Journal of Korea Multimedia Society Vol. 22, No. 1, January 2019(pp. 35-43)
https://doi.org/10.9717/kmms.2019.22.1.035

템플릿 추적 문제를 위한 효율적인 슬라이딩 윈도우 기반 URV Decomposition 알고리즘

이 근 섭†

A Fast and Efficient Sliding Window based URV Decomposition Algorithm for Template Tracking

Geunseop Lee†

ABSTRACT

Template tracking refers to the procedure of finding the most similar image patch corresponding to the given template through an image sequence. In order to obtain more accurate trajectory of the template, the template requires to be updated to reflect various appearance changes as it traverses through an image sequence. To do that, appearance images are used to model appearance variations and these are obtained by the computation of the principal components of the augmented image matrix at every iteration. Unfortunately, it is prohibitively expensive to compute the principal components at every iteration. Thus in this paper, we suggest a new Sliding Window based truncated URV Decomposition (TURVD) algorithm which enables updating their structure by recycling their previous decomposition instead of decomposing the image matrix from the beginning. Specifically, we show an efficient algorithm for updating and downdating the TURVD simultaneously, followed by the rank-one update to the TURVD while tracking the decomposition error accurately and adjusting the truncation level adaptively. Experiments show that the proposed algorithm produces no-meaningful differences but much faster execution speed compared to the typical algorithms in template tracking applications, thereby maintaining a good approximation for the principal components.

Key words: Template Tracking, Principal Components, Truncated URV Decomposition

1. 서 론

템플릿 추적 문제는 연속된 이미지의 각 프레임 내에서 특징 템플릿과 가장 유사한 객체를 찾아내는 과정이다[1]. 하지만 추적 과정에서 이미지 내에 대상 객체가 조명 변화나 미묘한 모양의 변화로 인해 초기 템플릿과 차이가 발생하기 때문에 정확한 객체 모형 설정을 위해 기준 템플릿의 모양도 상황에 따라 변경되어야 한다. Matthew 등은 이를 위해 No-update, Naive update, 그리고 Hybrid update를 제안하였다 [2]. No-update 방식은 추적 중 템플릿을 업데이트 하지 않으며 Naive update는 매 프레임마다 템플릿을 업데이트 하는 방식이다. 하지만 위의 두 방식은 실험을 통해 매 프레임마다 매칭 오차를 누적시켜 대상 객체를 추적하는데 실패하였음을 확인되었다. 한편 하이브리드 업데이트 기법은 템플릿을 매

※ Corresponding Author : Geunseop Lee, Address: (17035) 81, Oedae-ro, Mohyeon-eup, Yongin-si, Gyeonggi-do, Korea, TEL : +82-31-330-4083, FAX : +82-31-330-4074, E-mail : geunseop.lee@hufs.ac.kr
Receipt date : Sep. 4, 2018, Revision date : Dec. 17, 2018
Approval date : Dec. 21, 2018

† Division of Global Business and Technology, Hankuk University of Foreign Studies
※ This work was supported by Hankuk University of Foreign Studies Research Fund and National Research Foundation of Korea (NRF) grant funded by the Korean government (2018R1C1B5085022).

이미지 프레임마다 업데이트 하지만 첫 프레임에서 추출한 템플릿을 반영하여 함께 템플릿을 추적하는 방식이다. 실험결과 하이브리드 업데이트 방식이 더 우수하였으므로 본 논문에서는 하이브리드 템플릿 추적 방식을 사용하였다.

템플릿 추적에서는 각 이미지 프레임 내의 후보 영역과 추적하려는 객체와의 가중치 합을 통해 유사도를 판별한다. 이때 객체 외에도 형태 이미지(appearance image)를 사용하여 정확도를 높일 수 있는데 모양이나 조명의 미세한 변화와 같은 추적 대상의 형태 변화(appearance variation)를 모델링 하는데 도움이 되기 때문이다. 그러므로 가장 최근의 템플릿과 과거의 프레임으로부터 계산된 형태 벡터(appearance vector)를 계산하는 것이 필요하다. 하지만 매 프레임마다 형태 벡터를 계산하기 위해 증강 이미지 행렬(augmented image matrix)의 주성분(principal component)을 계산하여야 하는데, 행렬의 크기가 증가함에 따라 계산 복잡도가 기하급수적으로 증가하는 문제가 있다. 이를 극복하기 위해 주성분 계산을 계산하는 효율적인 방법에 대한 다양한 연구가 진행되었다[3,4,5,6,7,8]. 하지만 기존의 연구는 데이터 행렬이 업데이트됨에 따라 누적 오차의 정확한 추적이 불가능하거나 공분산 행렬의 업데이트가 아닌 일반적인 행렬의 특이값 분해 (singular value decomposition 혹은 SVD)를 업데이트 하는 등의 한계가 있었다. 또한 또한 증강 이미지의 공분산 행렬을 Sliding Window 방식으로 업데이트하는 방식에 관한 연구가 활발하지 않았다. 예를 들면, Badeau등이 제안한 방법 [3]은 Sliding window 방식으로 일반적인 행렬의 SVD를 업데이트 하기 때문에 임의의 공분산 행렬을 처리하는데 적절하지 않으며, 직교 행렬을 업데이트 할 때 절단 (truncation) 으로 인한 누적 오차에 대한 고려가 포함되어 있지 않다.

본 논문에서는 truncated URV decomposition (TURVD)를 이용하여 공분산 행렬이 Sliding Window 방식으로 업데이트 되었을 때, 효율적인 주성분 분석 계산 방식을 제안하고 이를 템플릿 추적 문제에 적용하였다. 특히 누적 오차를 비교적 정확하게 추정하고자 하였고, 이를 기준으로 효율적인 주성분의 절단을 구현하였다. 본 논문의 구성은 다음과 같다. 2장에서는 형태 이미지를 이용한 템플릿 추적 문제를 설명하고 3장에서는 TURVD를 이용하여 빠르고 효율적인 주성분을 계산 하는 알고리즘을 제안한다. 4장에서는 성능 테스트 및 결과를 설명하고, 마지막 5장에서 결론을 맺는다.

2. 형태 이미지를 이용한 템플릿 추적 문제

이번 장에서는 하이브리드 템플릿 추적 방식과 형태 이미지를 이용한 템플릿 추적 문제를 간단히 소개한다. $\boldsymbol{x} = (x_1, x_2)^T$ 를 픽셀 좌표계의 위치 벡터, n을 현재의 이미지 프레임 위치라고 했을 때 이미지 프레임을 $I_n(\boldsymbol{x})$로, 템플릿은 $T(\boldsymbol{x})$로 표시하였다. 또한 템플릿의 변형된 영역을 표현하는 파라미터 벡터를 $\boldsymbol{p} = (p_1, p_2, \ldots)$, 이미지 프레임 내의 후보 영역 템플릿을 $W(\boldsymbol{x};\boldsymbol{p})$로 가정하였다. 일반적으로 템플릿 영역은 회전, 이동, 크기변화 등의 변형이 가능하며 다음과 같이 표현할 수 있다.

$$W(\boldsymbol{x};\boldsymbol{p}) = \begin{pmatrix} (1+p_1)x_1 + & p_2 x_2 & + p_5 \\ p_3 x_1 & + (1+p_4)x_2 & + p_6 \end{pmatrix} \quad (1)$$

형태 이미지를 고려하지 않은 간단한 템플릿 추적 문제의 경우, 현재 프레임에서 주어진 템플릿과 가장 유사한 영역을 찾는 것이라 가정했을 때 다음과 같이 모델링 할 수 있다.

$$\boldsymbol{p_n} = \arg\min_{\boldsymbol{p}} \sum_{\boldsymbol{x} \in T(\boldsymbol{x})} [I_n(W(\boldsymbol{x};\boldsymbol{p})) - T(\boldsymbol{x})]^2 \quad (2)$$

이때 수식 (2)는 비선형 경사하강법 (gradient descent method)으로 계산 할 수 있다 [9]. 형태 이미지를 고려한 하이브리드 방식을 적용한다면 템플릿 추적 경로를 다음과 같은 순서로 계산한다[2]. 이때 $gd\min_{\boldsymbol{p}=\boldsymbol{p_s}} f(\boldsymbol{p})$는 함수 $f(\boldsymbol{p})$ 에서 $\boldsymbol{p_s}$를 초기 값으로 하는 경사하강법으로 정의한다.

1) 초기 템플릿 $T_1(\boldsymbol{x})$ 과 변형 파라미터 $\boldsymbol{p_1}$이 주어졌다고 가정한다.

2) 경사하강법을 이용하여 $\boldsymbol{p_1}$에서 시작하는 최적해 $\boldsymbol{p_1}^*$를 다음과 같이 계산한다.

$$\boldsymbol{p_1}^* = gd\min_{\boldsymbol{p}=\boldsymbol{p_1}} \sum_{\boldsymbol{x} \in T_1(\boldsymbol{x})} [I_1(W(\boldsymbol{x};\boldsymbol{p})) - T_1(\boldsymbol{x})]^2 \quad (3)$$

3) 현재 템플릿 $T_n(\boldsymbol{x})$과 형태 벡터 $\boldsymbol{u_n^i}(\boldsymbol{x})$ 이 있을 때 프레임 2부터 N까지 다음을 계산한다.

$$(\boldsymbol{p_n}, \Lambda_n) = gd\min_{\boldsymbol{p}=\boldsymbol{p_{n-1}}} \sum_{\boldsymbol{x} \in T_n(\boldsymbol{x})} [I_n(W(\boldsymbol{x};\boldsymbol{p})) - T_n(\boldsymbol{x}) - \sum_{i=1}^{d_n} \lambda_i \boldsymbol{u_n^i}(\boldsymbol{x})]^2$$

$$(\boldsymbol{p_n}^*, \Lambda_n^*) = gd\min_{\boldsymbol{p}=\boldsymbol{p_n}} \sum_{\boldsymbol{x} \in T_n(\boldsymbol{x})} [I_n(W(\boldsymbol{x};\boldsymbol{p})) - T_1(\boldsymbol{x}) - \sum_{i=1}^{d_1} \lambda_i \boldsymbol{u_n^i}(\boldsymbol{x})]^2 \quad (4)$$

4) 주어진 임계값 ϵ이 있을 때 매 프레임마다 조건 $\| \boldsymbol{p_n} - \boldsymbol{p_n}^* \|_2 \leq \epsilon$ 을 조사한다. 만약 조건을 만족하면 새로운 템플릿 $T_{n+1}(\boldsymbol{x})$는 증강 이미지 행렬 $(I_n(W(\boldsymbol{x};\boldsymbol{p_1}^*),...,I_n(W(\boldsymbol{x};\boldsymbol{p_n}^*))$의 평균, 형태 벡터 $\boldsymbol{u}^i_{n+1}$는 이미지 행렬의 주성분으로 각각 업데이트 하고, 조건을 만족하지 않으면 업데이트 하지 않는다. 여기서 Λ_n은 형태 벡터의 계수 $(\lambda_1,\lambda_2,...,\lambda_{d_n})$이다. Fig. 1에서 Hybrid update와 Sliding Window 기반의 TURVD 업데이트와 rank-1 행렬 업데이트 과정을 간략히 표현하였다. Fig. 1에 나타난 이미지 행렬의 주성분 계산 과정은 3장에서 자세히 설명한다.

3. Sliding Window 기반 TURVD 알고리즘

수식 (4)에서 템플릿이 업데이트될 때마다 계산되어야 할 변수들은 $T_n(\boldsymbol{x})$ 와 $\boldsymbol{u_n}(\boldsymbol{x})$ 이다. 벡터 $\boldsymbol{e} \in R^n$을 모든 값이 1로 구성된 벡터라고 했을 때 $T_n(\boldsymbol{x}) \in R^m$은 이미지 행렬 $\overline{X_n}(\boldsymbol{x}) = [I_1(W(\boldsymbol{x};\boldsymbol{p_1}^*)), I_2(W(\boldsymbol{x};\boldsymbol{p_2}^*)),...,I_n(W(\boldsymbol{x};\boldsymbol{p_n}^*))]$, $\overline{X_n}(\boldsymbol{x}) \in R^{m\times n}$의 평균값이며,

$$T_n(\boldsymbol{x}) = \frac{1}{n}[I_1(W(\boldsymbol{x};\boldsymbol{p_1}^*)), I_2(W(\boldsymbol{x};\boldsymbol{p_2}^*)),...,I_n(W(\boldsymbol{x};\boldsymbol{p_n}^*))]\boldsymbol{e} \tag{5}$$

와 같이 계산한다. 행렬 $X_n(\boldsymbol{x})$ 를

$$X_n(\boldsymbol{x}) = [I_1(W(\boldsymbol{x};\boldsymbol{p_1}^*)), I_2(W(\boldsymbol{x};\boldsymbol{p_2}^*)),...,I_n(W(\boldsymbol{x};\boldsymbol{p_n}^*))] - T_n(\boldsymbol{x})\boldsymbol{e}^T \tag{6}$$

으로 가정하고, 벡터 $\boldsymbol{u}_n(\boldsymbol{x})$ 를 행렬 $X_n(\boldsymbol{x})$ 의 특이값 벡터(singular vector)이라 했을 때, $X_n(\boldsymbol{x})X_n(\boldsymbol{x})^T$은 $\overline{X_n}(\boldsymbol{x})$의 공분산 행렬이기 때문에 $\boldsymbol{u}_n(\boldsymbol{x})$ 와 공분산 행렬의 주성분, 혹은 형태벡터는 동일한 부분영역(subspace)을 나타낸다. 하지만 일반적인 특이값 분해 알고리즘을 적용하면 대략 $O(n^3)$의 복잡도가 필요하기 때문에 실시간 템플릿 추적에는 부적절하다[10].

새로운 프레임이 추가되면 행렬 $X_n(\boldsymbol{x})$ 은 $X_{n+1}(\boldsymbol{x})$로 업데이트 되는데, 수식 (7)과 같이 행렬의 크기를 일정하게 유지하기 위해 Sliding Window 방식을 사용한다.

$$X_{n+1}(\boldsymbol{x}) = [I_2(W(x;p_2^*)), I_3(W(x;p_3^*)),..., I_{n+1}(W(x;p_{n+1}^*))] - T_{n+1}(x)e^T \tag{7}$$

이때 새로운 평균 $T_{n+1}(\boldsymbol{x})$ 는 다음과 같다.

$$T_{n+1}(\boldsymbol{x}) = \frac{1}{n}[I_2(W(\boldsymbol{x};\boldsymbol{p_2}^*)),...,I_{n+1}(W(\boldsymbol{x};\boldsymbol{p_{n+1}}^*))]\boldsymbol{e} \tag{8}$$

Sliding Window 방식을 사용할 때, 기존 계산 결과를 이용하여 효율적으로 계산할 수 있도록 새로운 공분산 행렬식은 다음과 같이 표현할 수 있다.

$$\begin{aligned}&(X_n(\boldsymbol{x}) \quad I_{n+1}(W(\boldsymbol{x};\boldsymbol{p_{n+1}}^*)) - T_{n+1}(\boldsymbol{x}))\\ &= (I_1(W(\boldsymbol{x};\boldsymbol{p_1}^*)) - T_n(\boldsymbol{x}) \;\; X_{n+1}(\boldsymbol{x})) + (T_n(\boldsymbol{x}) - T_{n+1}(\boldsymbol{x}))\boldsymbol{e}^T.\end{aligned} \tag{9}$$

여기서 수식 (9)는 두 가지 절차로 나눌 수 있는데, 기존의 이미지 행렬 $X_n(\boldsymbol{x})$ 에서 처음 프레임 $I_1(W(\boldsymbol{x};\boldsymbol{p_1}^*)) - T_n(\boldsymbol{x})$을 제거하고 새로운 프레임 $I_{n+1}(W(\boldsymbol{x};\boldsymbol{p_{n+1}}^*)) - T_{n+1}(\boldsymbol{x})$을 추가하는 Sliding Window 과정과 rank-1 행렬 $(T_n(\boldsymbol{x}) - T_{n+1}(\boldsymbol{x}))\boldsymbol{e}^T$를 업데이트 하는 과정이다.

3.1 Sliding Window 기반의 TURVD 알고리즘

TURVD는 주어진 행렬을 두 개의 직교 행렬과 하나의 상삼각행렬(upper triangular matrix)로 분해하는 방식이다. 특히 TURVD은 행렬의 일부가 업데이트 되었을 때 SVD와는 다르게 처음부터 계산할 필요 없이 기존의 TURVD를 재활용하여 훨씬 쉽게 새로운 행렬의 TURVD를 계산할 수 있는 장점이 있

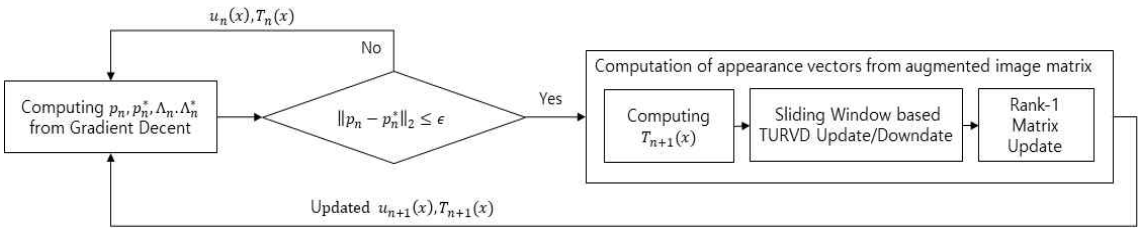


Fig. 1. Brief procedure of Template Tracking based on Hybrid update strategy and proposed algorithm.

다 [11,12]. 또한 이 과정에서 행렬의 계수(rank)를 유추할 수도 있다. 특히 본 논문에서는 기존에 연구되지 않았던 Sliding Window 방식의 TURVD를 업데이트하는 새로운 알고리즘을 제안하였으며, 오차 행렬의 정확한 값을 유추하기 위한 제약 조건(constraint)들을 추가하여 보다 정확한 절단이 가능하도록 하였다.

임의의 행렬 $X_n \in R^{m\times n}$의 TURVD를 다음과 같이 가정한다.

$$X_n = U_n R_n V_n^T + E_n \tag{10}$$

여기서 $U_n \in R^{m\times k}$와 $V_n \in R^{n\times k}$은 직교 행렬이고 $R_n \in R^{k\times k}$은 상삼각행렬이다. 또한 k는 X_n의 계수값을 나타내며 E_n은 low-rank 추정으로 인해 발생하는 오차 행렬이다. 이때 보다 정확한 템플릿 추적을 위해 E_n은 주어진 임계값 ϵ과 함께 다음과 같은 제약 조건들을 만족해야 한다.

$$\| E_n \|_F \le \epsilon, \quad E_n V_n = 0 \tag{11}$$

수식 (9)의 전반부에서 표현된 것 같이, X_n에 새로운 데이터 $\boldsymbol{x_{n+1}}$이 추가되고 기존의 X_n에서 $\boldsymbol{x_1}$이 삭제되어야 한다면 새로운 행렬 X_{n+1}은 다음과 같다.

$$(\boldsymbol{x_1} \quad X_{n+1}) = (X_n \quad \boldsymbol{x_{n+1}}) \tag{12}$$

X_{n+1}의 TURVD를 구하기 위해 먼저 다음의 조건을 만족하는 벡터 $\boldsymbol{l}, \boldsymbol{d}, \boldsymbol{h}$과 $\boldsymbol{v_1}, \boldsymbol{u_{k+1}}, \boldsymbol{u_1}$를 각각 계산한다.

$$\begin{aligned} \boldsymbol{e_1} &= (V_n \quad \boldsymbol{v_1})\boldsymbol{l}, \\ X_n \boldsymbol{v_1} &= (U_n \quad \boldsymbol{u_{k+1}})\boldsymbol{d}, \\ \boldsymbol{x_{n+1}} &= (U_n \quad \boldsymbol{u_{k+1}} \quad \boldsymbol{u_1})\boldsymbol{h} \end{aligned} \tag{13}$$

수식 (13)에서 $\boldsymbol{v_1}, \boldsymbol{u_{k+1}}, \boldsymbol{u_1}$를 계산하기 위해서는 고전적인 그람-슈미트(Gram- schmidt)를 사용할 수 있다 [11,13]. $\boldsymbol{d}^T = (\boldsymbol{d_1^T} \quad d_2)^T$라 하고 $\boldsymbol{h}^T = (\boldsymbol{h_1^T} \quad h_2 \quad h_3)^T$라 하면 수식 (10)과 (12)는

$$\begin{aligned} (\boldsymbol{x_1} \quad X_{n+1}) &= (X_n \quad \boldsymbol{x_{n+1}}) \\ &= (U_n \quad \boldsymbol{u_{k+1}} \quad \boldsymbol{u_1}) \begin{pmatrix} R_n & \boldsymbol{d_1} & \boldsymbol{h_1} \\ 0 & d_2 & h_2 \\ 0 & 0 & h_3 \end{pmatrix} \begin{pmatrix} V_n^T & 0 \\ \boldsymbol{v_1^T} & 0 \\ 0 & 1 \end{pmatrix} + \tilde{E}_{n+1} \end{aligned} \tag{14}$$

와 같이 표현할 수 있다. 여기서 $\tilde{E}_n$은 새로운 오차 행렬이며 $\tilde{E}_n = E_n - X_n \boldsymbol{v_1} \boldsymbol{v_1}^T$이다. 다음으로 수식 (15)와같이 직교 행렬 P_1과 Q_1를 계산한다.

$$\begin{pmatrix} V_n & \boldsymbol{v_1} & 0 \\ 0 & 0 & 1 \end{pmatrix} Q_1 = \begin{pmatrix} 0 & 1 \\ V_{n+1} & 0 \end{pmatrix}, \; P_1^T \begin{pmatrix} R_n & \boldsymbol{d_1} & \boldsymbol{h_1} \\ 0 & d_2 & h_2 \\ 0 & 0 & h_3 \end{pmatrix} Q_1 = \begin{bmatrix} R_{n+1} & \boldsymbol{r_1} \\ 0 & r_2 \end{bmatrix} \tag{15}$$

이때 P_1과 Q_1은 k-1번의 연속된 기븐스 회전(Givens rotation)으로 계산할 수 있다. 수식 (15)의 R_{n+1}은 업데이트된 상삼각행렬이다. 그러므로 업데이트된 U_{n+1}는

$$(U_{n+1} \quad \widetilde{\boldsymbol{u_1}}) = (U_n \quad \boldsymbol{u_{k+1}} \quad \boldsymbol{u_1}) P_1 \tag{16}$$

와 같이 계산하며, 행렬 X_{n+1}의 새로운 TURVD는 다음과 같다.

$$X_{n+1} = U_{n+1} R_{n+1} V_{n+1}^T + \tilde{E}_{n+1}(2:m,:). \tag{17}$$

오차 행렬 $\tilde{E}_n$의 모든 값을 다시 계산 하는 것은 계산 속도의 측면이나 메모리 활용 측면에서 굉장히 비효율적이며, 조건 (11)을 확인하기 위해서 $\| \tilde{E}_n \|_F$의 값으로 충분하다. 그러므로 업데이트된 오차 행렬의 값을 다음과 같이 계산한다.

$$\| E_{n+1} \|_F = \| \tilde{E}_{n+1}(2:m,:) \|_F = \sqrt{\| \tilde{E}_{n+1} \|_F^2 - \| X_n \boldsymbol{v_1} \|_2^2}. \tag{18}$$

마지막으로 TURVD를 업데이트하기 위한 조건 (11)을 만족하는지 확인하여야 한다. 새로운 데이터가 추가되었으므로 X_{n+1}의 계수 값은 $k+1$이 되었는데 새로 업데이트된 이미지 행렬의 가장 작은 특이값이 조건 (11)을 만족하는 범위에서 충분히 작다면 이를 절단할 수 있다. 이를 통해 TURVD의 크기를 작게 관리할 수 있어 연산 속도나 메모리 관리에 더욱 효율적이다. 조건 (11)을 조사하기 위해, 먼저 R_{n+1}의 가장 작은 특이값과 특이 벡터쌍 (smallest singular triplet)를 계산한다. 이를 $(\sigma_{k+1}, \boldsymbol{u_{k+1}}, \boldsymbol{v_{k+1}})$이라고 가정했을 때, 온전한 SVD 보다는 적당한 수의 Power iteration이나 Lanczos iteration을 적용하여 간단히 계산한다. 만약 $\sqrt{\sigma_{k+1}^2 + \| E_{n+1} \|_F^2} \le \epsilon$ 이라면 해당 TURVD의 가장 작은 특이값은 무시할 정도로 작은 값이기 때문에 절단한다. 이를 위해 공식 (15)와 유사하게 k+1의 기븐슨 회전을 적용하여 다음을 만족하는 직교 행렬 P_2과 Q_2를 계산한다 [6].

$$Q_2^T \boldsymbol{v_{k+1}} = \pm \boldsymbol{e_{k+1}}, \; P_2^T R_{n+1} Q_2 = \begin{pmatrix} R_{n+1} & 0 \\ 0 & \sigma_{k+1} \end{pmatrix} \tag{19}$$

그 후 직교 행렬 P_2과 Q_2을 각각 U_{k+1}과 V_{k+1}에 업데이트 한다. 또한 최종적으로 X_{n+1}의 계수는 k가 된다. 위의 과정에서 가장 복잡한 계산은 행렬 U_{k+1}과 V_{k+1}를 P_1과 Q_1으로 업데이트하는 과정인데 보통 $O(nk^2)$ 연산이 필요하기 때문에 일반적인 주성분 분석을 계산하기 위한 복잡도인 $O(n^3)$보다 훨씬 빠르다.

3.2 Rank-1 행렬 업데이트

수식 (9)의 후반부에서 항목 $(T_n(\boldsymbol{x})-T_{n+1}(\boldsymbol{x}))\boldsymbol{e}^{\boldsymbol{T}}$를 간단히 $\boldsymbol{st}^{\boldsymbol{T}}$라 하면 TURVD의 rank-1 업데이트 문제는 다음과 같이 모델링 될 수 있다 [6].

$$X_{n+1}=X_n+\boldsymbol{st}^{\boldsymbol{T}}=U_nR_nV_n^T+E_n+\boldsymbol{st}^{\boldsymbol{T}}. \tag{20}$$

먼저 수식 (13)과 유사하게 3번의 그람-슈미트를 적용하여 다음을 계산한다.

$$\begin{aligned}\boldsymbol{s} &= (U_n \quad \boldsymbol{u_{k+1}})\hat{\boldsymbol{s}},\\ \boldsymbol{t} &= (V_n \quad \boldsymbol{v_{k+1}})\hat{\boldsymbol{t}},\\ X_n\boldsymbol{v_{k+1}} &= (U_n \quad \boldsymbol{u_{k+1}} \quad \boldsymbol{u_{k+2}})\boldsymbol{g}.\end{aligned} \tag{21}$$

벡터 $\boldsymbol{g}^{\boldsymbol{T}}=(\boldsymbol{g_1^T} \quad g_2 \quad g_3)$ 라 하면 X_{n+1}의 TURVD는 다음과 같이 재구성 될 수 있다.

$$X_{n+1}=(U_n \quad \boldsymbol{u_{k+1}} \quad \boldsymbol{u_{k+2}})\left[\begin{pmatrix}R_n\,\boldsymbol{g_1}\\ 0\;\; g_2\\ 0\;\; g_3\end{pmatrix}+\begin{pmatrix}\hat{\boldsymbol{s}}\\ 0\end{pmatrix}\hat{t}^T\right]\begin{pmatrix}V_n^T\\ \boldsymbol{v_{k+1}^T}\end{pmatrix}+E_{n+1} \tag{22}$$

다음 단계에서는 식(22)의 중간 행렬을 QR 변환을 이용하여 상삼각행렬로 변환하며 이때 필요한 직교 행렬을 아래와 같이 P_3라고 가정한다.

$$P_3^T\left[\begin{pmatrix}R_n\,\boldsymbol{g_1}\\ 0\;\; g_2\\ 0\;\; g_3\end{pmatrix}+\begin{pmatrix}\hat{\boldsymbol{s}}\\ 0\end{pmatrix}\hat{t}^T\right]=\begin{pmatrix}R_{n+1}\\ 0\end{pmatrix} \tag{23}$$

또한 오차 행렬의 값은 $\|E_{n+1}\|_F=\sqrt{\|E_n\|_F^2-\|X_n\boldsymbol{v_{k+1}}\|_2^2}$와 같이 업데이트 한다. 3.1장에서와 마찬가지로 rank-1 업데이트는 조건 (11)을 만족해야 하는데 이를 위해서 업데이트 된 상삼각행렬 R_{n+1}의 가장 작은 특이값을 조사하여 불필요한 성분을 제거할 수 있는지 여부를 조사한다. 일반적으로 R_{n+1}의 가능한 계수는 $k+1, k, k-1$이므로 가장 작은 특이값을 최대 2번 조사하여 해당 성분을 절단한다. Rank-1 행렬 업데이트 과정에서 가장 연산 시간이 많이 걸리는 단계는 Sliding Window 방식과 마찬가지로 직교 행렬 업데이트나 오차 행렬의 업데이트에 필요한

$\|X_n\boldsymbol{v_{k+1}}\|_2$ 계산이다. 이는 대략 $O(n^2)$ 계산 복잡도를 가지고 있어 일반적인 주성분 분석 알고리즘에 비해 훨씬 효율적이다.

4. 실험 결과 및 고찰

이번 장에서는 Sliding Window 기반의 TURVD 알고리즘을 실제 템플릿 추적 문제에 적용하고 주성분을 계산하는 기존의 알고리즘들과 성능을 비교하였다. 실험은 Intel Core-i7 프로세서와 RAM 8GB 컴퓨터에서 진행되었으며 소프트웨어는 MATLAB 버전 9.3.0.713599에서 제작하였다.

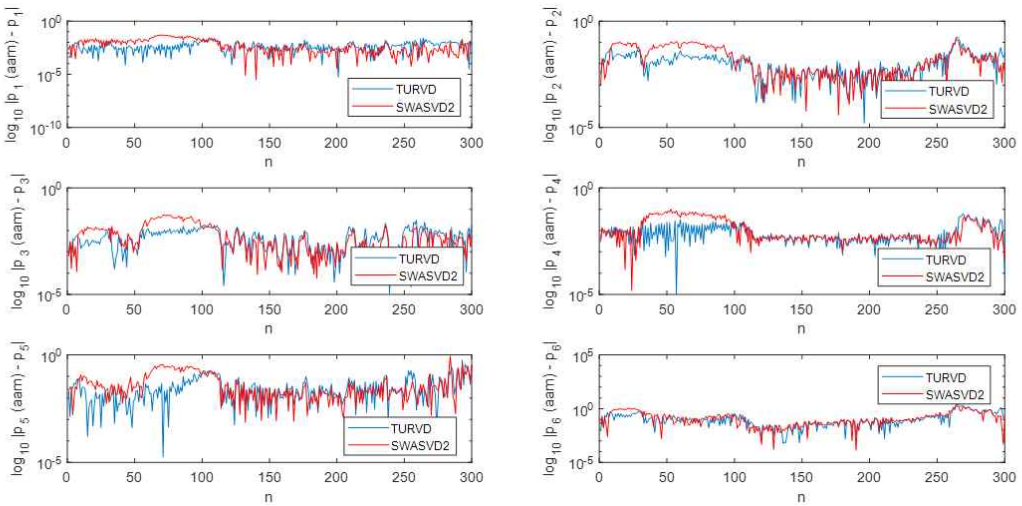
실험을 위한 테스트 영상은 PETS2001의 5번째 데이터 셋과 PETS2006의 두 번째 데이터 셋 중에서

(a) PETS2001 with dataset 5

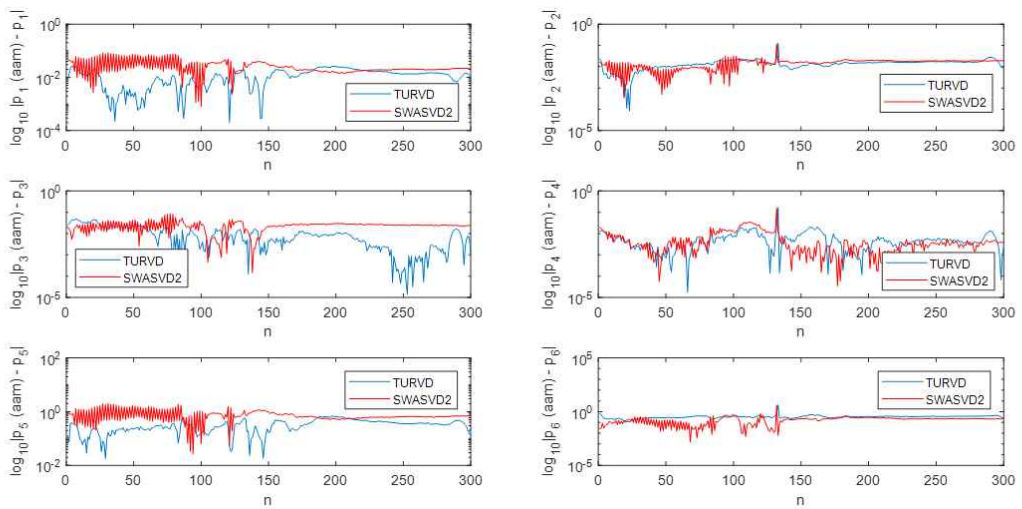
(b) PETS2006 with dataset 2

Fig. 2. The example image sequence for every 100 frames.

각각 500 frame을 선정하였다. 특히 본 논문의 목적이 기존의 주성분 분석과 유사한 정확도를 가진 효율적인 알고리즘을 제안하는데 있기 때문에 복잡한 예제 보다는 주성분만을 이용하여 간단히 추적할 수 있는 예제를 선정하였다. 초기 템플릿은 임의로 지정하였고 크기는 각각 60×60, 72×72이다. Fig 2는 매 100 프레임마다 템플릿을 추적한 결과를 나타내었다. 여기서 빨간 사각형은 각 프레임에서 추적한 템플릿의 위치를 나타낸다. 성능 비교를 위해 매 프레임마다 템플릿 이미지 행렬에 MATALB 'princomp' 함수, Badeau 등이 제안한 알고리즘 [3] (이후 SWASVD2)과, 제안한 알고리즘(TURVD)을 사용하여 형태 벡터를 각각 계산하였다. 윈도우 사이즈는 200이며 최초 윈도우 사이즈에 도달하기 까지는 Sliding Window 방식으로 TURVD를 업데이트 하지 않고 TURVD에 데이터를 추가하는 형태로 업데이트를 진행하였다. TURVD에 데이터를 추가하는 알고리즘은 [6]을 참조하였다. SWASVD2의 경우는 윈도우 사이즈에 도달하기 까지는 Matlab 'princomp' 함수를 사용하였고 이후부터는 SWASVD2를 사용하였다. 다만



(a) PETS2001 with dataset 5

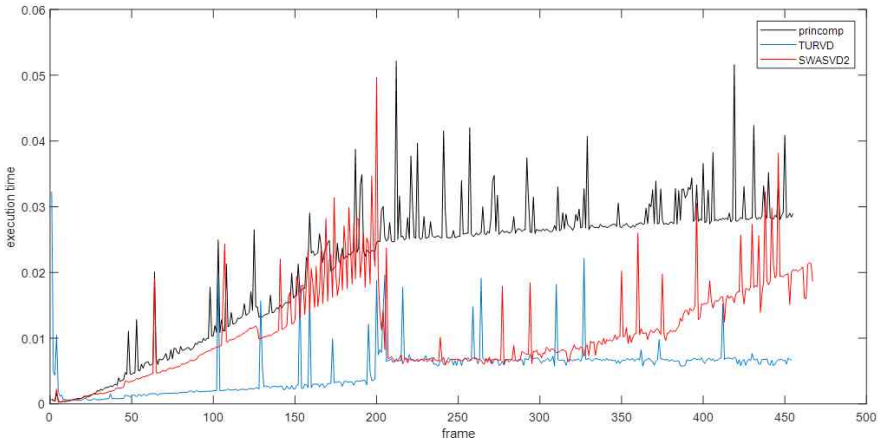


(b) PETS2006 with dataset 2

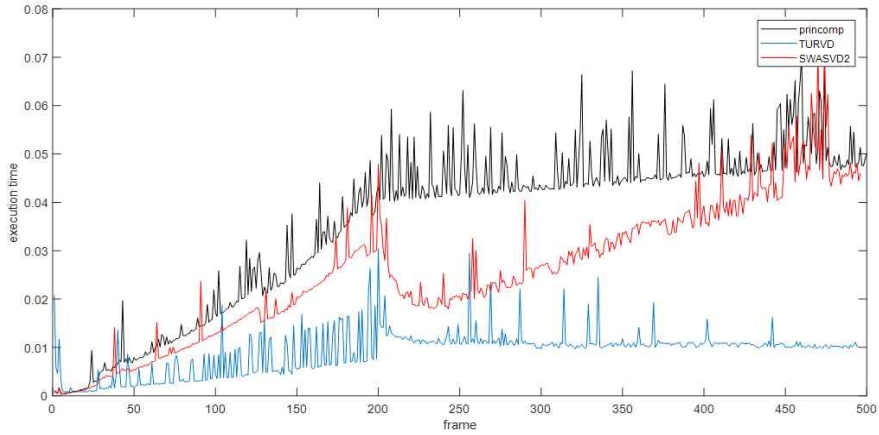
Fig. 3 Differences of the wrapping parameter between SWASVD2 and the proposed algorithm with log10 scale compared to the parameters from Matlab's 'princomp' function.

SWASVD2는 공분산 행렬 형태의 업데이트가 불가능하기 때문에 수식 (6)으로 계산한 행렬을 SWASVD2로 업데이트 한 후, 3.2장의 rank-1 업데이트 알고리즘을 적용하였다. TURVD와 SWASVD2의 절단 임계치 ε은 모두 20으로 설정하였다.

Fig. 3은 식 (1)에 나타난 템플릿 형태 표현을 위한 파라미터 $\boldsymbol{p}$ 값을 두 알고리즘으로 계산한 후 'princomp' 함수를 사용하여 나타난 $\boldsymbol{p}$ 값을 기준으로 각각 얼마나 차이가 나는지를 표현하고 있다. 예를 들면 p_5, p_6는 각각 템플릿의 x와 y축의 이동(translation) 변화를 나타내는데 기준 $\boldsymbol{p}$ 값과 비교하여 제안한 알고리즘과 SWASVD2에서 계산된 차이가 대부분의 프레임에서 1 픽셀 이상 차이가 나타나고 있지 않았다. 또한 그 외의 파라미터 값들도 결과에 큰 영향을 미칠만한 차이를 발견하지 못하였다. 또한 Fig 4와 5는 각각 매 프레임마다 기준 알고리즘과 SWASVD2, 제안한 알고리즘의 계산 시간, 계수값의 변화를 함께 표시하여 성능을 비교하였다. Fig 3와 4를 통해서 제안한 알고리즘이 일반적인 주성분 분석을 이용하여 템플릿을 추적하는 방식에 비교하여 유의미한 차이가 나타나지 않으면서도 훨씬 빠른 계산 속도를 가지는 것을 알 수 있었다. 또한 Fig 5에서 나타난 바와 같이, 제안된 알고리즘이 적은 계수값으로 충분한 성능을 보이고 있는 것과는 달리 SWASVD2이 프레임이 진행될수록 오차가 누적됨에 따라 유사한 객체 추적 성능을 위해 형태 벡터의

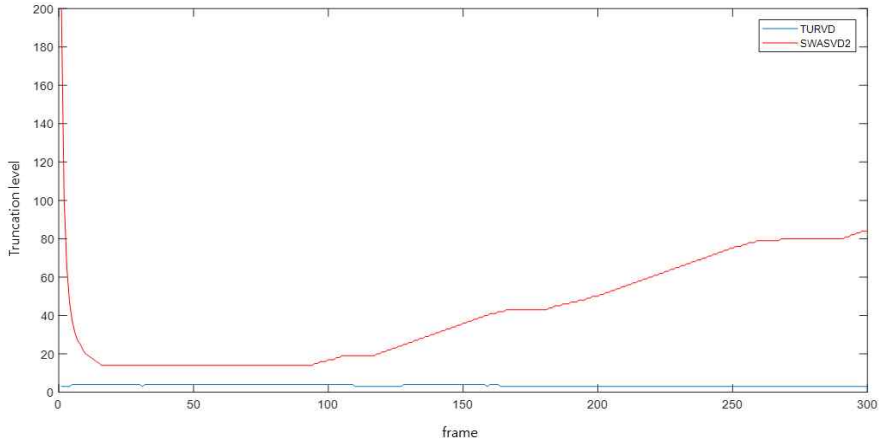


(a) PETS2001 with dataset 5.

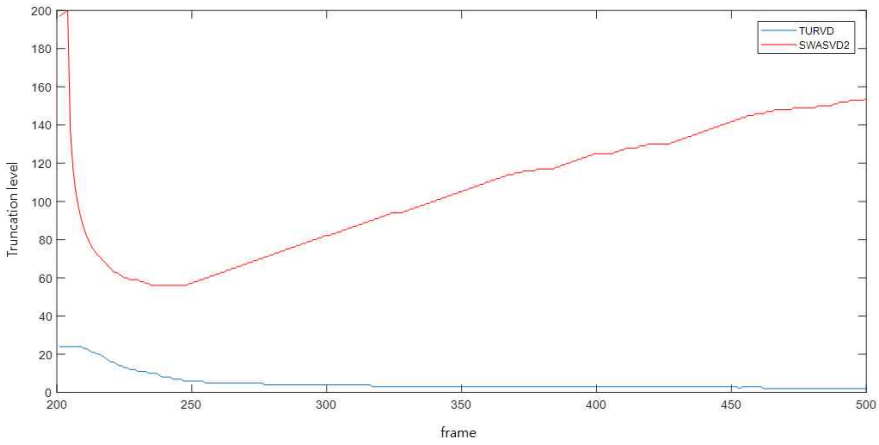


(b) PETS2006 with dataset 2.

Fig. 4. Comparison of execution time among Matlab's 'princomp', SWASVD2 and the proposed algorithm when the templates are updated.



(a) PETS2001 with dataset 5



(b) PETS2006 with dataset 2.

Fig. 5. Comparison of truncation level between SWASVD2 and the proposed algorithm when the templates are updated.

차원수가 늘어나는 것으로 유추할 수 있다. 행렬의 계수가 작으면 형태 벡터의 차원수를 작게 관리할 수 있어 그만큼 연산시간과 메모리 사용 측면에서 매우 유리하다.

5. 결 론

본 논문에서는 템플릿 추적을 위한 빠르고 효율적인 Sliding window 기반 TURVD 알고리즘을 제안하였다. 일반적인 템플릿 추적 문제에서 정확도를 높이기 위해 형태 이미지를 사용하는데 이때 필요한 주성분을 계산하는 알고리즘은 $O(n^3)$의 계산 복잡도를 가지기 때문에 실시간 계산 환경에 부적합하다. 그러나 TURVD 알고리즘은 데이터가 업데이트 되었을 때에도 기존 TURVD를 재활용하여 빠르고 효율적으로 계산할 수 있다. 본 논문에서는 이를 위해 Sliding Window 방식으로 템플릿 이미지 행렬이 업데이트 되었을 때 TURVD를 업데이트 하는 방식과 공분산 행렬 계산을 위한 효율적인 rank-1 행렬 업데이트 알고리즘을 제안하였다. 실험 결과 기존의 방식들과 비교하여 정확도 면에서 유의미한 차이가 없었으며 계산 속도는 훨씬 빠른 것을 발견하였다.

REFERENCE

[1] I. Whoang and K.N. Choi, “An Algorithm for Color Object Tracking,” *Journal of Korea Multimedia Society*, Vol 10. No. 7, pp. 827-837, 2007.

[2] I. Matthews, T. Ishikawa, and S. Baker, “The

Template Update Problem," *IEEE Transaction on Pattern Analysis and Machine Intelligence*, Vol. 26, No. 6, pp. 810-815, 2004.

[3] R. Badeau, G. Richard, and B. David, "Sliding Window Adaptive SVD Algorithms," *IEEE Transaction on Signal Processing,* Vol. 52, No. 1, pp. 1-10, 2004.

[4] M. Brand, "Fast Low-Rank Modification of the Thin Singular Value Decomposition," *Linear Algebra and Its Applications*, Vol. 415, No. 1, pp. 20-30, 2006.

[5] J.R. Bunch and C.P. Nielsen, "Updating the Singular Value Decomposition," *Numeriche Mathematik*, Vol. 31, No. 2, pp. 111-129, 1978.

[6] G. Lee and J.L. Barlow, "Updating Approximate Principal Components with Applications to Template Tracking," *Numerical Linear Algebra with Applications*, Vol. 24, No. 2, pp. e2081, 2017.

[7] J. Weng, Y. Zhang, and W.S. Hwang, "Candid Covariance-Free Incremental Principal Component Analysis," *IEEE Transaction on Pattern Analysis and Machine Intelligence,* Vol. 25, No. 8, pp. 1034-1040, 2003.

[8] H. Zhao, P.C. Yuen, and J.T. Kwok, "A Novel Incremental Principal Component Analysis and Its Application for Face Recognition," *IEEE Transaction on System Mans, and Cybernetics*, Vol. 36, No. 4, pp. 873-886, 2006.

[9] B. Lucas and T. Kanade, "An Iterative Image Registration Technique with an Application to Stereo Vision," *Proceeding of Defense Advanced Research Projects Agency Image Understanding Workshop*, pp. 674-679, 1981.

[10] G.H. Golub and C.F. Van Loan, *Matrix computations*, The Johns Hopkins Press, Baltimore, 2013.

[11] G.W. Stewart, "Updating a Rank-Revealing ULV Decomposition," *Journal of Society for Industrial and Applied Mathematics Matrix Analysis and Applications*, Vol. 14, No. 2, pp. 494-499, 1993.

[12] G.W. Stewart, "An Updating Algorithm for Subspace Tracking," *IEEE Transaction on Signal Processing*, Vol. 40, No. 6, pp. 1535-1541, 1992.

[13] J.L. Barlow and H. Erbay, "Modifiable Low-Rank Approximation to a Matrix," *Numerical Linear Algebra with Applications*, Vol. 16, No. 10, pp. 833-860, 2009.

이 근 섭

2000년 2월 한양대학교 전자전자통신전파공학과군 학사
2002년 2월 한양대학교 전자통신전파공학과군 석사
2013년 8월 펜실베니아주립대학교 Computer Science & engineering 박사
2013년 9월~2014년 8월 펜실베니아 주립대학교 연구원
2014년 9월~2015년 10월 루벤카톨릭대학교 연구원
2016년 1월~2018년 2월 삼성전자 책임 연구원
2018년 3월~현재 한국외국어대학교 Global Business & Technology 교수
관심분야 : Numerical Linear Algebra, 영상 처리