# Object Classification from Tactile Data

**Alanna Manfredini**
Department of Mechanical Engineering
Duke University, United States
alanna.manfredini@duke.edu

**Abstract:** In an effort to increase the effectiveness of human, robot interactions this paper aims to use tactile sensing to create a pointcloud of the object and subsequently use established algorithms to classify the object. By creating a pointcloud of the interactions between the robot and the object, it is possible to take advantage of well established classification algorithms commonly used for vision systems. To further increase the robustness of the system, the pointcloud is created without extraneous tactile and force data, which, in reality would need to come from delicate and costly sensors. The experimental results show that using Reinforcement Learning has large benefits on collecting data for analysis. To see the code for this project, visit https://github.com/AlannaMan/TactileObjectClass.

**Keywords:** Tactile, Robots, Learning

## 1 Introduction

There has been a lot of research into improving computer vision and using algorithms to interpret the environment with cameras. However, there are a lot of environments wherein humans do not make decisions solely based off their vision. These situations could occur because cameras are not feasible due to occlusions, dirt or low light levels. Additionally, improving the machine learning techniques for robots to understand their environment using tactile sensors means robots can take advantage of the inbuilt tactile cues evident in the human environment; for some tasks, robots may be able to use vision while simultaneously using tactile sensing to perform another task.

The current work that exists in the tactile sensing domain frequently aims to use force sensors, object normals and texture sensors to identify the objects with which the the robot is coming in contact. These sensors are very delicate and require a lot of maintenance over the period of training and use. To avoid this lack of robustness, this paper aims to build upon the work done identifying point clouds from vision systems by only using the contact point between the robot and the object rather than the additional extraneous data. If this were to be successful, the field of vision and touch sensing could be combined and could thus quickly accelerate the research into touch sensing to a scope comparable to vision sensing. Additionally, by collecting data in the same structure as the data inputted into object classification algorithms for vision, the amount of data to effectively train a classification algorithm is negligible since it can potentially be pre-trained on vision datapoints and then only use a small amount of physical data to create a more robust model.

## 2 Related Work

There has been relatively limited research into the field of tactile sensing for robot learning compared to the volume of research on object identification by vision. A couple of notable works that are in the tactile sensing field are *Learning to Identify Object Instances by Touch: Tactile Recognition via Multimodal Matching* [1] and *Tac2Pose: Tactile Object Pose Estimation from the First Touch* [2].

The *Learning to Identify Object Instances by Touch: Tactile Recognition via Multimodal Matching* uses GelSensors to get tactile and force information from object interactions. The GelSensors pro-

duce an image, which the researchers process with a Convolutional Neural Network (CNN). This CNN identifies the features in the GelSensor image and predicts an image of the related object. By comparing images, it merely classifies the objects and does not construct a 3D representation of the object which is highly critical for a robot interacting with an environment. Additionally, the project collects extra information such as tactile data, which provides another feature the model can work with to get an accurate classification. By doing this, the data is not in the same form as the vision classification algorithms and thus cannot take advantage of the large amount of research in that field. These are both things which our research aims to implement.

As mentioned in *Tac2Pose: Tactile Object Pose Estimation from the First Touch*, most of the previous work which combines tactile and vision merely uses the tactile aspect to refine the vision prediction with a binary prediction. The Tac2Pose paper and other papers referenced within it aim to classify the pose of the object that is being touched. That means they use very few touches to get a general outline of a small shape; our research aims to classify large objects by touching many locations around the object before using an algorithm to determine the large-scale shape and pose of the object. This vastly increases the number of objects with which it can classify and interact.

The paper that is the most similar to our work is *Tracking objects with point clouds from vision and touch* [3]. This paper creates a pointcloud from tactile sensing and uses this to classify a model, however, the pointclouds are supplemented with vision data and the sensor used is the GelSensor which incorporates additional tactile data rather than just point locations.

# 3 Methodology

The process for this experiment is broken up into three parts: Locating the object, collecting pointcloud data from an object, and classifying the object. The parts were performed in a Pybullet environment and coded with python.

## 3.1 Locating the Object

To setup the environment, a robot xarm was installed into a Pybullet environment with a cube loaded nearby within reach. Initially, the first stage of locating the object was skipped by hard-coding the robot's movements to move towards the cube. Later in the development process, a "searching" algorithm was created which was to have the robot locate the object from a random starting position. This was done by programming in a circular traverse. The traverse would return the robot back to a standard position close to the base and perform a circular traverse at increasing radii from the base of the robot. Depending on the direction of the traverse, clockwise or counter-clockwise, once there was a collision, the robot would then start to traverse the object collecting points in the same direction as the previous motion.

## 3.2 Collecting Pointcloud Data

### 3.2.1 Traversal Algorithm

A couple of methods were trialled to effectively collect the pointcloud data. The robot was programmed to traverse the face of the cube by establishing a starting point from the first collision. The robot then retreated in the direction it came 0.05 units, moved 0.05 units to the appropriate direction, and moved back towards the object. In different versions of the program, the robot moved back towards the object by either moving parallel to the direction of the object normal, by aiming for a specific offset from the previous collision, or by moving parallel to the previous approach. The various methods were chosen to minimise drift. It was noticed throughout the pointcloud collection process that errors between the desired robot position and the real robot position would quickly accumulate and cause the robot to quickly move away from the object rather than staying in a similar location and collecting datapoints.

To avoid drift, a constant y and z variable was kept for each traversal step. Whilst traversing horizontally, although the intended next position was created with respect to the most recent collision point, the z value was kept constant so the next intended location was not higher than the previous one due to the collision being higher than anticipated.

Initially, the object interaction algorithm was structured as a state machine with each of the components of the data collection process as a different state with various conditions leading between the states. The structure of the state machine can be seen in Figure 1.

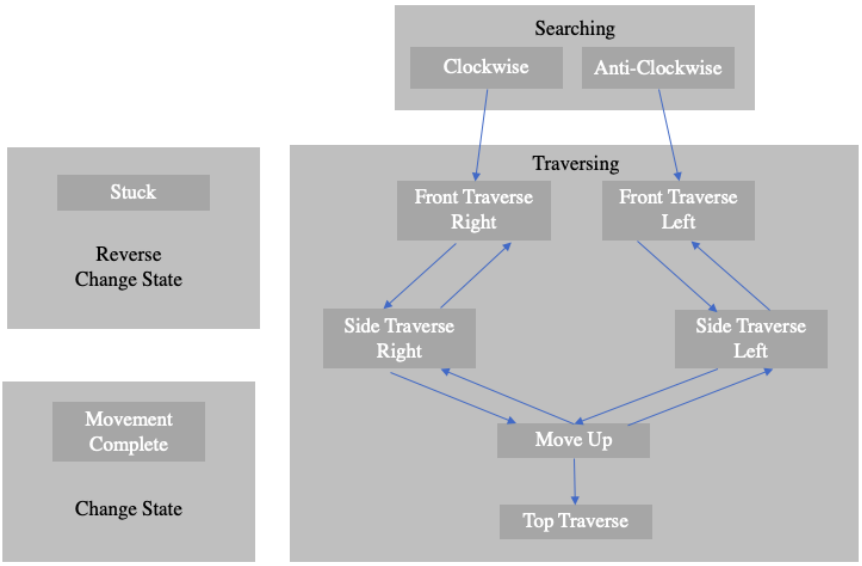


Figure 1: State Machine Structure

The main stages of the state machine are when the robot is searching for the object and when it is traversing the face of the object. A high level overview of the face traversal stage is that the robot collides with the object, the robot reverses away from the object, adjusts its position and then re-approaches. When the robot approaches the next location but doesn't make contact, the state machine enters the state called "movement complete." This indicates that more than likely the traversal has extended further than the face of the object and should therefore enter a new traversal state where it turns and aims to traverse the face of the object perpendicular to the face it was just traversing. The other important state is the "stuck" state. This is when the robot is aiming to move but does not change location because it has had a collision with the object (or itself) in a way that means it cannot move. When this state is entered, the robot returns in the direction it came by executing the prior movements that have been stored in a deque. After returning to what is hopefully a valid prior position, a new state is entered.

It can be visualised in Figure 2 that if the robot were to initially come into contact with the outermost point of a convex surface, when reversing back to the location of the surface slightly further in the positive y direction but in the same x and z location, the robot could reach its desired location but sightly in front of the surface it was aiming to collide with. To avoid this, the robot moves further forwards than the previous collision point; the retreat distance is 3 times the approach distance.

### 3.2.2 Reinforcement Learning (RL)

The final step was to code a reinforcement learning algorithm that teaches the robot to effectively and efficiently traverse the object whilst collecting the maximum number of points. This was not part of the intended process initially, but after struggling to collect enough datapoints to perform PointNet++ on, it became an important feature.

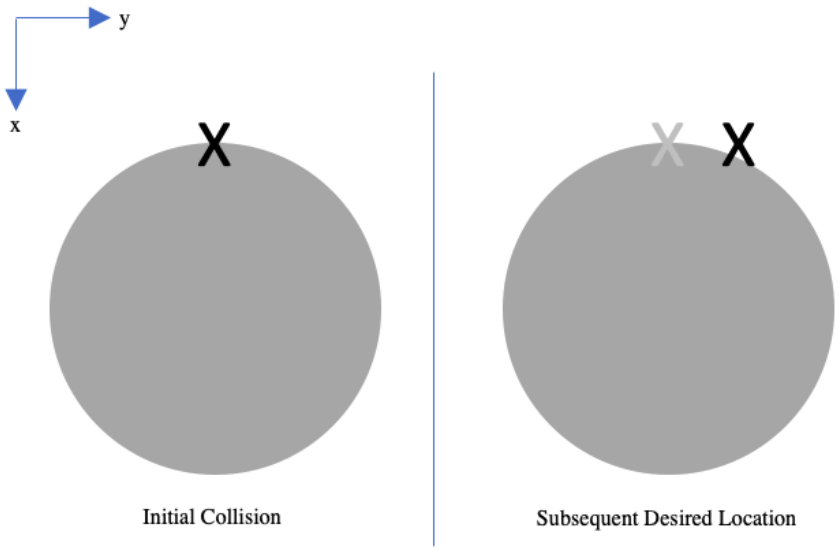


Figure 2: Convex Position Target Problem without Adjustment

A Proximal Policy Optimisation (PPO) algorithm was chosen because the reward function is simple and PPO is an algorithm well known for being stable. PPO is also good at maximising the reward. The relevant rewards used in the training of this reinforcement algorithm are rewards for touching the object and backing away from the box once it has been touched, and penalties for touching the object with another part of the robot, retouching the same spot and getting too far away from the previous collision.

$$R_{shaping}(s) = -d(s)$$

where d(s) is the distance between the current state and the previous collision.

$$\begin{aligned} R(s,a)_{intrinsic} = &-3 \times \mathbb{1}_{\text{non end-effector robot-object collision}} + 4 \times \mathbb{1}_{\text{backing off box after collision}} \\ &+ 5 \times \mathbb{1}_{\text{collision after not touching box}} + -1 \times \mathbb{1}_{\text{same spot collision}} \end{aligned}$$

$$J(\theta) = (E)[\Sigma_{t=0}^{\infty} \gamma^t R(s_t, a_t)]$$

### 3.3 Classifying Objects

Unfortunately, collecting enough points to resemble an object was considerably more difficult than anticipated. This meant that as of this paper, there had been no testing on classification by PointNet++. More details about how the algorithm works can be found at the original paper [4].

## 4 Experimental Results and Discussion

During experimentation, a couple different pointclouds were created. The first pointcloud was from using the traversal algorithm; in this pointcloud the highest density of the points is located towards the bottom of the object, which is because that is the location which the searching algorithm would first come into contact with the object. Then as the traverse progressed higher up the object, the

robot started to accumulate more drift in the algorithm an thus the distance between points increased and thus the density of the pointcloud decreased.

On the other hand, when using the RL algorithm, the high density area of the pointcloud was near the top edge, because this is where the learning algorithm initialised to.

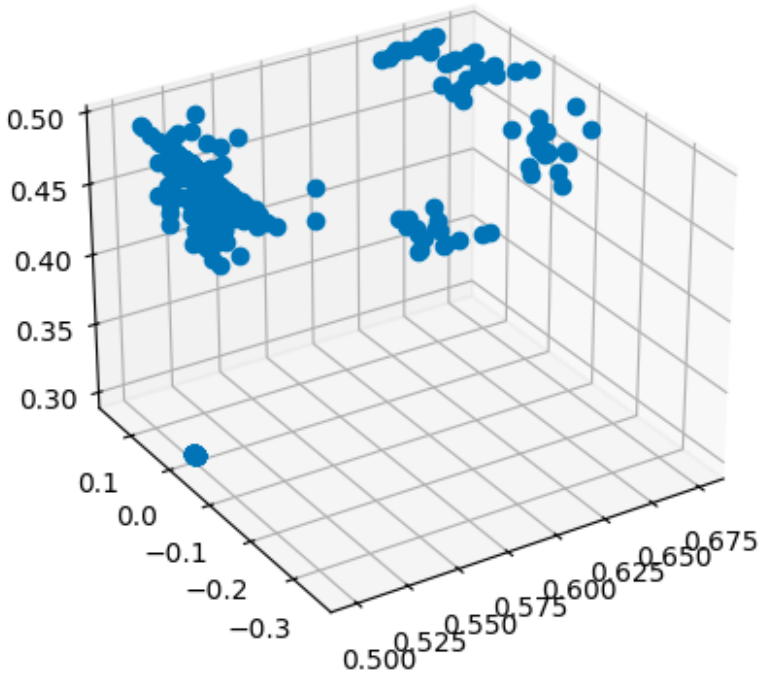


Figure 3: Reinforcement Learning Pointcloud

The pointclouds were exported as a csv with xyz values. The first few lines are summarised in Table 1. It is clear that these points follow the expected values with the x values being close to 0.5, the true location of the front of the cube.

Table 1: First Values of Point Cloud (Rounded for Display Purposes Only)

| x | y | z |
|---|---|---|
| 0.5 | 0.010520 | 0.304395 |
| 0.5 | 0.008560 | 0.310420 |
| 0.5 | 0.030099 | 0.317025 |
| 0.5 | 0.030608 | 0.313234 |
| 0.5 | 0.051414 | 0.305771 |
| 0.5 | -0.168675 | 0.288846 |
| 0.5 | -0.053964 | 0.277278 |
| 0.5 | -0.068076 | 0.273632 |
| 0.5 | -0.083459 | 0.271331 |
| 0.5 | -0.089854 | 0.278497 |

The various motions of the robot can be viewed at this YouTube Playlist: https://www.youtube.com/@alannagurl7760/playlists.

## 5 Future Work

To extend this work, more training should be done so the robot collects points on all sides of an object. Additionally, training should be done with different objects in different positions. After this is completed, it will be possible to classify the objects with the PointNet++ algorithm.

A very important consideration for this project is that the objects in the environment are currently stuck in position. In reality, the object would not be fixed to the ground and thus any shifting of the object would mean some points would be taken when the object was in a specific location and other points would be taken after the object had shifted, thus leaving the pointcloud with a distorted half of the object. To mitigate this problem, the approach of the robot should be very slow and should back up quickly as soon as there was a collision.

In future, since much of the coding was done by programming the robot to move slightly in cartesian directions, the code should be changed to moving in relation to the base of the robot. This is so that if the object is in a different location the traversal will still be effective.

To further avoid unwanted contact, it would be interesting to train the robot to avoid locations where it had already documented the object to be. This would require the robot understanding its own morphology and comparing the intended path to the pointcloud before each motion.

To extend this project to a more advanced application, the future work should be to change the end effector on the robot so that it can grab the object at multiple points simultaneously. This would also mean that it is easier to have a reference of where the points are if there is any shifting of the object during execution.

## 6 Conclusion

This project shows an interesting application of other fields to tactile sensing. Additionally, it show that machine learning can be effectively utilised to circumvent issues with touch sensitivity and navigation of an object. Reinforcement learning is very effective at helping researches collect large amounts of data without costly human supervision and teaching. With some additional work, this project could be very effective at bridging the gap between vision and tactile sensing.

## References


[1] J. Lin, R. Calandra, and S. Levine, "Learning to Identify Object Instances by Touch: Tactile Recognition via Multimodal Matching," arXiv (Cornell University), May 2019, doi: https://doi.org/10.1109/icra.2019.8793885.

[2] M. Bauzá, A. Bronars, and A. Rodríguez, "Tac2Pose: Tactile Object Pose Estimation from the First Touch," arXiv (Cornell University), Apr. 2022, doi: https://doi.org/10.1177/02783649231196925.

[3] G. Izatt, G. J. Mirano, E. H. Adelson, and R. Tedrake, "Tracking objects with point clouds from vision and touch," DSpace@MIT (Massachusetts Institute of Technology), May 2017, doi: https://doi.org/10.1109/icra.2017.7989460.

[4] Y. Xiao, Y. Ma, Q. Huang, and J. Zhang, "Deep Multi-level Feature Learning on Point Sets for 3D Object Recognition," DEStech Transactions on Computer Science and Engineering, no. csse, Jul. 2018, doi: https://doi.org/10.12783/dtcse/csse2018/24500.