Data Mining Atomically Resolved Images for Material Properties

Sebastian Klaassen
Department of Visualization and Data Analysis
University of Vienna
Universitätsring 1, 1010 Wien, AT
sebastian.klaassen@univie.ac.at

***Abstract*—Feature detection is a widely used technique for atomically resolving localized imaging. Capturing atomic motion in spectroscopy for observation and quantitatively correlating structure-property relationships with functionality generates insight into physics and chemistry of 2D materials. This paper presents a set of algorithms for tracking motion of molybdenum and selenium atoms, captured throughout a time sequence of scanning transmission electron microscopy images for density functional theory simulations. The discussed set of algorithms efficiently detects, classifies and tracks atoms in the MoSe2 dataset of the 2017 Smokey Mountain Computational Science and Engineering Conference challenge.**

The accompanying video to this paper is located at https://www.dropbox.com/s/w1xiic5ecr9p6e9/MaterialMining.mp4

I. Introduction

The data challenge consists of 3 parts. Atom detection, atom classification and atom tracking. Implementations of each of the three parts are described in sections II, III and IV respectively. Section VI outlines potential areas of improvement.

II. Atom detection

Scanning transmission electron microscopy (STEM) produces grayscale images of individual atoms (figure 1a).
To detect atom centers, the image is matched with a feature template at every pixel. The feature template is a grayscale image of a sphere with a radius of 14 pixels (see figure 1b). Since features are rotation invariant, we can use cross correlation to perform this step. Cross correlation yields better results than other recognition techniques by using all of the information captured for an object [2]. The correlation outputs one residual value R (see equation 1) and one intensity value I (see equation 2) per pixel of the STEM image. In order to detect atoms closer than the radius of the atom template from the border, the cross correlation result is normalized with the number of pixels that have been compared. Detection accuracy is lower for such border-atoms. Only pixels within the circular region of the template sphere are compared.

Residual values from the cross correlation represent likelihood that an atom center resides at the corresponding location. Pixels are iterated in order of increasing R. The first iteration returns the pixel that's the most likely to contain an atom center. Subsequent iterations are also considered atom centers as long as they are at least 14 pixels away from previously detected atoms. A minimum distance of 14 pixels is used, rather than two times the template radius, because atoms can overlap in STEM images.

$$R_{x,y} = \frac{\sum^{i,j} (template_{i,j} - pixelx, y)^2}{n} \tag{1}$$

$$I_{x,y} = \frac{\sum^{i,j} template_{i,j} - pixelx, y}{n} \tag{2}$$

Atom centers with high residuals are false positives. We define a stable residual threshold and reject atoms above that threshold. The plot of residuals of all detected atom centers shows a sigmoid shaped function (see figure 1c). Atom centers below the center are true-positives, atom centers above the center are false-positives and atom centers around the center are border-atoms with a low prediction accuracy. The steep slope around the center indicates that the algorithm is stable over the choice of threshold.

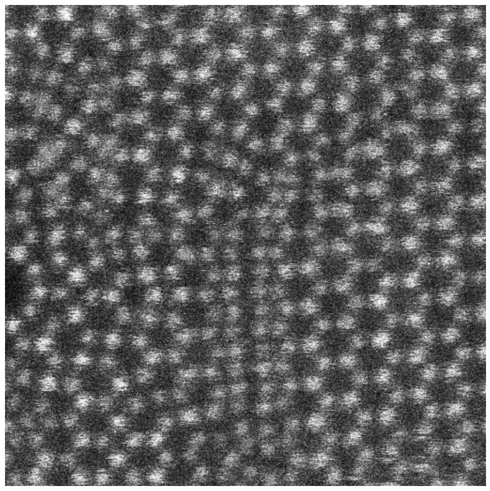
Atom intensities are stored along with atom positions for later classification.

III. Atom classification

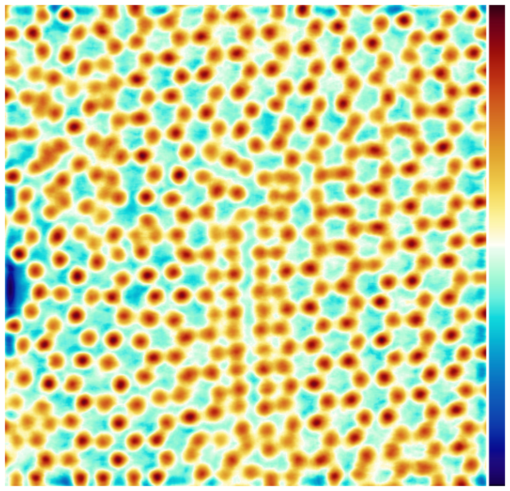
Images of the challenge dataset contain two types of atoms, selenium (SE) and molybdenum (MO). SE atoms are 8% brighter than MO atoms. This slight difference in brightness is insufficient to detect atom types from noisy STEM images. Atoms are part of hexagonal molecules, consisting of three SE and three MO atoms in alternating order. We determine which atoms represent SE and which atoms represent MO by comparing the relative brightness of atoms in a molecule. Atom brightness was detected during atom detection (see above). Molecules are detected by finding hexagonal formations of atoms.

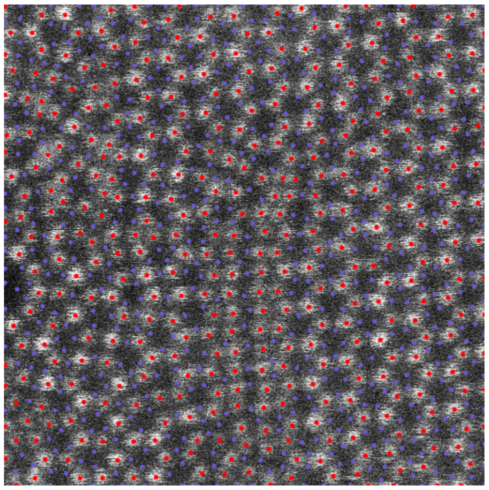
A. Molecule detection

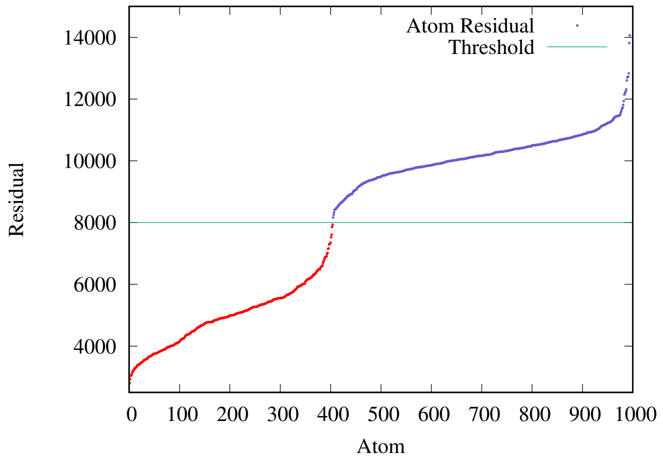
Molecules are formations of six atoms that are uniform, and have a small perimeter. Uniformity is high when all angles of the hexagon are close to 60°. The small perimeter condition is required to avoid hexagons that contain other atoms, while uniformity is suitable as a cost function. The general approach is similar to atom detection. Hexagons are extracted and added in order of increasing hexagon


(a)

(b)


(c)


(d)



(e)

Fig. 1. Atom detection: The input image (a) is convoluted with a template atom (b), yielding per-pixel residuals (c). The colormap shows low residuals in brown and high residuals in blue. Pixels are marked as atom centers in order of increasing residual and with a minimum distance of 14 pixels (d). Residuals above a threshold of 8000 are considered noise (d, e).

uniformity, such that they don't violate previously added features.

A STEM image with 400 detected atoms can theoretically contain $5.48 * 10^{12}$ unique hexagons. A strategy is required to restrict the number of hexagons considered. We detect only the smallest clockwise and the smallest counter-clockwise hexagon starting with each of the n atoms, resulting in $2n$ hexagons.

After sorting those $2n$ hexagons by uniformity, we remove duplicate hexagons. Since the list of hexagons is already sorted this step takes only linear time.

Next, we remove misaligned hexagons by enforcing two constraints. New hexagons cannot intersect with existing hexagons (intersection avoidance, see figure 2a) and the edges of new hexagons cannot form loops of length 3 with edges of existing hexagons (loop avoidance, see figure 2b). Loops of length 4 or 5 are not tested, because their detection is computationally far more expensive. Also length-3 loops are particularly harmful to atom classification, because they cannot be balanced with an equal amount of MO and SE atoms. Both intersection- and loop avoidance require a temporary data structure that maps vertices to hexagons intersecting that vertex. Each hexagon is validated once by both algorithms. The runtime is therefore again linear.

We implemented intersection avoidance on a hexagon h by computing two-dimensional cross products of the normalized hexagon edges to detect if any of the edges of existing hexagons lie inside the angle formed by two adjacent edges of h. This condition is violated by both of the white edges ending at the red circle in figure 2a.

We implemented loop avoidance on a vertex v_1 of a hexagon h, by considering both vertices that share an edge of h with v_1 as second vertex in a possible loop v_2. The two vertices that share an edge of a hexagon $\neq h$ with v_2 are considered as third vertex in a possible loop v_3. If another

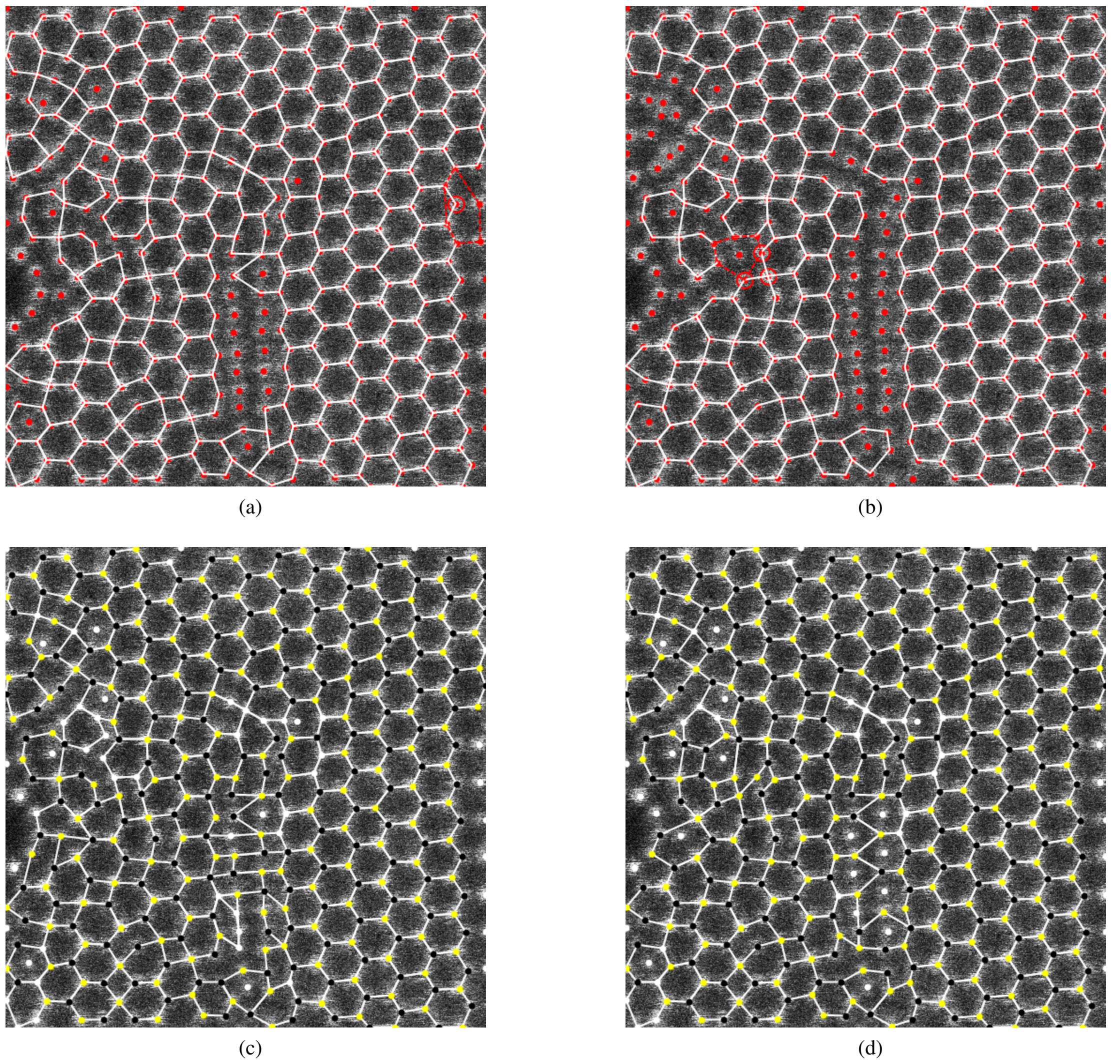


Fig. 2. Atom classification: Hexagons are added in order of decreasing uniformity. The red hexagon in (a) is removed because of an intersection marked with a red circle. The red hexagon in (b) is removed, because it would form a loop with a length of 3 edges around the marked vertices. The solution converges after 4 iterations (d). Selenium atoms are yellow, molybdenum atoms are black, unknown atoms are white. The result improves only marginally after the first iteration (c).

hexagon was found that connects v_3 with v_1 over a single edge, hexagon h is rejected. An example of such a scenario, with h drawn in red and vertices $v_{1\ldots3}$ marked with red circles, is shown in figure 2b. Typically vertices are shared between up to three hexagons. Loop avoidance is applied to all 6 vertices of h, each having 2 candidates for v_2. v_2 is typically shared with two more hexagons, each having 2 candidates for v_3. Again v_3 is typically shared with two more hexagons, each having 2 candidates that might coincide with v_1. This results in a maximum of 192 possible loops that can be formed with h.

Optionally, after removing hexagons in the last step, we detect new hexagons with the same start-vertex and rotation direction, but slightly longer perimeter than the removed vertex. The list of newly found hexagons is then added to the other hexagons, followed by another iteration of duplicate removal and misaligned hexagon removal. After each such iteration we compare the quality of the computed hexagon list by classifying atoms (see below). Frame 1 of the challenge dataset converges after 4 iterations (see figure 2d). For this image, none of the hexagons generated after 4 iterations improve the solution.

B. Atom classification

After hexagons (i.e. molecules) are detected, each hexagon gets to vote on the atom type of its atoms. We sum up the brightness of the three even atoms and subtract the brightness of the three odd atoms. The magnitude of a vote is defined as the difference in brightness between even and odd atoms. As a classification quality measure we use the sum of magnitudes of votes of each hexagon.

For classification we compute per-vertex votes as the sum of votes of each hexagon that shares this vertex. Odd hexagons have negative votes so that higher per-vertex votes always represent higher brightness.

At this point each per-vertex' vote depends only on the votes of up to three hexagons that share this vertex. However, information over the entire image must be assimilated to get the most accurate solution. An incorrect vote on a hexagon negatively affects the votes of all hexagons that share vertices. We detect this case by combining per-hexagon error factors. The error factor of a hexagon is computed by comparing the hexagon's vote with the combined vote of neighbor hexagons. The error factor is negative if hexagon vote and combined neighbor vote contradict (i.e. have different signs).

We inverse the original vote of each erroneous hexagon and recompute per-vertex votes. Atoms with a per-vertex vote higher than a threshold T are classified as selenium, atoms with a vote lower than $-T$ are classified as molybdenum and atoms with a vote between $-T \leq v \geq T$ are classified as unknown.

IV. Atom tracking

It is important in material analysis to track atoms across multiple frames of STEM images to analyze their movement over time. This task is trivial if the frame rate of the captured image sequence is high enough that the maximum distance an atom travels between subsequent frames is below half the minimum distance between atoms. Unfortunately this is not valid for the challenge data set. It is insufficient to analyze each atom in isolation. We analyze the motion of an atom's neighborhood to guide motion tracking of that atom.

To track atoms across frames, we find a mapping M between atom positions of two consecutive frames P_0 and P_1 according to equation 3.

$$M[p_0] = \begin{cases} p_1 & \text{if the atom moved from } p_0 \text{ to } p_1 \\ -1 & \text{if the atom disappeared between frames} \end{cases} \quad | \quad p_0 = \{1, 2, ..., |P_0|\}, p_1 = \{1, 2, ..., |P_1|\} \tag{3}$$

We compute a rank matrix R, so that $R[p_0, r]$ is the index of the r-th nearest point in P_1 to p_0. The trivial case described above, where each point in P_0 maps to the nearest point in P_1 corresponds to the mapping $M[p_0] = R[p_0, 0]$.

There are $n!$ possible mappings M between two frames with n atoms. To find the best possible mapping, we define a graph search problem [4] as follows. Each unique mapping is a state and the cost of a state is a function of the total distance of the movements of all points between frames $\sum_{p_0}^{P_0} |p_0 - M[p_0]|$ and the number of unmapped points $\sum_{p_0}^{P_0} M[p_0] = -1$. The search space is too large to be fully traversed. Therefore, it is important to pick a good start state.

We assign each point of P_0 to the closest unassigned point in P_1. Similar to both atom detection and molecule detection, the best results are achieved by finding the ideal order in which points are assigned. The likelihood of a point in P_0 belonging to a certain point in P_1 is proportional to the distance between them. Points that most likely belong to their respective closest point in P_1 are assigned first, resulting in points of lower likelihoods to be assigned to the remaining points. The start state is shown in figure 3a.

Between frames 26 and 27 of the challenge dataset we observe the formation of a gap in the material, displacing many atoms in a small region. When close atoms are prioritized in this area, we first assign atoms from the outside of the gap to their nearest points in P_1, which leaves atoms in the center of the gap without nearby points. In such regions, a better strategy is to first assign points in the center of the gap to their nearest points in P_1, leaving points at the outside of the gap to be mapped to the remaining points. Therefore, we prioritize both atoms with close points in P_1, as well as atoms in regions of low P_1 point density (e.g. at the center of a gap). This improved start state is shown in figure 3b. P_1 point density at a point p_0 is computed by equation 4.

$$d_{p_0} = \sum_{p_1}^{P_1} \exp^{-\frac{|p_0 - p_1|^2}{2\sigma^2}} \tag{4}$$

We iterate through the search space starting from the start state by changing the mapping of one point at a time and recomputing the state cost. It is unfeasible to try out every combination of mappings. Instead we detect mapping anomalies in the current state to determine which mapping should be changed. Mapping anomalies are measured by comparing the direction and distance of a mapping $p_0 - P_1[M[p_0]]$ with the weighted average direction and distance of the mappings of nearby points. The point whose mapping distance and direction most significantly differs from its neighbors $\widetilde{p_0}$ is tested for replacement or removal. We compute $\widetilde{p_1}$ as the most likely destination of $\widetilde{p_0}$ according to the weighted average of mapping distances and directions of nearby points. This adds two more states to our search: the state where $M[\widetilde{p_0}] = \widetilde{p_1}$ and the state where $M[\widetilde{p_0}] = -1$.

Figure 3c shows the result of atom tracking after 64 iterations of local search (i.e. at every iteration we only continue with the lowest cost state). Figure 3d shows the result of atom tracking after 1367 iterations of uniform cost search (i.e. at every iteration we add all possible future states into a search queue and pop the lowest cost state from the queue). Uniform cost search achieves slightly higher accuracy, but at high runtime cost and the memory overhead of storing a queue of states. Therefore, we implemented local search.

V. Conclusion

The implementation of the described set of algorithms takes 17 seconds to detect atoms, 160 milliseconds to classify atom types and 7.5 seconds to track atoms over all 49 frames of the challenge dataset on a desktop computer with an Intel i7-2600K cpu running Ubuntu 17.04. Most time is spend on the computationally expensive cross correlation step. Some strategies to speed up cross correlation are referred to in section VI.

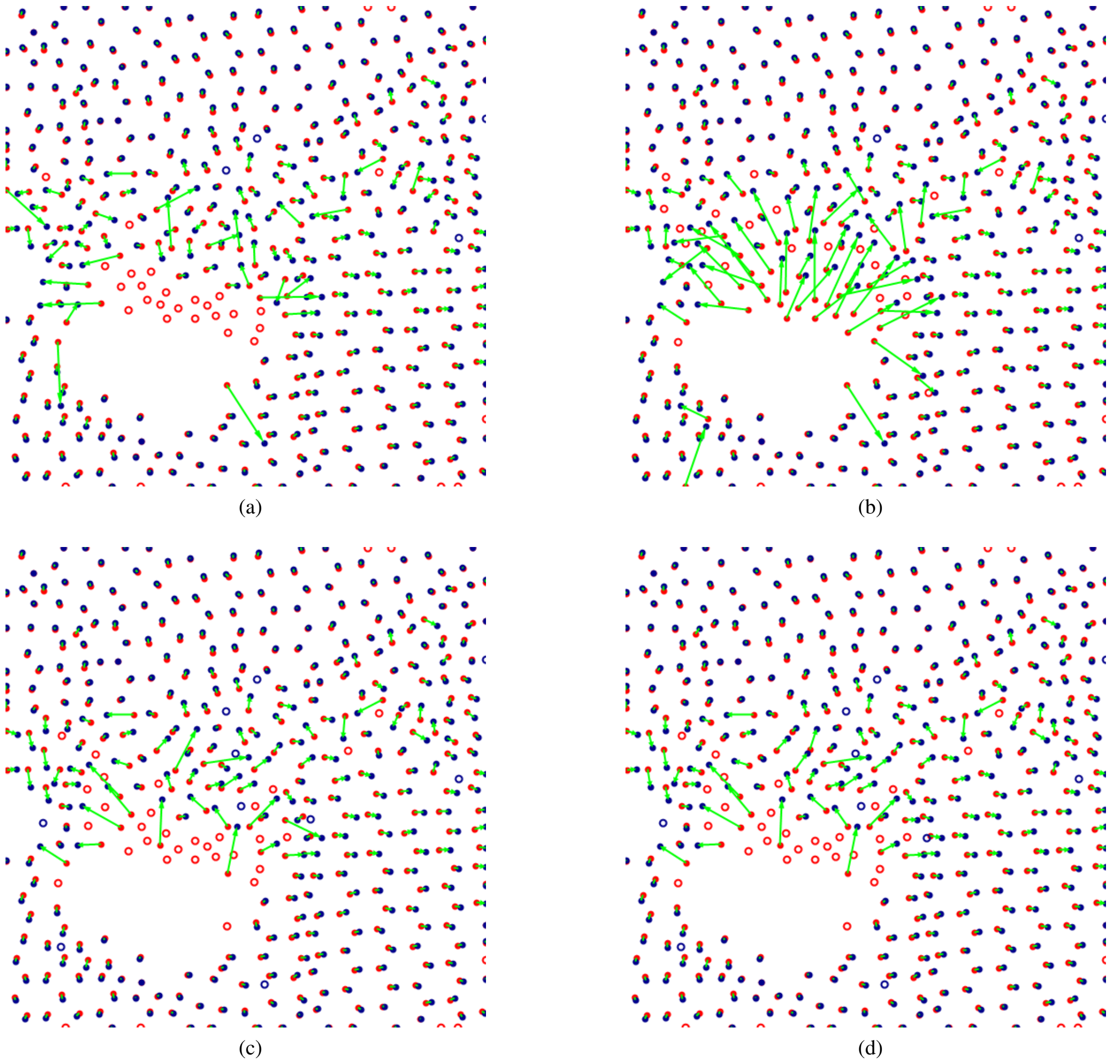


Fig. 3. Atom tracking: Atoms of the current frame (red points) are mapped (green arrows) to atoms of the following frame (blue points). Unmapped points are shown as circles. (a) shows points mapped to free targets in order of mapping distance. (b) prioritizes points within low target point density. (c) shows mappings after 64 rounds of anomaly detection. (d) shows mappings after 1367 iterations of anomaly detection using uniform cost search.

VI. Future work

Atom detection yields optimal results as long as atoms are at least 14 pixels apart (see figure 1d). Advanced techniques for template matching can compute the cross correlation faster in frequency domain [1], [3] or by computing algebraic moments of an integral image [5].

Errors in atom tracking appear in the form of atoms either disappearing or randomly jumping between frames, depending on the cost of unmapped atoms. Atom tracking accuracy depends on the accuracy of atom detection and on the quality of the search method for atom tracking. Figure IV shows that there are 27 less detected atoms in frame 27 (blue points) than in frame 26 (red points). In section IV we conclude that search agents such as uniform cost search are too slow and that the obtained improvement in accuracy is not significant. Further research in this area, however, might improve global search methods.

Solutions can be improved by incorporating derived information from one algorithm as input to another. For example, when a uniform pentagon is detected during atom classification, atom detection can be used to investigate whether another atom is hidden behind one of the pentagon's vertices. Also, after tracking atoms, atom classification can include hexagons of multiple frames to improve atom type prediction.

References

[1] *Template matching using fast normalized cross correlation*, volume 4387, 2001.

[2] Luke Cole, David Austin, and Lance Cole. Visual object recognition using template matching. In *Proceedings of Australian Conference on Robotics and Automation*, 2004.

[3] John P Lewis. Fast template matching. In *Vision interface*, volume 95, pages 15–19, 1995.

[4] Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach (2nd Edition)*. Prentice Hall, December 2002.

[5] Haim Schweitzer, J. W. Bell, and F. Wu. Very fast template matching. In *Proceedings of the 7th European Conference on Computer Vision-Part IV*, ECCV '02, pages 358–372, London, UK, UK, 2002. Springer-Verlag.