[18] X. Nie, S. Chen, and R. Hamid, "A Robust and Efficient Framework for Sports-Field Registration," in *Proc. IEEE/CVF Winter Conference on Applications of Computer Vision*, Virtual, 2021, pp. 1936-1944.
[19] K. He, G. Gkioxari, P. Dollar, and R. Girshick, "Mask R-CNN," in *Proc. IEEE International Conference on Computer Vision*, Venice, Italy, 2018, pp. 2961-2969.
[20] Y. Takezawa, M. Hasegawa, and S. Tabbone, "Robust Perspective Rectification of Camera-Captured Document Images," in *Proc*. IAPR International Conference on Document Analysis and Recognition, Kyoto, Japan, 2017, pp. 27-32, DOI: 10.1109/ICDAR.2017.345.

## APPENDIX A

Vertex matching

Procedure 1: The detected vertices are clustered by their x and y coordinates, and the cluster numbers are assigned in ascending order of the coordinate values. Let c and r be the number of clusters for the x and y coordinates, respectively. The vertex labels are stored in the matrix $\mathbf{M}$ of size r × c according to each cluster number, where the cluster numbers for the x and y coordinates are the column and row numbers, respectively (e.g., see Fig. 7). The vertex labels of the court model are stored in an 8 × 6 or 6 × 8 matrix $\mathbf{M}_{\mathrm{m}}$: the size of $\mathbf{M}_{\mathrm{m}}$ is selected manually according to the court orientation. Zero is inserted into each empty element.

Procedure 2: The function *elasticMatching* is executed to obtain a matrix $\mathbf{M}_{\mathrm{Ex}}$ with the rows and columns stretched so that the non-zero elements of $\mathbf{M}$ match $\mathbf{M}_{\mathrm{m}}$. /* Because $\mathbf{M}$ is a matrix obtained by removing the rows and columns with 0 elements from $\mathbf{M}_{\mathrm{Ex}}$, $\mathbf{M}$ and $\mathbf{M}_{\mathrm{m}}$ correspond to each other. */

```
Function elasticMatching (M, M_m){
    matrix M_Ex
    for i = 1, 2 do{
        if M(0,0) ≠ 0 then M_Ex = ex00(M, M_m)
        else if M(0, c − 1) ≠ 0 then M_Ex = ex01(M, M_m)
        else if M(r − 1,0) ≠ 0 then M_Ex = ex10(M, M_m)
        else if M(r − 1, c − 1) ≠ 0 then M_Ex = ex11(M, M_m)
    }
    return M_Ex /* the matrix indicates matching result */
}

Function ex00(M, M_m){
    dynamic array v_0, v_1, v, w
    for y = 0, ⋯, M_m.rows − 1 do{
        for x = 0, ⋯, M_m.columns − 1 do{
            if M(0,0) = M_m(y, x) && y + r ≤ M_m.rows && x + c ≤
            M_m.columns then{
                createMatrix(M_Ex, (r + y)rows, (c + x)columns) /*
                initialized with zero */
                copyMatrix(M, M_Ex(range(y, y + r − 1), range(x, x + c −
                1)) /* copyMatrix(source, destination) */
                v_0.push(M_Ex)
                M_Ex.release
            }
    /* Column extension */
    for j = 0, ⋯, v_0.size − 1 do{
        v_1.push(v_0[j])
        for x = 1, ⋯, c − 1 do{
            for k = 0, ⋯, v_1.size − 1 do{
                if M(0, x) ≠ 0 then{
                    for all x_m such that: M(0, x) ≡ M_m(y_Ex, x_m(≥ x_Ex)),
                    where the (y_Ex, x_Ex) element of v_1[k] corresponds to the
                    (0, x) element of M.
                        copyMatrix(v_1[k], M_Ex)
                        /* insertColumns(destination, column, number of
                        columns) inserts the specified number of zero-padded
                        columns before the specified column. */
                        insertColumns(M_Ex, x_Ex, x_Ex − x_m)
                        w.push(M^Ex)
                        M^Ex.release
                }
                else{
                    for all x_m such that: M(y_nz, x) ≡ M_m(y_nz_Ex +
                    Δy(> 0), x_m(≥ x_Ex)), where the (y_nz, x) element of M
                    is the first nonzero element at the x column and the
                    (y_nz_Ex, x_Ex) element of v_1[k] corresponds to the
                    (y_nz, x) element of M.
                        copyMatrix(v_1[k], M_Ex)
                        insertColumns(M_Ex, x_Ex, x_Ex − x_m)
                        w.push(M_Ex)
                        M_Ex.release
                }
            }
            v_1.clear
            for k = 0, ⋯, w.size − 1 do{
                v_1.push(w[k])
            }
            w.clear
        }
        for k = 0, ⋯, v_1.size − 1 do{
            v.push(v_1[k])
        }
        v_1.clear
    }

    v_0.clear

    for j = 0, ⋯, v_2.size − 1 do{
        v_0.push(v[j])
    }
    v.clear

    /* Row extension */
    for j = 0, ⋯, v_0.size − 1 do{
        v_1.push(v_0[j])
        for y = 1, ⋯, r − 1 do{
            for k = 0, ⋯, v_1.size − 1 do{
                (omitted)
            }
        }
    }

    if v.size > 1 do
        filter(v) /* The image is homography transformed based on each
        correspondence that v[k] indicates, then the correspondence that
        maximizes the degree of coincidence between the color figures of
        the transformed image and the color figures of the court model is
        selected. Finally, copy it to v[0]. */

    return v[0]
}

Function ex01(M, M_m){
    (omitted)
    return v[0]
}

Function ex10(M, M_m){
    (omitted)
    return v[0]
}

Function ex11(M, M_m){
    (omitted)
    return v[0]
    }
```