Low Frame Rate Video Target Localization and Tracking Testbed

Yu Pang[a] **Dan Shen**[b] **Genshe Chen**[b] **Pengpeng Liang**[a] **Khanh Pham**[c] **Erik Blasch**[c] **Zhonghai Wang**[b] **Haibin Ling**[a]

[a]Computer & Information Science Department, Temple University, Philadelphia, PA, USA
[b]Intelligent Fusion Technology, Inc, Germantown, MD, USA
[c]Air Force Research Lab, USA
{*yu.pang,pliang,hbling*}*@temple.edu*, {*dshen,zwang,gchen@intfusiontech.com*},
Khanh.Pham@kirtland.af.mil, erik.blasch@rl.af.mil

ABSTRACT

Traditional tracking frameworks are challenged by low video frame rate scenarios, because the appearances and locations of the target may change considerably in consecutive frames. Our paper presents a saliency-based temporal association dependency(STAD) framework to deal with such a low frame rate scenario and demonstrate good results in our robot testbed. We first use median filter to create a background of the scene, then apply background subtraction to every new frame to decide the rough position of the target. With the help of the markers on the robots, we use a gradient voting algorithm to detect the high responses of the directions of the robots. Finally, a template matching with branch pruning is used to obtain the finer estimation of the pose of the robots. To make the tracking-by-detection framework stable, we further introduce the temporal constraints using a previously detected result as well as an association technique. Our experiments show that our method can achieve a very stable tracking result and outperforms some state-of-the-art trackers such as Meanshift, Online-AdaBoosting, Mulitple-Instance-Learning, Tracking-Learning-Detection etc. Also. we demonstrate that our algorithm provides near real-time solutions given the low frame rate requirement.

Keywords: low frame rate, target localization, tracking

1. INTRODUCTION

Target tracking has been well investigated by the computer vision community for decades. An extensive survey of object tracking can be found at.[1] Most of the popular tracking methods are aimed at dealing with targets that have smooth trajectories, for example in Meanshift,[2] Online-AdaBoosting (OAB),[3] Multiple-Instance-Learning (MIL),[4] L1 tracking,[5] Visual-Tracking-Decomposition (VTD)[6] etc. When the image data is obtained under low frame rate conditions, previous studies[7] has shown that traditional trackers will meet serious problems. One method to solve the problem is to use a tracking-by-detection framework, for example Tracking-Learning-Detection (TLD),[8] Structured tracking (Struck),[9] Multiple-Kernel-Learning (MKL)[10] etc. However, many of such methods are prone to appearance changes including the rotations and illumination variations.

In many real applications, even when the scenario seems to be easy to solve (i.e., tracking a high contrast target), it will be quite hard due to many reasons. For example, in Figure 1, Meanshift will fail when different objects are near each other, and it could easily drift away by similar objects as shown in Figure 1(b). Furthermore, the Meanshift cannot distinguish similar markers because of the histogram based tracking. It has no ability to capture the pattern of the markers. We use the public code of TLD tracker from original authors[8] in our experiments. As can be seen from Figure 1(d), when the appearance model changes abruptly, the learning process cannot capture such a change, thus fail to detect the object in the next frame. Also for OAB and MIL, we use the public code from MIL.[4] As shown in Figure 1(f) and (h), they tend to learn an inaccurate model when the background is not clearly separated from the foreground since they use the gray scale images input. Many of the such state-of-the-art video tracking methods are not robust when applied to the real application. Our saliency-based temporal association dependency(STAD) framework will try to tackle these problems under our robot testbed and to achieve a high accuracy solution.

The paper is organized as follows, Section 2 will introduce our testbed followed by Section 3 describing our main techniques used to solve the low-frame rate (LFR) problem. Section 4 shows the performance of our method as well as some comparison results and Section 5 provides conclusions..

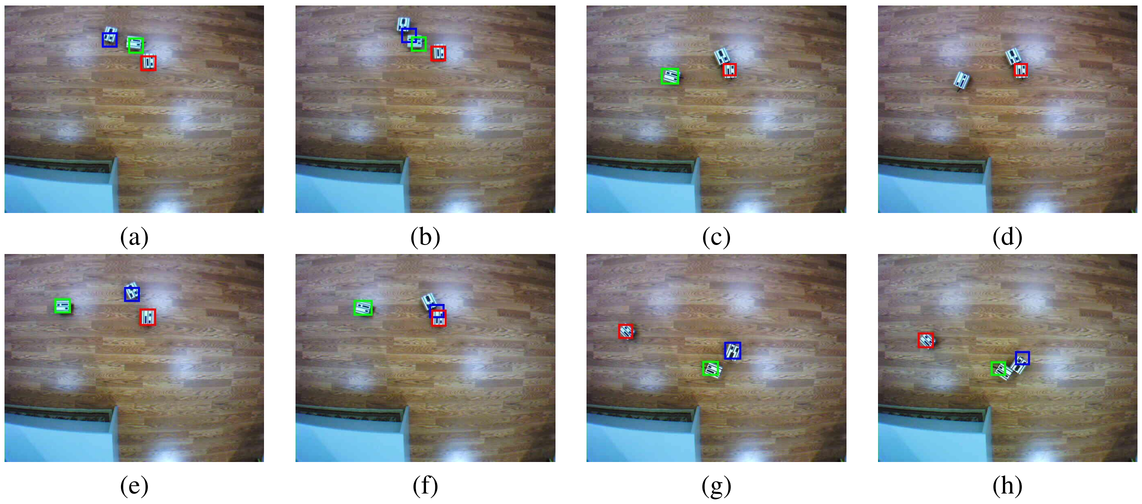


Figure 1. Every pair of images are the consecutive frames from the dataset. The left image is the good result while the right image is when the tracker fails. (a),(b) are obtained from Meanshift, (c),(d) are from TLD, (e),(f) are from OAB and (g),(h) are from MIL.

2. TESTBED SETTINGS

2.1 Environment setting

A typical setting of our environment is a constrained area as shown in Figure 2. The background is static but can be affected by illumination changes. The objects we're tracking are multiple rigid mobile robots. We want to mention that the images are distorted by the wide-angle camera, especially near the camera boundaries. When the robots move in the boundary regions, the tracking accuracy is severely affected which possibly requires alternative methods. The robots can move in and out of the camera view, thus we also need to automatically detect each new instance for track initiation.

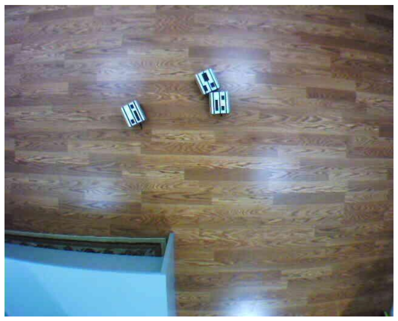


Figure 2. A typical setting of the environment.

2.2 Marker setting

To enable salient robot detection, we experimented with marker designs to identify the robot. In Figure 3, (a) is the original robot with no markers. To enable faster detection, distinguish between robots, and determine robot pose, we compared different marker designs. Figure 3 (b), (c) and (d) are color-based markers. Although we're expecting some color features can be used, the captured images are not stable and the values of the colors can vary a lot due to the illumination changes. As shown in Figure 4, the colors are not saturated and tend to be easily mixed with others. We always experience such color drifting and the spectrum bands overlapping in both red-green-blue (RGB) and hue-saturation-intensity (HSI) spaces. Figure 3 (e) is a pure binary template from which we could apply simple template matching algorithm. But in this case, the search space will be very large and in many situations, the boundary of robots can be mixed with the templates and result in poor results. So finally we come up with the Figure 3 (f) marker, the auxiliary lines are used to detect the gradient to provide the direction, which will be explained later in Section 3.2.

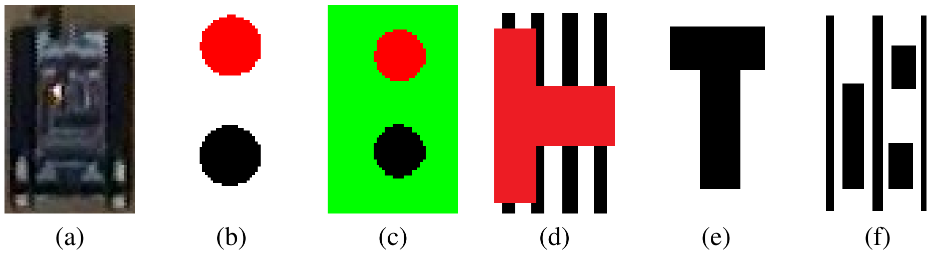


Figure 3. Examples of candidate markers we tried.

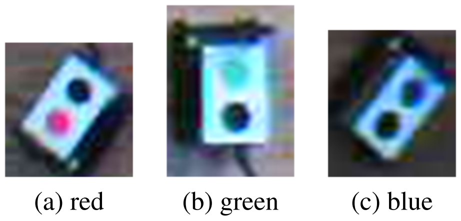


Figure 4. Color markers captured by camera.

3. METHODOLOGY

3.1 Background subtraction and candidate detection

Since we assume a static background, we could take advantage of the background subtraction. We apply a median filter algorithm[11] to get the background model and use it to subtract each image we're processing. For a given set of images $\mathcal{I} = I_1, I_2, ...I_k$, the background image I_{bg} will be computed as:

$$I_{bg}(x, y) = Median(I_1(x, y), I_2(x, y), ...I_k(x, y)) \tag{1}$$

where $I(x, y)$ represents the pixel value at position (x, y) for image I.

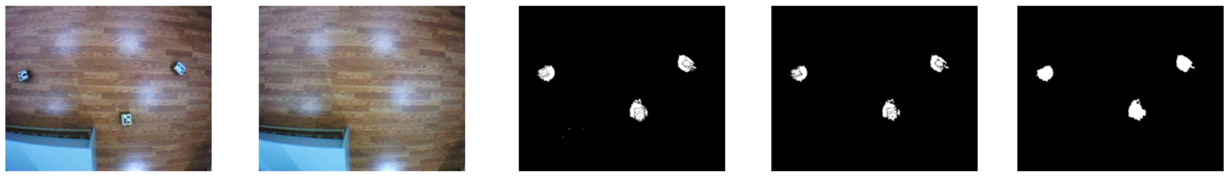
The median filter algorithm works even if when the image sequence contains moving targets. The assumption is that the moving targets in the scene will not stay in one position for more than half of the frames. After we obtained the background model of I_{bg}, we will subtract it from each incoming image I, then a threshold T is set to binarized the subtracted image:

$$I_{sub}(x, y) = \begin{cases} 1; \text{if } |I_{bg}(x, y) - I(x, y)| > T \\ 0; \text{otherwise} \end{cases} \tag{2}$$

Then we apply several standard morphological operations[12] to I_{sub} image. First we apply eroding and dilating to eliminate the foreground noise and keep the size of the objects unchanged. Next, we apply dilating first and then eroding to fill the holes within one object. A sample result is shown in Figure 5. This will guarantee that each region will be fully connected, thus we could perform a connected component algorithm to find each region. It is possible that several robots merge together to form one region, so we need to detect such cases and deal with it accordingly. Empirically, one robot will cover a fixed range of number of pixels, so as soon as we detect some large number of connected pixels, we'll know that there are multiple robots.

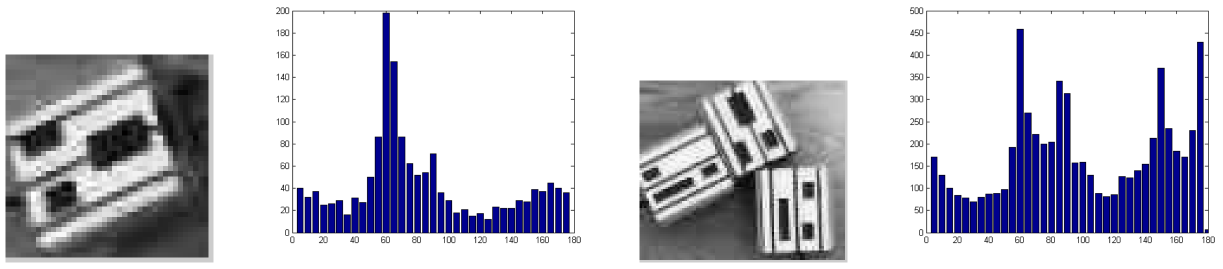
3.2 Direction detection

As mentioned in Section 2.2, our designed marker allows us to quickly determine the robot pose (and hence direction based on the trajectory). Typically, for a template matching algorithm, we need to search along two dimensions of position and one dimension of direction. In our testbed, we typically are trying to detect three robots, thus for a size of 30×30 candidate region and discretized $360/5 = 72$ directions, we will need $3 \times 30 \times 30 \times 72 = 194,400$ searches for only one connected region. This is obviously very insufficient and could take many seconds to process only one frame. With the help of the auxiliary line markers which have a strong edge feature as shown in Figure 3, we could calculate the edge direction and narrow down the search range in direction from 72 to only 2 possibilities.



(a) original image (b) background model (c) subtraction result (d) noise elimination (e) hole filling

Figure 5. An example of background subtraction and foreground objects localization.



(a) (b)

Figure 6. Two examples of gradient detection, the first image (a) contains only one target and the histogram has one peak. The second image (b) contains multiple targets so the histogram has multiple peaks.

For each detected region, we get the gradient of each pixel along its x direction and y direction, then we could get the pixel direction using:

$$\theta = atan(\frac{\partial p(x,y)}{\partial y}, \frac{\partial p(x,y)}{\partial x}) \tag{3}$$

Notice that we only use the range $[0, \pi)$, because the edge gradients of a same line are in two opposite directions but representing the same perpendicular direction of the object.

To calculate the correct direction, we need to consider both candidate directions. After that, we will test both opposite directions to find the correct one. Figure 3.2 shows some examples of the detection results. For a typical one region target as in Figure 3.2(a), we could find one peak in the voting histogram. Ideally, this peak corresponds to the correct direction or the opposite, i.e. correct direction plus π. More interestingly, even if the region contains several candidates, in most cases, this gradient detection approach could detect all the directions. As can be shown from Figure 3.2, the first 3 peaks correspond to the 3 robots directions. But to make it robust, we will always process them one by one. That is, after we detect the first object, we will crop out the detected region and recalculate the gradient histogram again. It will then give us another highest peak corresponding to one of the objects remaining in the region. We will repeat this process until all the objects are assessed for the candidate direction.

3.3 Template matching

After determining the direction, we are ready to apply the template matching algorithm. For each candidate region, we rotate the image according to the detected direction. To further increase detection accuracy in the presence of image noise, we perform a thresholding technique to find the white and black regions of the target pattern and identify the background pixels. The threshold is based on RGB and HSI spaces and the values are set empirically. As shown in Figure 7, the right side is the thresheld image, where white regions of the thresheld image mean the white part of the pattern, gray regions are the black pattern region in original image, and black regions are background. We then compare this thresheld image X with the template images T at each position and 2 directions to find the best match score. The score is computed as:

$$score(X(x,y), T(x,y)) = \begin{cases} 1 & \text{if } T(x,y) = X(x,y) \\ 0 & \text{otherwise} \end{cases} \tag{4}$$

$$Sim(X,T) = \sum_{(x,y)} score(x,y) \times mask(x,y) \tag{5}$$

where $mask(x, y)$ indicates whether the pixel is foreground or background and is obtained for candidate selection (described in Section 3.1) Finally, the parameters with best score will be selected as the detected result.

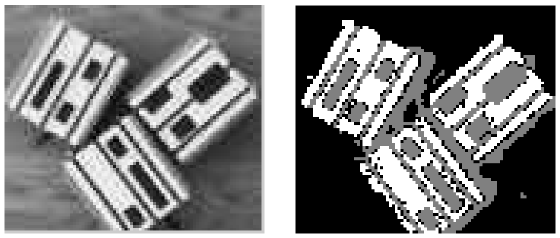


Figure 7. An example of a thresholding result

3.4 Branch pruning

However, the run time is still quite slow even after we implemented our detection methods using MEX in MATLAB, achieving only 0.8 second per frame. As we noticed a fact that if our current best achieved matching score is 500, any region with foreground pixel count that are less than 500 cannot achieve a better score, thus we can reject the region without even comparing it to the template.

To efficiently calculate the foreground pixel count within any given region, we use the idea of an integral image.[13] For any image, we first compute its integral image with respect to the number of foreground pixels. And then for any query of any region, we could get the result with only three add and minus operations. This branch pruning method could further reduce our run time from 0.8 to 0.1 seconds per frame and Wachieve real-time performance in MATLAB.

3.5 Temporal Dependency

If there is no further constraints, the algorithm is solely a tracking-by-detection framework. When there is a false alarm, the detected location will jump to the false alarm which could possibly be far away from the target. As shown in Figure 8, in the upper 2 rows, the green detector detected a false alarm where it should be a red target. Without other constraints, the algorithms will always assign the target to the template which achieves the highest score. In the dataset, especially near the boundary, the target appearance tends to be very noisy. So such error will appear frequently.

This can be corrected by constraining the tracked results using information from the previous frames. More specifically, we obtained the templates positions and directions from the last frame, which we use to constrain our candidate targets with the assumption that the position and direction won't change much in consecutive frames. To achieve this, we use cosine similarity of the difference of the direction and the exponential distance similarity to weigh the final score.

$$\begin{aligned} Score(X,T) =& Sim(X,T) + \lambda_1 cos(|X_{dir} - T_{dir}^{t-1}|) \\ &+ \lambda_2 e^{-\lambda_3 Dist(X_{pos}, T_{pos}^{t-1})} \end{aligned} \tag{6}$$

As shown in Equation 6, X represents the current candidate region, T is an arbitrary template. Besides the similarity score as defined in Equation 5, the second part with λ_1 will contribute to the direction similarity and the third part with λ_2, λ_3 will contribute to the distance similarity. With the absolute operation and exponential constraints, the similarity function will only take values between 0 and 1. So the maximum achievable score will be $Sim(X,T) + \lambda_1 + \lambda_2$. We set the λs empirically with $\lambda_1 = 40$, $\lambda_2 = 80$, $\lambda_3 = 0.02$.

And the result can be seen in the lower two rows in Figure 8. Compared with the upper two rows, we're able to get a much better result because the corrected templates will have higher score than the wrong ones due to the temporal model.

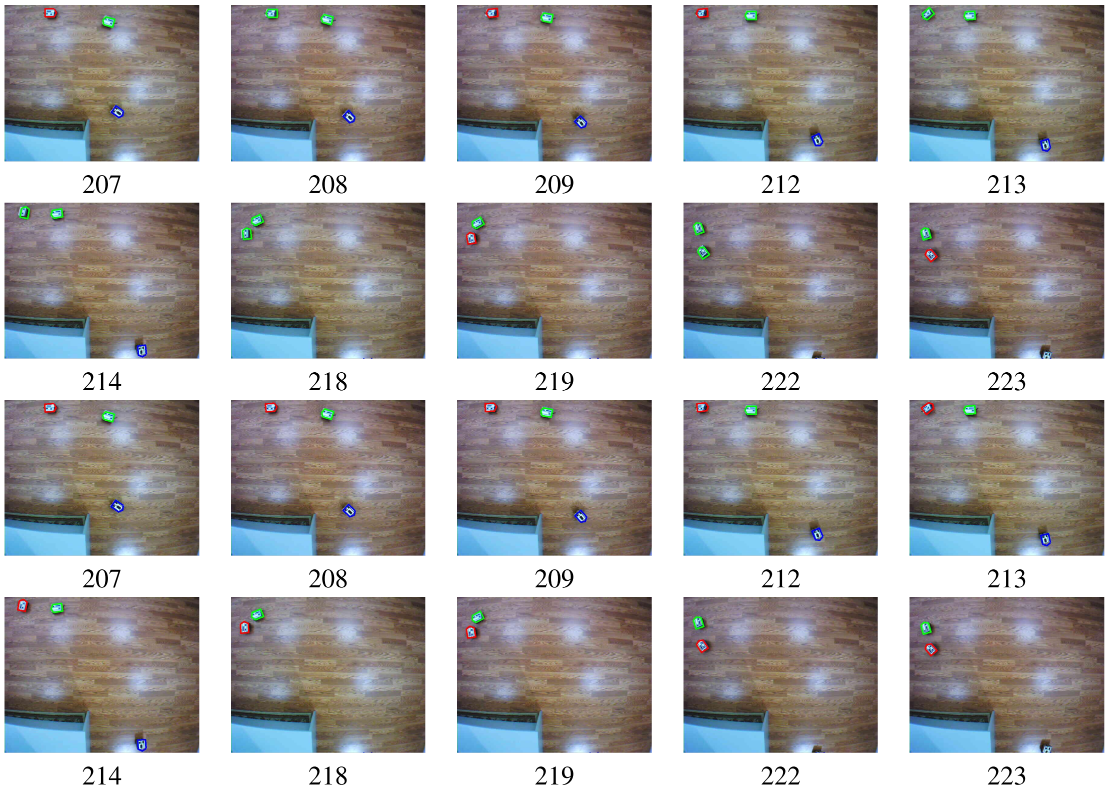


Figure 8. Upper 2 rows are the case where no temporal constraint is introduced, lower 2 rows show when we use previous information as constraints, we can correct some of the errors.

3.6 Confidence Factor

Using only temporal dependency in this way, we are risking the chance that the previous information is incorrect. If the previous information is wrong, we are matching using a wrong similarity score, and thus we have no way to correct it unless there is a very strong differences between correct score and incorrect ones. Especially when a new robot comes into the scene from the boundary for the first time, it's very likely we have a wrong assignment at that point and this will continue for quite some frames before it can be corrected. To alleviate this problem, we introduce the confidence factor and it will be added to Equation 6 and we have:

$$\begin{aligned} Score(X,T,t) =& Sim(X,T) + \lambda_1 cos(|X_{dir} - T_{dir}^{t-1}|) T_{conf}^{t-1} \\ & + \lambda_2 e^{-\lambda_3 Dist(X_{pos}, T_{pos}^{t-1})} T_{conf}^{t-1} \end{aligned} \tag{7}$$

When we have not seen the template target before, we will set the confidence factor to 0. As soon as we determine that a candidate region belongs to this template according to Equation 7, we will update the confidence factor according to Equation 8.

$$\begin{cases} T_{conf}^{0} = 0 \\ T_{conf}^{t} = T_{conf}^{t-1} + \lambda_4 (Sim(X,T_c) - \max\limits_{T_i \neq T_c} Sim(X,T_i)) \end{cases} \tag{8}$$

where T_c is the chosen template. The intuition here is if the chosen template score is far better than others, then our chosen template has a high probability to be the correct one. If the scores of different templates are very similar, then we're not very confident to claim it to be correct. However, if the chosen template score is lower than other templates, we will have a negative result and thus decrease the confidence factor, which means we are less confident about our chosen one to be correct. If we examine the Equation 7 more carefully, we'll see that if the confidence factor is low, the last two parts will not contribute too much to the final score. If the confidence is 0, the final score will be only determined by similarity score.

	Meanshift	OAB	MIL	TLD	our method
Frames	203	140	138	159	8
Targets	569	344	342	234	8

Table 1. The statistics of failed frames and targets

An example is when the target first comes in that we have no prior information about it. When the target is tracked for a few frames, our cumulated confidence factor will increase until we have a full confidence of 1.

3.7 Association

The above processes alone cannot guarantee that the templates will be assigned to only one candidate region. For example, if we detected two regions that both achieved a high score with one template, the above method will assign this template to both regions. This problem falls into the category of association. Given a set of target and templates, we're trying to find an assignment that each target associate with no more than one template and vice versa. In our case, for each region we have the scores against all templates. This will create a bipartition graph between candidate regions and templates. And we want to find the assignment that could maximize the total score. This is a classical assignment problem and can be solved efficiently by the famous Hungarian algorithm. The Hungarian method is widely used in multi-target tracking algorithms and is efficient to implement and guarantee the one-to-one correspondence.

4. EXPERIMENTS

All the experiments are tested on a Intel Core-7 3.40GHz computer with 10GB memory and MATLAB R2011b. As mentioned before, our algorithm could achieve 0.1 second per frame on average for detection of all targets. In most cases, our STAD algorithm can track the position and pose accurately since the directions are detected robustly. And the position error only happened when the robots reach the boundary. For example, as shown in Figure 9, the red target and the green target in the first two images are slightly off the target center. The last image shows when the target comes into the scene from the boundary, the direction of the target is wrong caused by the shadow and distortion of the image.

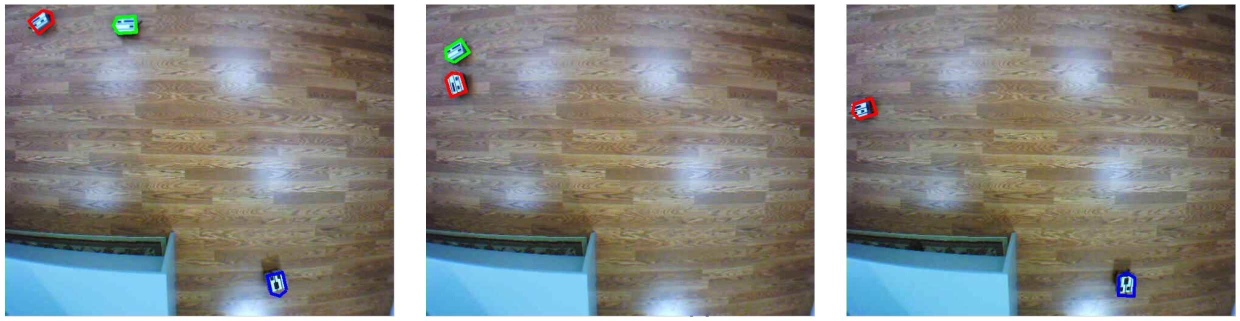


Figure 9. Examples of inaccurate tracking result of our method.

For a quantitative study, we compared the failed frames and failed targets. There are 221 out of 253 frames containing targets, so we will only compare on the 221 frames. The number of ground truth targets in the images is 634. A failed frame means there is at least one detection that is not correct. By correct we mean the center difference between the detected box and the ground truth is no larger than 10 pixels. A failed target is the target not detected by the algorithm, i.e. false negative. We also consider the direction errors in our method to be failed objects. Since none of the other methods could detect the directions of the targets, we won't be able to compare them on this criterion directly.

As shown in Table 1, our method achieves a far better performance than the other four algorithms. In our dataset, the targets can move out of the scene and come back from a different location, resulting in huge problems for Meanshift, OAB and MIL, but not for TLD and our method. Since the Meanshift algorithm is based on histogram comparison, it could easily be off the target when the background pattern is similar to the template pattern in the histograms. Thus very few frames are tracked correctly. OAB and MIL performed quite similarly except for a few targets. Because OAB and MIL use grayscale image inputs, the background pattern can be mixed with the foreground. Also as they search around a small region of the previous detected position, they cannot handle large change in consecutive frames, nor can they detect the target again when the target moves out of the scene and comes back in again. For TLD tracking, we don't have enough positive samples because of the low frame rate, it cannot learn a good model in time before the target change its direction and thus appearance. So TLD tends to get a bad approximation of the template. Our method instead is able to track accurately for almost all the frames. It can both find the correct location and direction. But we also want to emphasize that we are not claiming our method is in general better than these other algorithms but is very robust in our robot testbed compared with others. We did some parameter tuning and applied useful techniques to solve this problem and yield a very good and stable result.

5. CONCLUSION

In this paper, we propose an saliency-based temporal association dependency(STAD) framework that solves the multiple target tracking and pose estimation under our designed testbed. We first use a background subtraction model to obtain the foreground candidate regions. To accelerate the template matching search, we introduce a direction detection method based on the gradients on the pattern. Also we use a branch pruning technique to achieve a better speed. To have a more stable tracking result, we introduce the temporal constraints that use previous detected information to alleviate the final score. A confidence factor is used to reduce the possible error from previous information.

In the experiments, we compared our method with Meanshift, OAB, MIL and TLD to show that our algorithm outperforms their methods in this scenario. We have shown that many state-of-the-art tracking algorithms do not work well under the low frame rate scenario. To make our algorithm work under our specific testbed environment, we have combined several techniques and achieved robust results. Future efforts include incorporating the methods using game-theoretical methods to conduct multi-robot pursuit-evasion strategies comparisons etc.

REFERENCES

[1] Yilmaz, A., Javed, O., and Shah, M., "Object tracking: A survey," *Acm Computing Surveys (CSUR)* **38**(4), 13 (2006).

[2] Comaniciu, D., Ramesh, V., and Meer, P., "Real-time tracking of non-rigid objects using mean shift," in *Computer Vision and Pattern Recognition (CVPR), IEEE Conference on*, **2**, 142–149, IEEE (2000).

[3] Grabner, H., Grabner, M., and Bischof, H., "Real-time tracking via on-line boosting," in *British Machine Vision Conference (BMVC)*, **1**, 47–56 (2006).

[4] Babenko, B., Yang, M.-H., and Belongie, S., "Visual tracking with online multiple instance learning," in *Computer Vision and Pattern Recognition (CVPR), IEEE Conference on*, 983–990, IEEE (2009).

[5] Mei, X., Ling, H., Wu, Y., Blasch, E., and Bai, L., "Minimum error bounded efficient l1 tracker with occlusion detection," in *Computer Vision and Pattern Recognition (CVPR), IEEE Conference on*, 1257–1264, IEEE (2011).

[6] Kwon, J. and Lee, K. M., "Visual tracking decomposition," in *Computer Vision and Pattern Recognition (CVPR), IEEE Conference on*, 1269–1276, IEEE (2010).

[7] Ling, H., Wu, Y., Blasch, E., Chen, G., Lang, H., and Bai, L., "Evaluation of visual tracking in extremely low frame rate wide area motion imagery," in *Information Fusion (FUSION), Proceedings of the 14th International Conference on*, 1–8, IEEE (2011).

[8] Kalal, Z., Matas, J., and Mikolajczyk, K., "P-n learning: Bootstrapping binary classifiers by structural constraints," in *Computer Vision and Pattern Recognition (CVPR), IEEE Conference on*, 49–56, IEEE (2010).

[9] Hare, S., Saffari, A., and Torr, P. H., "Struck: Structured output tracking with kernels," in *Computer Vision (ICCV), IEEE International Conference on*, 263–270, IEEE (2011).

[10] Liang, P., Teodoro, G., Ling, H., Blasch, E., Chen, G., and Bai, L., "Multiple kernel learning for vehicle detection in wide area motion imagery," in *Information Fusion (FUSION), Proceedings of the 15th International Conference on*, 1629–1636, IEEE (2012).

[11] Piccardi, M., "Background subtraction techniques: a review," in *Systems, Man and Cybernetics, IEEE International Conference on*, **4**, 3099–3104, IEEE (2004).

[12] Haralick, R. M., Sternberg, S. R., and Zhuang, X., "Image analysis using mathematical morphology," *Pattern Analysis Machine Intelligence (PAMI), IEEE Transactions on* (4), 532–550 (1987).

[13] Crow, F., "Summed-area tables for texture mapping," *ACM SIGGRAPH Computer Graphics* **18**(3), 207–212 (1984).