A fast stitching method for container images using texture and weighted speed

Quanling Meng, Mengqin Zhang, Weigang Zhang
{mengquanling@gmail.com, mqzhang950710@163.com, wgzhang@hit.edu.cn}

School of Computer Science and Technology, Harbin Institute of Technology, Weihai, China

Abstract. With the rapid development of oceans economy, a huge number of shipping containers are transported all around the world. To reduce the risk of container damages during the transportation, existing solution relies mainly on human beings to observe the container appearance before or after it enters a dock, which is time-consuming and inaccurate. To solve this problem, one intelligent approach is to develop an automatic container damage detection framework based on computer vision techniques. But how to obtain the panorama images for container damage detection is a challenging issue. In this paper, a real-time container panorama producing system is developed based on container surveillance videos, which is implemented by container image stitching with texture features. When there is no reliable offset, the weighted speed for splicing is used. Experimental results indicate that the developed system could achieve approving results in a real-time manner.

Keywords: container; computer vision; image stitching; weighted speed

1 Introduction

In recent years, with the development of ocean economy, an increasing number of goods are packed in containers and shipped via freighters to everywhere of the world. The number of containers shipped over the world has been increased every day. For the safety of the goods, it is necessary to ensure the integrity of the container and avoid damage of the container. To reduce the risk of container damage during transportation, existing solution resorts to human beings to observe the container before and after it enters a dock, which is very time-consuming and easy to miss the damage. To solve this problem, one intelligent solution is to develop an automatic container damage detection method based on computer vision techniques.

One of the most challenging issues is to obtaining the panorama of the container, which can be used for subsequent damage detection. Such a technique is called image mosaicking, which outputs the panorama of a specific scene given a set of overlapped images. Mosaicing could be regarded as a special case of scene reconstruction where the images are related by planar homography only[1]. It has been attracted increasing attention in the past few years [2-4]. Although existing methods have achieved great performance in real-time applications, obtaining the panorama of the container is not easy because the surveillance camera is installed very close to the container, which causes the video frame only capture a small part of the whole container. For example, some close-up video frames are shown in Figure 1 (a). In addition, a container has usually few texture. In particular, there may be no texts on the container as shown in Figure 1(b). In such a case, traditional image mosaicing methods cannot obtain a desired performance. There are relatively few existing methods for container splicing. Li et al.[5] conducted research on container image stitching. In the matching stage, the feature matching process is divided into

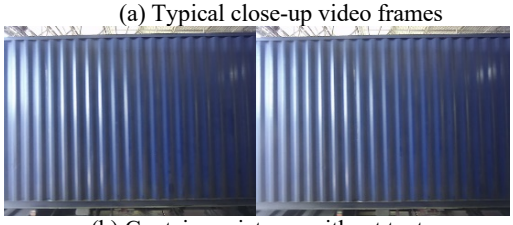
MOBIMEDIA 2019, June 29-30, Weihai, People's Republic of China

DOI 10.4108/eai.29-6-2019.2283071

three rounds to eliminate the image mismatch. Although this method can obtain relatively reliable matching points, it is still difficult to match almost the same two frames of images. Moreover, their equipment was able to photograph most of the container, and only a small amount of images were stitched together during the experiment.

After analyzing the videos of the container captured by the camera installed at the scene, we found that the texture features of the container are discriminative cues for the image mosaicing. Inspired by this observation, we propose an effective mosaicing method based on texture features. First, we pre-process the input images. Then, we seek the starting frame and remove the static frames using template matching. After that, SURF[6] is used to detect keypoints and FLANN[7] is used to match those keypoints. RANSAC[8] is subsequently used to compute transformation between two frames and compute their offset and the offset speed. When there are no good keypoints or the computed offsets are not reliable, we use the weighted speed to stitch two frames.

(a) Typical close-up video frames



(b) Container pictures without text

Fig. 1. Video frame samples of the containers.

The contributions of this paper are two-fold: 1) A complete container mosaicing method is proposed, which achieves desired even if when the container has few obvious features. 2) The proposed mosaicing method is computationally efficient, which can be used in real-time applications.

2 Methods

Based on the uniform sampling of the video frames, we use texture features for image stitching. The entire method is shown in Algorithm 1. We first scan and get the number of images N_i. If the number is too small, no stitching is performed. If the number is too large, the images are selected at equal interval. The first image I is opened and pre-processed while obtaining a small size image I_s for template matching. Then the following steps are performed, including preprocessing, template matching and image stitching. If the number of matching points after the filtering is greater than the threshold T_m, the perspective transformation is performed and the offset is calculated. If the offset is within a reasonable range, we calculate and update the offset speed, and then perform image stitching. If the offset is not in a reasonable range and the hard stitching condition is satisfied, we use the weighted speed for stitching. If

the number of matching points after the screening is not greater than the threshold T_m and the hard splicing condition is satisfied, the splicing is also performed using the weighting speed. At the last step, if there are some remainder images that are not stitched, and the tail stitching conditions are met, the weighting speed is also used for stitching. Finally, we save the images I_o and I_h.

Algorithm 1: Image stitching.

Input: D: Image set.
T_m: Threshold of good match point.
Output: I_o: Small size result image.
I_h: Big size result image.
Get the number of images N_i;
Open and preprocess the first image I;
Get small size image I_s;
for i=1 to N_i **do**
    Open and preprocess image I_i;
    Templates match and update small size image I_s;
    Perform image matching to obtain matching points;
    Get good matching points M_i;
    if Size(M_i) >T_m then
        Perform perspective transformation ;
        Calculate and update the offset R_i;
        if R_i is within reasonable range then
            Calculate and update the offset speed;
            Perform image mosaic;
        else
            if meet hard stitching conditions then
                Calculate the offset R_i;
                Perform image mosaic;
    else
        if meet hard stitching conditions then
            Calculate the offset R_i;
            Perform image mosaic;
if meet the tail stitching conditions then:
    Calculate the offset R_i;
    Perform image mosaic;
Generate image I_o and I_h;

2.1 Image preprocessing

We first preprocess the images to facilitate matching. We splicing from left to right in a unified manner, thus the left image part and the upper image part need to be flipped. At the same time, we crop the image to remove unnecessary image areas or blurred areas, and only use the middle areas of the images for splicing. The size of image I_i is resized to $W_i \times H_i$.

2.2 Template matching

We perform template matching to seek the starting frame and remove the static frames. In order to speed up the matching step, we reduce the image I_i and crop it to get the image I_s for template matching. We only focus on the container region in the image, so we cut the upper and lower parts of the image. When the splicing is not started, if the current image is highly similar to the previous one, skip it directly; otherwise, the current image is considered to be the starting image. In the splicing process, if the current image is very similar to the previous one, the current vehicle is considered to be stationary and the current image is skipped directly. During processing, we will record the number of skips N_{pass}.

2.3 Image matching

We need to calculate the matching points of the current image and the stitched images to calculate the offset R_i. In order to obtain good matching points, we only use the middle part of the image for matching. Firstly, the current image I_i and the intermediate result image I_o are respectively set to the region of interest roi_i and roi_o to obtain the image I_r and the image I_t. The image I_r and image I_t are converted to grayscale images, and then histogram equalization is performed to get good texture features.

We use SURF[6] to calculate the characterization matrix and use FLANN[7] to find the optimal matching points and the sub-optimal matching points. As shown in Figure 2(a), we show the top 20 best matching points and connect them. It is usually unreliable to use the optimal

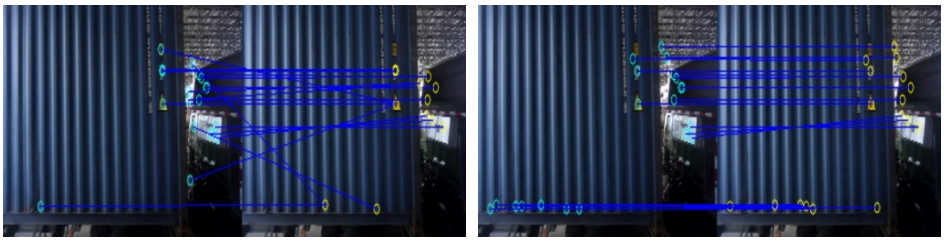
Algorithm 2: Get good matches.

Input: I_r: Current image.
I_t: Intermediate result image.
N_{sc}: Number of speeds.
N_{fc}: Number of images since the last stitching.
T_d:Threshold for finding good matching points.
T_y:Vertical threshold.
T_x:Horizontal threshold.
N_s:Threshold of the number of speeds.
V_{sc}:Speed set.

Output: V_m:Good match point set.
S_a:Average speed.

```
Calculate feature points K1 and K2 using SURF.
Get matching points Vt using FLANN.
if Nsc> Ns then
    Sa ← Median(Vsc)
    Rp ← (Nfc + 1) * Sa
for i=1 to Vt.Size do
    if Vt[i][0].distance < Td * Vt[i][1].distance then
        if abs(Vertical difference) < Ty :
            if Nsc > Ns then
                if abs(Horizontal difference+Rp)< Tx:
                    Vm.pushback(Vt[i][0]);
                continue;
            else
                Vm.pushback(Vt[i][0]);
```

matching points directly, so we need to filter the optimal matching points. We will select the median of the nearest N_s offset speeds as the current offset speed, and predict the current offset R_p for filtering good matching points. The whole process of filtering matching points is shown in Algorithm 2. When we look for good matching points, we will consider the distance between the matching points, the vertical offset size and the horizontal offset size. As shown in Figure 2(b), the matching points become more reliable after filtering.



(a) Images before filter matching points (b) Images after filter matching points

Fig. 2. Image matching.

2.4 Image stitching

Using the matching points calculated in the previous step, we can calculate the offset R_i for image stitching. If the number of good matching points is greater than the threshold T_m, we calculate the offset R_i. Otherwise, we use the weighted speed to calculate the offset R_i.

When the number of matching points after filtering is bigger than the threshold T_m, we perform a perspective transformation to convert the current image and the stitched images to the same coordinate system. First, RANSAC[8] algorithm is used to eliminate false matches as well as to calculate the transformation matrix. Then the two points of the upper left corner and the lower left corner of the image I_r are transformed, and the average value of the transformed abscissa is used as the offset R_i. We use the two thresholds T_{min} and T_{max} to determine whether the offset R_i is within a reasonable range, and use L_p to control the fusion ratio. If the above conditions are met, we calculated and update the speed and then perform image stitching. Otherwise, if the hard splicing condition is satisfied, the product of the offset speed S_{fs} and the number of images N_{fc} is taken as the offset R_i. If the number of matching points is not greater than the threshold T_m, it is also determined whether the hard splicing condition is satisfied, and if so, splicing is performed in the same manner. Where the hard stitching condition refers to the number of images N_{fc} skipped since the last splicing is equal to the threshold T_{mux} or the predicted offset of the next frame will exceed the distance L_p.

The process of calculating and updating the offset speed is as shown in Algorithm 3. We first calculate the current offset speed. If the recorded speed is less than the threshold N_s, the speed is directly added; if not less than N_s, the median of all the statistics is selected as the average speed; if it is greater than N_s, we use the thresholds T_{n1}, T_{n2} and T_a to determine whether the offset has a mutation. If so,the current offset is unreliable, and then the offset speed is calculated proportionally. If the offset does not change much, then record the current speed and delete the first speed.

$$C_o = C_i * ((W_i - s)/L_p) + C_o * ((s - W_i)/L_p) \tag{1}$$

In order to obtain a good stitching effect and remove the seam, we first stitch the left R_i column of the image I_i to the right of the image I_o. Then, the intermediate result image I_o is

weighted and fused in units of columns according to the threshold L_p. As shown in the Formula(1) , s indicates the current number of columns, C_o and C_i represent the current column of image I_o and image I_i, respectively.

Algorithm 3: Calculate and update speed.

Input: R_i: The Offset.
N_{fc}:Number of images.
N_{sc}:Number of speeds.
V_{sc}:Speed set.
S_{fs}:Frame rate.
T_{n1}:Speed change threshold.
T_a: Speed change threshold.
T_{n2}: Speed change threshold
a : Speed weighted ratio.

Output: R_i: The Offset.
V_{sc}:Speed set.
S_{fc}:Frame rate.
S_a:Average speed.

S_{fc} ←int(R_i/N_{fc});
if $N_{sc} < N_s$ then
  V_{sc}.pushback(S_{fc});
if $N_{sc} \geq N_s$ then
  S_a ←MEDIAN (V_{sc})
if $N_{sc} > N_s$ then
  if((S_a-S_{fs})> S_a*T_{n1}) or (abs(S_a-S_{fs})> T_a) or ((S_{fs}-S_a)> S_a*T_{n2}):
    S_{fs} ← V_{sc}.last * a + S_a * (1-a)
  else
    V_{sc}.pushback(S_{fc});
    delete V_{sc}.first

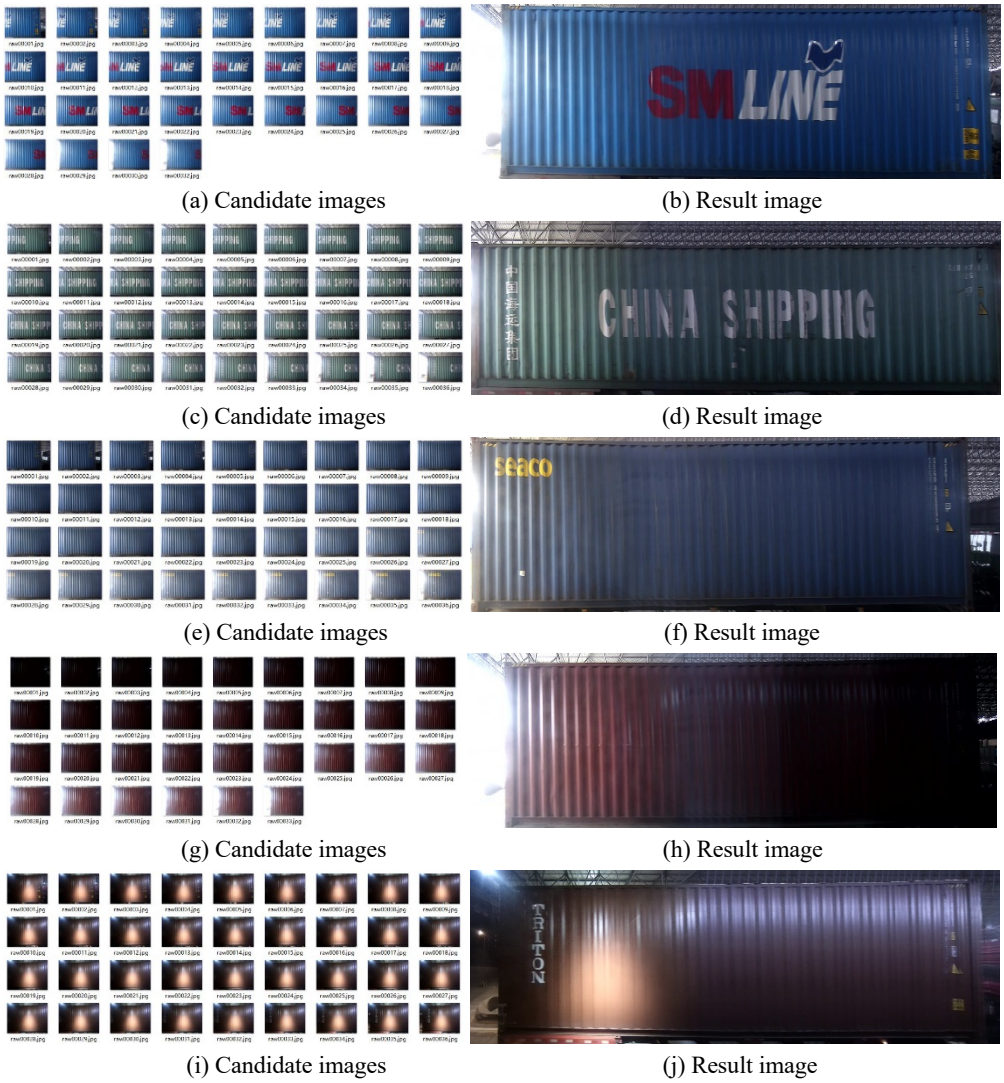
2.5 Image generating

Considering that some images are still not stitched after the stitching is over, in order to achieve a good visual effect, if the tail stitching condition is satisfied, then stitch the current image. After the stitching is finished, we save the result image I_o. In addition, we also recorded the offsets in the stitching process, and then stitched the high quality images based on these offsets. Where the tail stitching condition refers to the number of images skipped N_{fc} since the last stitching is smaller than the threshold T_{fc} and total number of images skipped N_{pass} is smaller than the threshold T_{pass}.

3 Experiments

We experimented with the CPU i7-8700K and 32G memory, using C/C++ to implement the algorithm, and image processing relies on OpenCV[9]. It is important to note that we get the

video frame images at a 100 milliseconds interval. In the process of collecting images, when the container accounts for about 80% of the entire picture, we start or end the collecting. The size of original image is 640×480. During the pre-processing, we did not crop the original image. We set T_m to 30, N_s to 3, T_d to 0.8, T_y to 35, T_x to 50, T_{min} to 0, T_{max} to 90, L_p to 25, T_{mux} to 1, T_{n1} to 0.5, T_{n2} to 0.5, T_a to 20, a to 0.33, T_{fc} to 2, T_{pass} to 6, W_i to 320, H_i to 200. We use 30 to 200 lines of I_i and I_o, and 20 to 320 columns of I_i for image matching.



(a) Candidate images (b) Result image

(c) Candidate images (d) Result image

(e) Candidate images (f) Result image

(g) Candidate images (h) Result image

(i) Candidate images (j) Result image

Fig. 3. Some results of the proposed image stitching method.

Figure 3(a), 3(c) and 3(e) are ideal candidate image set, which are relatively uniform in illumination and have enough texts on the containers. They are very advantageous for image stitching and good results are achieved, as shown in Figure 3(b) ,3(d) and 3(f). Pictures taken during the daytime are usually very clear, but there are also occasions when the lighting conditions are poor. As shown in Figure 3(g) , the illumination is poor, and there is no obvious

text on the container. Even in the case, our method still achieved good result, as shown in Figure 3(h). In the case of the current parameters, from opening the image to generating the resulting image, our method runs for about 2 seconds each round.

As shown in Figure 3(i), the night image has a phenomenon of blurring and light interference, so it is more difficult to splicing images. The middle glare area is caused by a fixed electric light, which makes the image brighter, but because the light is uneven, it causes great difficulty in image matching. And because of the strong light, some texture information lost. We use the same parameters for the experiment, the effect is shown in Figure 3(j), the detail loss is more serious and the accuracy is reduced.

4 Conclusions

This paper presents an effective method for splicing container images using texture features. We flexibly use the template matching method to seek the starting frame and remove the static frames. Our method effectively filters matching points to obtain reliable offsets. Especially when there is no good matching points or the offset is unreliable, we splicing according to the offset calculated by the weighting speed. By recording the offset, our method can restore the entire stitching process to produce high quality images. Through experiments, our method has shown good results and can run at a faster speed. However, our method performs poorly in non-uniform sampling situations and in poorly lit environments. We will improve this method to solve the above-mentioned problem in the future.

Acknowledgments

This work was supported in part by Shandong Provincial Natural Science Foundation, China: ZR2017MF001, and the Scientific Research Innovation Foundation of HIT(Weihai).

References

[1] Ghosh, Debabrata, et al. "Quantitative evaluation of image mosaicing in multiple scene categories." Electro/Information Technology (EIT), 2012 IEEE International Conference on. IEEE, 2012.

[2] Ghannam, Sherin, and A. Lynn Abbott. "Cross correlation versus mutual information for image mosaicing." International Journal of Advanced Computer Science and Applications (IJACSA) 4 (2013).

[3] Okumura, Ken-ichi, et al. "Real-time feature-based video mosaicing at 500 fps." Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on. IEEE, 2013.

[4] Adel, Ebtsam, Mohammed Elmogy, and Hazem Elbakry. "Real time image mosaicing system based on feature extraction techniques." Computer Engineering & Systems (ICCES), 2014 9th International Conference on. IEEE, 2014.

[5] Li, Xueqi et al. "Research on Container Panorama Image Stitching Method." 2018 Chinese Automation Congress (CAC) (2018): 661-664.

[6] Bay, Herbert, Tinne Tuytelaars, and Luc Van Gool. "Surf: Speeded up robust features." European conference on computer vision. Springer, Berlin, Heidelberg, 2006.

[7] Muja, Marius, and David G. Lowe. "Fast approximate nearest neighbors with automatic algorithm configuration." VISAPP (1) 2.331-340 (2009): 2.

[8] Fischler, Martin A., and Robert C. Bolles. "Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography." Communications of the ACM 24.6 (1981): 381-395.

[9] https://opencv.org/