

# Visual Feature Extraction for Tool Construction Based on Geometric Reasoning

Cankut Bora Tuncer - 22001770, Ahmet Şahin - 21902702, Hikmet Şimşir - 21802880, İlyas Kocaer - 21202751  
Bilkent University, CS 554 - Computer Vision: Project Final Report

## I. INTRODUCTION

**P**RESENTLY, robots can interact with the environment where they operate and existing objects around them through various perception, manipulation, and locomotion capabilities. However, utilization of the existing objects is limited with predefined and predetermined possibilities due to the lack of reasoning. One step further, the creation of tools with existing primal objects for a specific purpose or functionality is also quite limited since the state-of-the-art techniques in robotics suffer from attaching meaning to corresponding objects. For these reasons, in the scope of the project, geometric reasoning will be searched to attach meaning to the existing objects and to investigate their potential use for higher-level relatively complex operations [1] [2].

## II. DESCRIPTION OF THE PROJECT

While *reasoning* can be achieved through different perspectives, in the scope of the project, visual reasoning will be taken into account for extraction of geometric and physical properties of corresponding objects. Tool construction and substitution processes can be analyzed with different levels of complexity by taking into account possibilities of object, action and effect equivalences such as object equivalence, object-action equivalence and object-action-effect equivalence solutions [1]. In this context, the answer for the question *how can a robot reach reasoning for geometric properties of an existing object to create a tool to use in a higher-level, more sophisticated task by proposing an object equivalence solution?* is searched.

In the following sections; generated dataset, implemented methods and correspondingly the obtained gradual outcomes, score computation approach and simulation based validation technique are described step by step respectively.

## III. DATASET

The overall dataset is composed from two categories as the reference tools and auxiliary objects. Although there exist tool datasets such as the ToolWeb dataset [3], we opt to generate our own toolset with the intention of being able to simplify and add additional tools to the dataset. The auxiliary objects are obtained from the GitHub repository [4]. In the dataset, there exist many daily life objects, spanning from foam to tongs. The dataset we have created includes many simplified versions of the tools such as a hammer, rake, axe, shovel, etc.



Fig. 1: The auxiliary object dataset (left), tool dataset (right).

## IV. METHODOLOGY

The methodology follows the approaches proposed in the article [1]. The dataset of the auxiliary objects is from the [4]. However, for the tools, we opt to generate our own dataset. The first step is to obtain the point clouds. Using the *Rai* [5] simulation environment, the point clouds are generated. The plane in the point clouds is removed by *RANSAC* and clustered using *Euclidean distance based clustering*. The point clouds for the objects are then fitted with *SuperQuadrics (SQ)* parameters. The *SQ* returns a set of parameters that describe its geometry and orientation: 3 or 4 for scale (representing the principal dimensions), 2 for shape variation (exponents  $\epsilon_1$  and  $\epsilon_2$ ), 3 for Euler angles (describing orientation), and 3 for central points (defining the position) [6]. A constrained optimization process, specifically the *Trust Region Reflective (TRF)* method [7], is utilized to find the best-fit parameters for each object. Given the *SQ* parameters, the scoring function proposed in the article [1] is used to find the best 2 objects fitting the parameters of the reference tool. Finally, the alternative tool is constructed in the simulation using the best matching objects. It should be clearly emphasized that all the methods mentioned above are implemented from scratch.

### A. Segmentation

After the scene point cloud is generated, the available objects need to be segmented. To achieve this purpose; as the first step, the plane surface is removed and then, clustering for each object is realized. The plane segmentation is achieved via *RANSAC* [8] where a 3D planar surface can be modeled with the equation  $a \times x + b \times y + c \times z + d$ . *RANSAC* finds appropriate inliers fitting to the given model. When inliers are removed, only the point clouds for the objects remain in the scene.

The remaining point clouds are clustered into individual objects via *Euclidean distance clustering*. Given a radius and minimum neighbor count, the algorithm searches the number of neighboring points inside the radius around every point and the corresponding points are inserted into a cluster. Then, the



Fig. 2: The plane segmentation.

same process continues for the neighboring points until there are no available points inside the given radius. The segmented objects are returned as a dictionary where the keys are the names of the objects and values are the point cloud values. The names of the objects are found by getting a random sample from the point cloud and checking the environment to which object the point cloud belongs.



Fig. 3: The object clustering.

### B. Superquadric Fitting

The fitting of superquadrics to segmented point clouds optimizes their parameters to best represent the given data. The project considers four superquadric types: superellipsoids, hyperboloids, toroids, and paraboloids. This enables the algorithm to represent a diverse range of geometric shapes effectively.

Superquadrics are defined by a set of parameters  $\Lambda = \{a_1, a_2, a_3, \epsilon_1, \epsilon_2, \phi, \theta, \psi, K_x, K_y, p_x, p_y, p_z\}$ , where: -  $a_1, a_2, a_3$  are scaling parameters determining the size along each axis; -  $\epsilon_1, \epsilon_2$  define the shape variance or roundness; -  $\phi, \theta, \psi$  are Euler angles defining orientation; -  $K_x, K_y$  are tapering parameters along the x- and y-axes, respectively; -  $p_x, p_y, p_z$  represent the central position of the superquadric [9].

In our implementation, tapering parameters  $K_x$  and  $K_y$  are not used, reducing the parameter set to 11 for most superquadric types. For toroids, an additional scaling parameter  $a_4$  determines the size of the hole, bringing the total number of parameters to 12.

Superquadrics provide an inside-outside function  $F(x, y, z; \Lambda)$  that determines whether a point  $(x, y, z)$  lies inside ( $F < 1$ ), on the surface ( $F = 1$ ), or outside ( $F > 1$ ) the superquadric [6]. For fitting, we use the constraint  $F(x, y, z; \Lambda) = 1$  to ensure the points lie on the surface of the superquadric. Given a set of  $n$  3D points obtained from a range sensor, the best-fitting superquadric minimizes the following function:

$$\min_{\Lambda} \sum_{i=1}^n \left( a_1 \cdot a_2 \cdot a_3 \cdot (|F(x_i, y_i, z_i; \Lambda)^{\epsilon_1}| - 1)^2 \right)$$

subject to the following constraints on the parameters:

$$a_{\min} \leq a_1, a_2, a_3, a_4 \leq a_{\max},$$

$$\epsilon_{\min} \leq \epsilon_1, \epsilon_2 \leq \epsilon_{\max},$$

$$\phi, \theta, \psi \in [0, 2\pi], \quad p_x, p_y, p_z \in [p_{\min}, p_{\max}]$$

These bounds ensure physical validity and numerical stability during optimization. For example,  $\epsilon_1$  and  $\epsilon_2$  are constrained to avoid degenerate cases where  $\epsilon = 0$ , which would lead to a trivial cost of zero. Scaling parameters  $a_1, a_2, a_3, a_4$  are limited to physically plausible values, while position parameters  $p_x, p_y, p_z$  are bounded based on the range of the point cloud data.

This optimization is performed using the Trust Region Reflective (TRF) algorithm [7], which allows setting lower and upper bounds on parameters. The TRF method provides robust convergence properties suitable for this constrained optimization problem.

### Superquadric Fitting Results

The optimized superquadric is then represented as a point cloud and compared with the original. Residual error provides a quantitative measure of accuracy, while 3D visualization enables qualitative assessment. As shown in Figure 4, the algorithm effectively fits all four superquadric types—superellipsoids, hyperboloids, toroids, and paraboloids—demonstrating its precision and versatility across different geometric shapes.

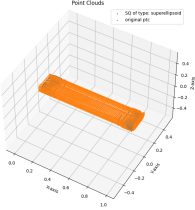
Residual errors, as shown in Table I, provide a quantitative comparison between our implementation and the base implementation. These results are calculated for the point cloud dataset. For superellipsoids, our approach results in an average residual error of 0.0329 compared to 0.0216 for the base implementation, indicating a slightly higher error in our method. Similarly, for paraboloids, the average residual error is 0.0285 for our approach compared to 0.0194 for the base, showing a similar trend. For hyperboloids, the residual error for our implementation is 0.0592, nearly double that of the base implementation (0.0304). This highlights the additional challenges in fitting shapes with sharp edges or elongated features. The largest discrepancy is observed for toroids, where our residual error is 0.1627 compared to 0.0498 for the base implementation. This significant difference can be attributed to the inherent complexity of fitting toroidal shapes, especially when considering the additional parameter  $a_4$  that determines the size of the hole.

### Limitations of Superquadric Fitting

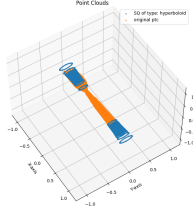
Despite its strengths, superquadric fitting has certain limitations. See Figure 5 for example bad fitting cases. One major limitation is its dependency on the shape of the point cloud. If the point cloud is not symmetric enough or lacks sufficient geometric features resembling a superquadric, the fitting process may fail to converge to a meaningful solution. This can result in inaccurate or nonsensical fits.

Another limitation arises from the use of gradient-based optimization solvers, such as the Trust Region Reflective algorithm. These solvers are sensitive to initial conditions and can often get stuck in local minima, especially in the presence of complex or noisy point clouds. This dependence on initialization makes the fitting process less robust, requiring careful selection of seed parameters or the use of multiple initializations to improve outcomes.

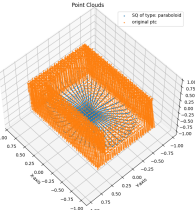
Overall, while our method performs competitively for simpler shapes such as superellipsoids and paraboloids, the residual errors for hyperboloids and toroids reflect the increased difficulty in accurately fitting more complex geometries.



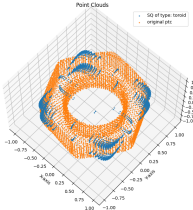
**Figure 4.A:** Superellipsoid fitted to hammer handle PC.



**Figure 4.B:** Superhyperboloid fitted to fork PC.



**Figure 4.C:** Superparaboloid fitted to hollow box PC.



**Figure 4.D:** Supertoroid fitted to metal nut PC.

Fig. 4: Example results of superquadric fitting applied to different point clouds.

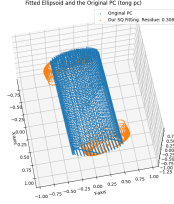
SQType	Avg. Base Residue Scores	Avg. Our Residue Scores
Superellipsoid	0.0216	0.0329
Hyperboloid	0.0304	0.0592
Toroid	0.0498	0.1627
Paraboloid	0.0194	0.0285

TABLE I: Residue Scores for Different Superquadrics Types Compared to Base Implementation. Average taken for all point clouds in the database.

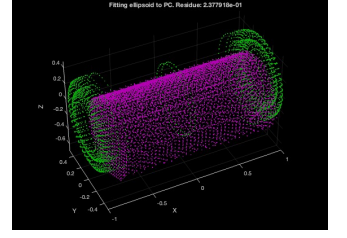
### C. Score Computation

The score computation and auxiliary object matching algorithm proposed in [1] is given in Algorithm 1. The algorithm first generates a permutation between a reference object and an auxiliary object. For the pair; shape error, scale error, and proportionality error are calculated below where  $r$  is the superquadric parameter of the reference and  $c$  for the candidate.

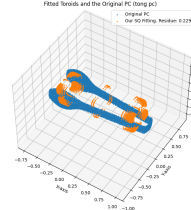
$$e_{\text{shape}} = \|[r_4 - c_4, r_5 - c_5]\|_2$$



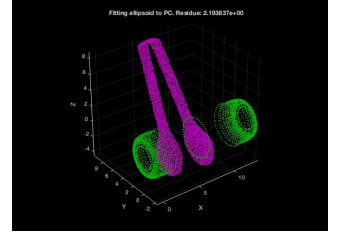
**Figure 5.A:** Superellipsoid fitted to cylinder point cloud using our implementation.



**Figure 5.B:** Superellipsoid fitted to cylinder point cloud using baseline implementation.



**Figure 5.C:** Supertoroid fitted to tongs PC using our implementation.



**Figure 5.D:** Supertoroid fitted to tongs PC using baseline implementation.

Fig. 5: Inconsistent fitting examples from the baseline and our implementation

$$e_{\text{size}} = \|[r_0 - c_0, r_1 - c_1, r_2 - c_2]\|_2$$

$$e_{\text{prop}} = \sqrt{(r_0/r_1 - c_0/c_1)^2 + (r_0/r_2 - c_0/c_2)^2 + (r_1/r_2 - c_1/c_2)^2}$$

After weighting the errors with importance weight (shape: 1, size: 1, prop: 0.5) (where it is found experimentally which fits to our own dataset while a similar approach is also employed in the referenced article [1]), the pairs are sorted according to their errors from least to most. In our implementation; we used predetermined attachment points, therefore we did not include the *AttachmentFit* algorithm [1] and attachment error in our implementation.

Best Handle	Best Head
Handle, Head	Total Error
drill bit, steel bowl	0.6228
wood L shape, steel bowl	0.6822
plastic clip, steel bowl	0.7705
drill bit, popcorn	0.7748

TABLE III: The scoring results with the reference object: spoon

In Table II, the best matching pair can function as a hammer when combined. However, the second-best pair has a plastic claw as a hammer head which is not quite feasible. In Table III, the best match for a spoon, drill bit and a steel bowl can indeed work as a spoon but combining a popcorn with a

---

**Algorithm 1: Parts Score Computation**


---

**Input:** Importance weights  $\lambda_1, \lambda_2, \lambda_3, \lambda_4$ , SQ parameters,  $T = \text{permute}(C, m)$   
**Output:**  $T^*$ , Att  
 $E \leftarrow []$ , Att  $\leftarrow []$ ;  
**for**  $i \leftarrow 1$  **to**  $|T|$  **do**  
  **for**  $j \leftarrow 1$  **to**  $m$  **do**  
     $e_{\text{shape}}^{T_i} += |\text{shape}(r_j) - \text{shape}(T_{ij})|$ ;  
     $e_{\text{scale}}^{T_i} += |\text{scale}(r_j) - \text{scale}(T_{ij})|$ ;  
    **for**  $k \leftarrow 1$  **to**  $m$  **do**  
      **if**  $k \neq j$  **then**  
         $e_{\text{ratio}}^{T_i} += |\text{rel}(r_j, r_k) - \text{rel}(T_{ij}, T_{ik})|$ ;  
      **end**  
    **end**  
  **end**  
   $e_{\text{att}}^{T_i}, A_{\text{closest}}^{T_i} \leftarrow \text{AttachmentFit}(T_i)$ ;  
   $e_{\text{const}}^{T_i} \leftarrow \lambda_1 e_{\text{scale}}^{T_i} + \lambda_2 e_{\text{shape}}^{T_i} + \lambda_3 e_{\text{ratio}}^{T_i} + \lambda_4 e_{\text{att}}^{T_i}$ ;  
   $E.\text{append}(e_{\text{const}}^{T_i})$ ;  
  Att.append( $A_{\text{closest}}^{T_i}$ );  
**end**  
 $T^* \leftarrow \text{sort}(T, E)$ ;  
**return**  $T^*, \text{Att}$ ;

---


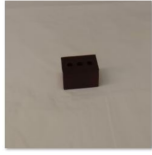
Best Handle	Best Head
	
Handle, Head	Total Error
wood L shape, purple foam	0.9486
wood L shape, plastic claw	0.9563
drill bit, purple foam	0.9625
drill bit, plastic claw	0.9702

TABLE II: The scoring results with the reference object: hammer

drill bit seems quite counterintuitive. However, when checking what a popcorn object is, it is a popcorn bucket toy that has a cavity underneath that can serve as a container. The overall disparity and outliers are mostly caused by our scoring weights, especially the proportional weight. Since the auxiliary object dataset and our tool dataset is not in the same scale, we had to make iterative changes with the proportional weights until the obtained outcomes are satisfactory and consistent. However, it should be noted that the current implementation with the corresponding optimization tends to lack the ability of generalization for distinct tool datasets but, on the other hand, it should not be overlooked that the nature of the physical problem of tool construction utilizes a decent level of over-fitting in a positive way to appropriately construct replacement tools. The generalizability capability (even though it can still possibly hold a certain level of over-fitting) could be improved with a larger dataset which includes geometrically distinct objects.

## V. TOOL CONSTRUCTION

Based on the obtained results which indicate the best match, the object pairs are combined with the use of a robotic arm in the Rai [5] simulation environment. Consequently, as one of the corresponding examples are presented in Fig. 6, tool construction is realized successfully as a tool validation step.



Fig. 6: The tool construction for the reference object: shovel.

## VI. CONCLUSION

In this project, we explored the potential of visual feature extraction and geometric reasoning for robotic tool construction. By leveraging segmentation, superquadric fitting, and scoring mechanisms; we demonstrated that without using deep learning or machine learning based techniques, it is possible to showcase computational creativity as accordingly to proposed targets. By implementing the aforementioned pipeline, we proved the validity of the corresponding outcomes. Given a reference tool, the framework is capable to find unusual pairs that can work for the intended use for a specific task. In addition to the successful results, the existing limitations and deficiencies are also discussed with their underlying reasons. Especially, improving the scale of the utilized dataset which will cover higher number of distinct objects can be stated as the future work for the current implementation while higher-level of reasoning capabilities which holds a more complex inference capacity such as *object-action-effect* equivalence solutions can be proposed as a future target for the methodology.

## REFERENCES

- [1] Lakshmi Nair, Jonathan Balloch, and Sonia Chernova. Tool macgyvering: Tool construction using geometric reasoning. In *2019 international conference on robotics and automation (ICRA)*, pages 5837–5843. IEEE, 2019.
- [2] Lakshmi Nair, Nithin Shrivatsav Srikanth, Zackory Erickson, and Sonia Chernova. Autonomous tool construction using part shape and attachment prediction. In *Robotics: Science and systems*, volume 2, 2019.
- [3] Paulo Abelha. Toolweb: A dataset of 116 synthetic household tool point clouds with mass and affordance ground truth for five tasks. <https://github.com/pauloabelha/ToolWeb>, 2017. Accessed: 2024-12-03.
- [4] Lakshya Nair. Robogyver: Tool macgyvering. <https://github.com/lnairGT/Robogyver-Tool-Macgyvering/tree/master>, 2023. Accessed: 2024-12-03.
- [5] M. Toussaint. Robotic systems toolbox. GitHub repository. Available: <https://github.com/MarcToussaint/robotic>.
- [6] Y. Wang, J. Zhang, and L. Chen. Robust and accurate superquadric recovery: a probabilistic approach. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1234–1242, 2023.
- [7] SciPy Developers. “scipy.optimize.least\_squares - scipy v1.11.3 manual”. [https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.least\\_squares.html#r20fc1df64af7-stir](https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.least_squares.html#r20fc1df64af7-stir). Accessed: December 24, 2024.
- [8] Martin A Fischler 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):381–395, 1981.
- [9] Paulo Abelha, Frank Guerin, and Markus Schoeler. A model-based approach to finding substitute tools in 3d vision data. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2471–2478, Stockholm, Sweden, May 16–21 2016. IEEE.