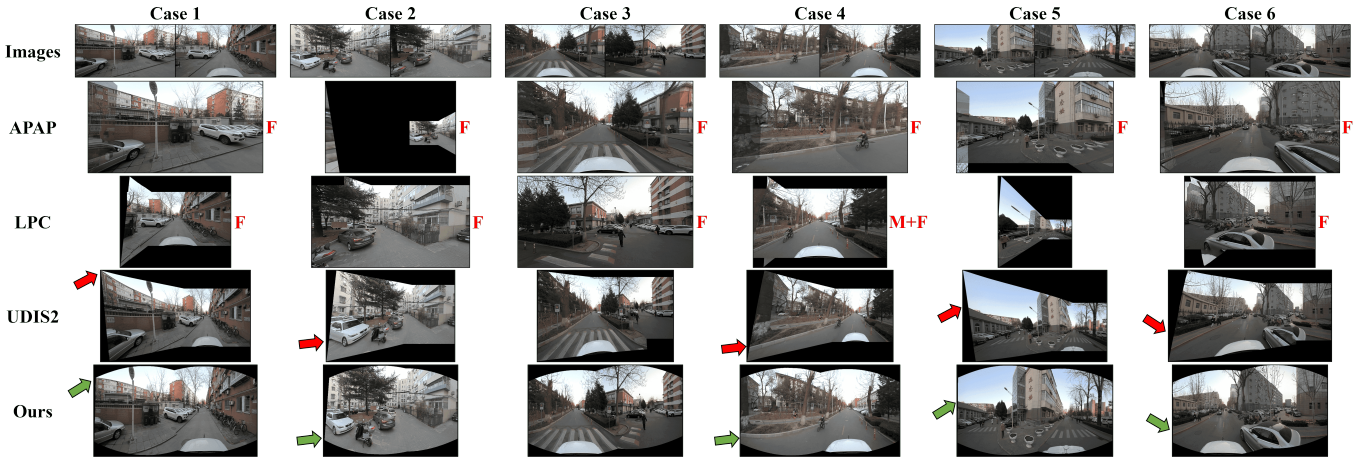

Fig. 8: More qualitative comparison of two-image stitching on the MCOV-SLAM [6] dataset with the APAP [1], LPC [3], and UDIS2 [5] methods. **F** indicates cases of stitching failure. **M** indicates that the LPC [3] algorithm cannot extract effective line feature matches and requires manual annotation of line feature matches.