DESIGN AND CONTROL OF A NONPREHENSILE IMPULSE MANIPULATOR

Chuizheng Kong

Submitted in Partial Fullfillment of the Requirements
for the Degree of

MASTER OF SCIENCE

Approved by:
Dr. John Wen, Chair
Dr. Agung Julius
Dr. Santiago Paternain

Department of Electrical Engineering
Rensselaer Polytechnic Institute
Troy, New York

[May 2023]
Submitted March 2023

CONTENTS

LIST OF TABLES

LIST OF FIGURES

ACKNOWLEDGMENT

I sincerely appreciate Dr. William Yerazunis (Bill) who accepted me and guided my internship journey at Mitsubishi Electric Research Labs (MERL) in Cambridge, MA. Bill invented the mechanical system of the object-of-interest of this thesis — the nonprehensile impulse manipulator; his creativity and passion inspired my development which resulted in two papers, a patent, and ultimately, this thesis.

I would like to thank my thesis advisor Professor John Wen who accepted me as his graduate student during my hardest time. Prof. Wen introduced me to the world of Robotics through his teaching and gave me the opportunity to apply my knowledge to industrial settings. The camera related works presented in this thesis were a direct application of his teaching. At the same time, I want to thank Glenn Saunders of the Manufacturing and Innovation Center who welcomed me into the robotics lab and provided me genuine advises in my research ideas and career paths.

I want to express my gratitude to my manager at MERL — Dr. Daniel Nikovski who provided me precious knowledge and feedback in machine learning algorithms and computer vision tools. Working with Bill, Daniel also taught me how to conduct research under the pressure of deadlines.

I want to thank my colleagues Burak Aksoy, Alex Elias, Honglu He, and Eric (Cheng-Lung) Lu for giving me programming advises and helping me with robot manipulator related questions.

I want to thank my girlfriend Alina Chaudhri and her loving family for providing me irreplaceable spiritual support.

Lastly, I want to give my most heartful appreciation to my parents Yuantie Kong and Qun Wang, and my second parents Linda Parks and Kevin Walsh for making my studying journey in the US possible for the past 10 years.

ABSTRACT

In modern manufacturing industries, small components such as bolts and nuts of a complex assembly are usually delivered to the plant in big loose batches. To autonomously feed those components into ongoing assembly processes with celerity, vibratory bowl feeders (VBF) were developed in 1950 to perform singulation, orientation, and manipulation tasks. In the past 20 years, however, as robot assembly systems became a prominent part of the new and more versatile manufacturing environment, VBFs appeared to be less suitable as 1) each of them is designed for one specific part only, and 2) the cost to design and tune a new variation is expensive.

This thesis proposes an alternative design of a nonprehensile impulse manipulator with the corresponding control method for singulation, orientation, and manipulation by means of seven fixed-position variable-energy solenoid impulse actuators located beneath a semi-rigid part supporting surface. To supervise the manipulator, a 640p webcam with computer vision tools was included to provide part pose information. To control the device, machine learning algorithms were used to generate a part-specific control policy that bring the part to a user specified target pose.

The device was tested by manipulating a six-faced craps-style die and an imprecise flat square wooden nut from a child's assembly toy. Compared with the benchmark policy, the trained optimal policy was able to flip the die to any desired face with six times higher probabilities and stand the flat nut up on its less stable pose with two times higher probabilities. The device was then put into a collaboration task with a 6-DoF robot manipulator to complete a manipulation task on the six-faced die. The resulted average execution time was faster than most state-of-the-art manipulation tactics.

1. INTRODUCTION

Automated assembly of products makes use of various factory automation devices whose purpose is to put the component parts together in the correct order and position. When typical first-generation industrial robots are used for the actual assembly, they execute the exact same sequence of operations without any variation [2]. The only way this would be successful is if the component parts are presented in the exact same position and orientation, and it is the job of other types of factory automation equipment to make sure that this is the case. A very common and popular such device is the vibratory bowl feeder (VBF) [3] that uses a circular vibratory pattern and a specially designed ramp to bring parts up the ramp in the desired orientation. VBFs are usually about a meter in diameter and half a meter tall. During operations, loose parts are poured into the center in an constant speed. Then, reoriented parts are received at the top of the spiral ramp. Figure 1.1 shows a VBF designed for spool pins manufactured by Saratha Electrical Works, an indian company.

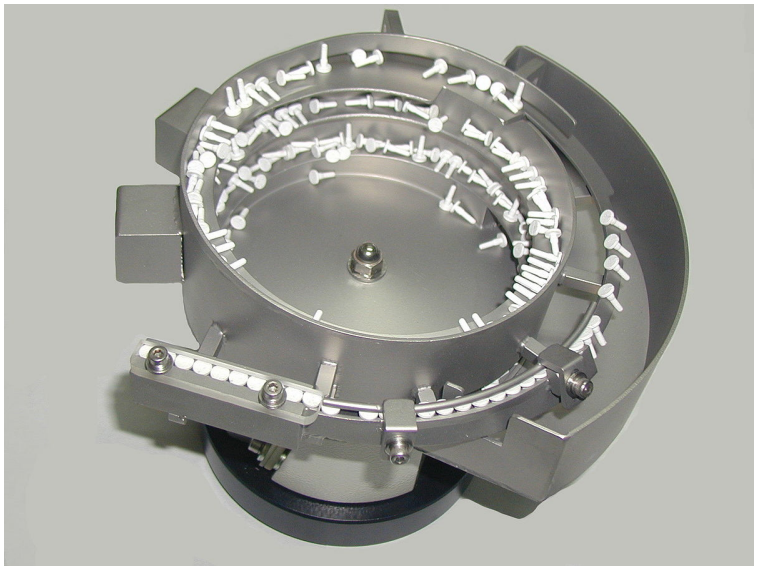


Figure 1.1: VBF orienting spool pins [1]

Such vibratory feeders can singulate and orient hundreds to thousands of parts per minute, a rate very suitable for hard automation assembly machines performing tasks such

Portions of this chapter are to appear as: C. Kong, W. Yerazunis, and D. Nikovski, "Learning Object Manipulation With Under-Actuated Impulse Generator Arrays," *2023 Am. Control Conf. (ACC)*, San Diego, CA, USA

as beverage bottling, writing instrument assembly, or other mass-production operations. Unfortunately, there is a huge "impedance mismatch" between the vibratory bowl feeder and robotic assembly; very few assembly robots are capable of keeping up with the vibratory bowl's very fast task times, on the order of a tenth of a second [4], [5]. Thus, the vibratory bowl feeder's speed advantage is essentially wasted. VBFs are typically noisy, expensive, and difficult to design, due to their size and complexity. With costs reaching hundreds of thousands of dollars and lead times of three to six months, they make economic sense only for very large production runs, and are a poor match to the increasing trend towards high-mix, low-volume manufacturing.

A new generation of industrial robots equipped with cameras has made it possible to grasp parts in a range of orientations [6], as long as they are sufficiently singulated from one another. This has led to the emergence of simplified part feeders [7] where the parts are deposited not in a bowl, but on a flat surface which vibrates in a fixed pattern, eventually singulating at least some of the parts so that they can be grasped by a camera-equipped robot. This solution reduces drastically the noise, size, and cost of the feeder, as the vibration pattern is generic and no custom design is needed for each part.

Still, this solution does not eliminate the problem of having the part often lie on the wrong facet. The robot has some flexibility about how to grasp the part, but at best the robot can approach it from a direction in no more than half of the unit sphere, that is, from above. To deal with this, when the part is facing the wrong way up, the robot would have to pick it up, place it down on a different facet, and regrasp it [8]. There is no generic robot program to do that reliably for an arbitrary part geometry, so a customized program would need to be developed. Moreover, even if such a program were developed, the robot would have to spend time executing it, instead of doing actual assembly, thus increasing the takt time of the assembly operation, which is highly undesirable.

To solve this problem, we propose a novel design for a part feeder (figure 1.2) that uses a set of solenoids mounted under the surface to impart impulse shocks. When parts are placed on the impact surface, impulses can create force and torque at selected locations of the surface that flip the parts to a different facet when the current one is not suitable for grasping. The device is equipped with a camera whose purpose is twofold: first, to recognize which facet the part is lying on, and second, to register the part's position and orientation in order to decide which solenoid to fire in order to maximize the chance of success in changing

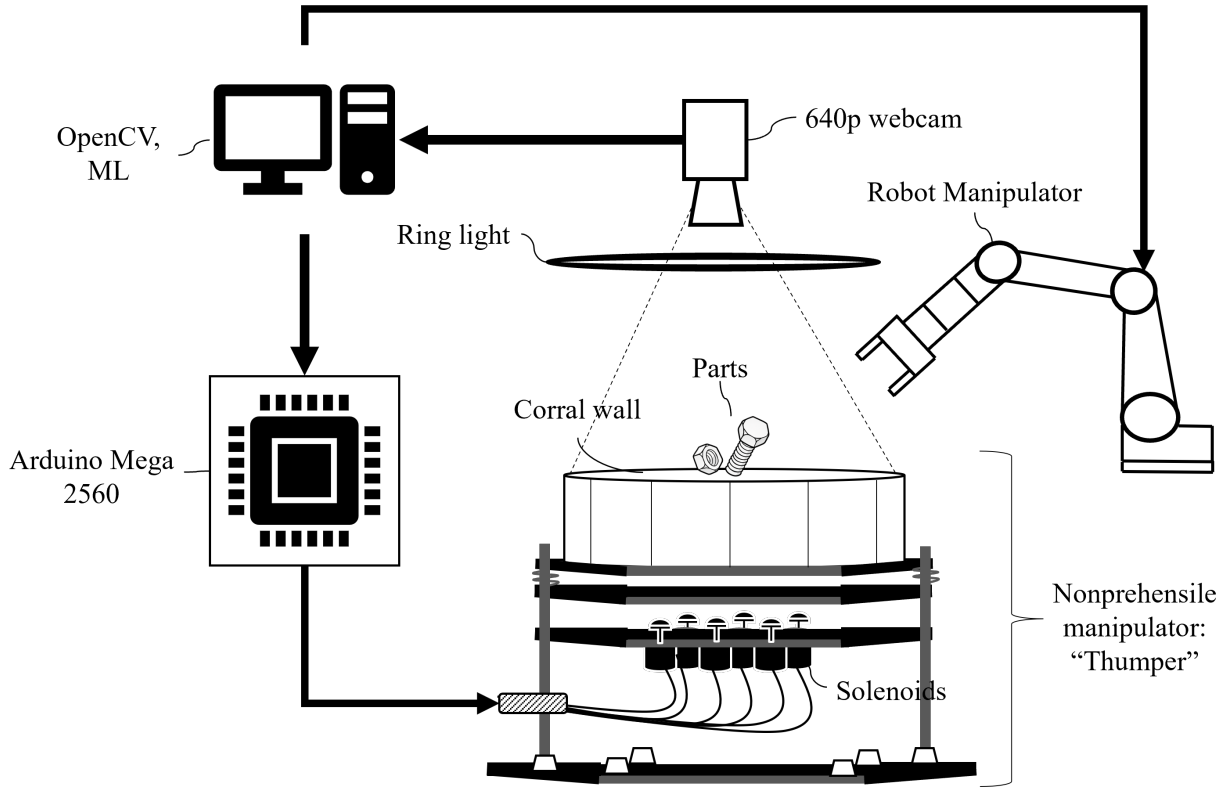


Figure 1.2: The 7-solenoid impact manipulation surface "Thumper"

the facet. Note that this camera could be the same camera that the robot uses for grasping decisions, so it adds no additional cost to the system, while effectively making the part feeder adaptive.

2. DESIGN OF THE MECHANICAL SYSTEM

Impulse-based manipulation has been a rarely visited research area as of 2023. In fact most people choose to avoid impulse as the means to deliver motion due to the complex contact dynamics [9]. Nonlinearities in the mechanically generated impulse make its corresponding manipulation hard to model and control. Therefore instead of using traditional control system modeling approach, we adopted a learning-based method [10] where the probabilistic models of manipulation outcomes were used to decide how to deliver the impulses. To ensure a good learning database, it was important to build a mechanical system that provides consistent impact force with enough possibilities to change the object's facing.

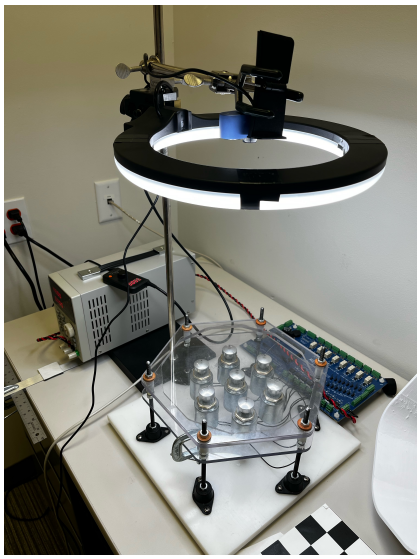


Figure 2.1: Thumper Assembly with Its Bowl Structure Showing

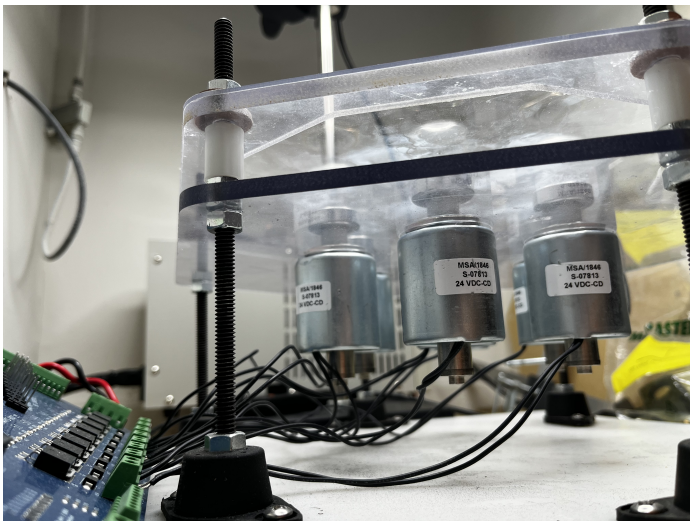


Figure 2.2: Detailed View of the Solenoid Connections

The proposed test apparatus is a seven-solenoid impulse manipulator, standing about 0.15m tall and 0.3m in diameter, as shown in Figure 2.1. The base of the manipulator is a heavy rigid HDPE slab that stabilize the device against unwanted shifts. Six threaded steel

Portions of this chapter are to appear as: C. Kong, W. Yerazunis, and D. Nikovski, "Learning Object Manipulation With Under-Actuated Impulse Generator Arrays," *2023 Am. Control Conf. (ACC)*, San Diego, CA, USA

Portions of this chapter have been submitted to: C. Kong, W. Yerazunis, and D. Nikovski, "Stochastic Control of Object Pose With Under-Actuated Impulse Generator Arrays," *2023 IEEE Int. Conf. Mach. Learn. Appli. (ICMLA)*, Jacksonville, Florida, USA

rods arranged in a hexagon with a 135 mm side length are mounted to the HDPE slab via rubber vibration dampers. These rods support a 1/4 inch thick polycarbonate plate that in turn support the seven upward-firing impulse solenoids[1] arranged as a filled hexagon on 60 mm spacings.

Consists of a cylindrical coil of wire, a solenoid can eject the metal rod at the solenoid core in a straight line when current passes through the wire because of the induced magnetic force. By changing the ON time of the passing current, we were able to deliver a stronger impulse force with longer ON time and weaker impulse force with shorter ON time. These solenoids are rated for 12 volts continuous use, but in order to achieve sufficient kinetic action, each solenoid was over-volted considerably — typically operating at 48 volts. Because the duty cycle is so low (8 to 25 milliseconds of ON time per second, distributed over seven impulse solenoids) the solenoids do not overheat.

As shown in figure 2.2, each of the solenoids carries an aluminium striking head of 25 mm diameter and 10 mm thickness. The convex impacting face of the striking head has a large radius of 250 mm to minimize damage to the actual impact plate (bowl floor). The impact plate is made of ∼1.6 mm thick (1/16 inch) PET[2], a tough yet flexible polymer resistant to impacts.

To assure equal energy on each of the solenoids, a gauging tool was used to adjust each solenoid to the same height at full extension against the internal stop. When in the "rest" (unenergised) position, there is about a 12 mm gap between the striking head and the bottom side of the impact plate, When activated, the solenoid core accelerates upward unhindered across the 12 mm gap and then hammers the flexible impact plate. The solenoid core can flex the impact plate upward by about 1 mm statically, a very small distance, but because of the high impact velocity, enough momentum transfers to the impact plate to couple an adequate impulse to any object in the bowl while avoiding large amounts of deflection[3] in the impact plate.

To actually control the current flow to each individual solenoid, a general-purpose Arduino Mega 2560 drives an array of PowerFET[4] pulldowns. The actual circuit also provides a freewheel diode to prevent inductive spike damage when the solenoid current is turned off.

[1]purchased from McMaster-Carr, p/n 69905K146

[2]PET is also known as polyethylene terephalate, commonly used as the clear plastic used to make soft drink bottles.

[3]By staying below the elastic limit, we can obtain much longer service lifetimes for the impact plate

[4]IRL 1080S n-channel enhancement PowerFETs

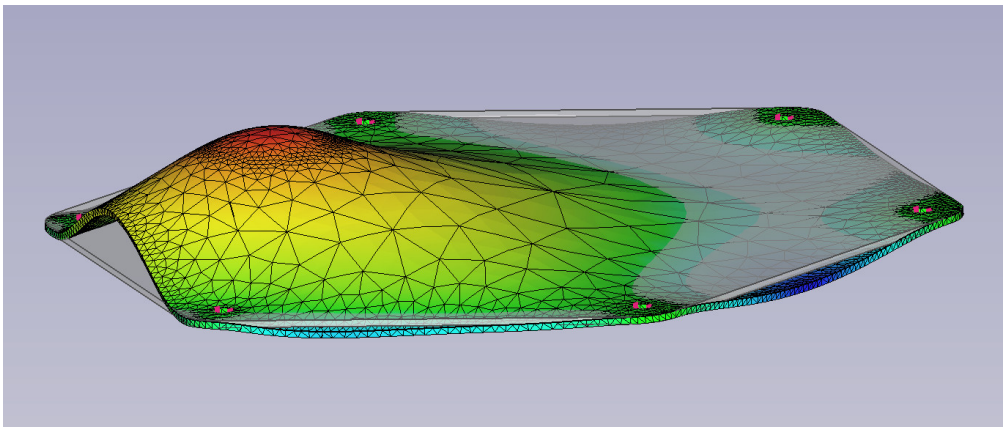


Figure 2.3: FEM of a single solenoid's static effect on the support surface; displacements accentuated 10x

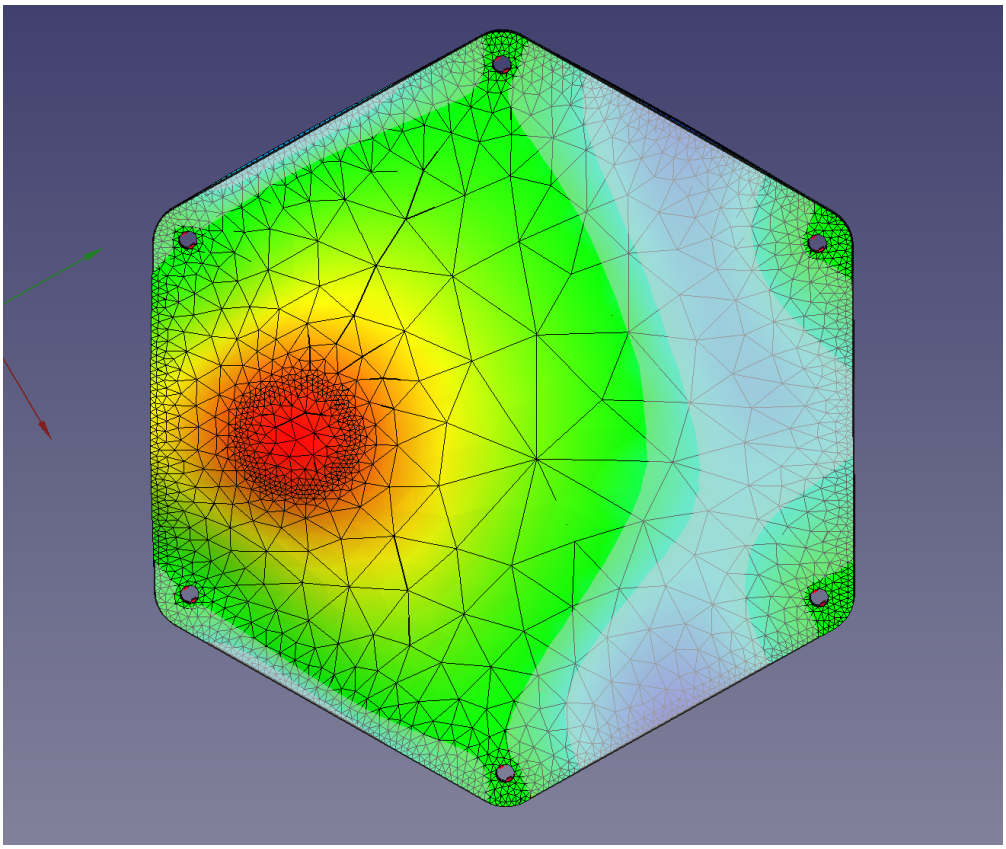


Figure 2.4: FEM Z-displacement top view; blue overlay shows areas where the support surface is moving downward.

Figure 2.3 shows an FEM analysis of the impact plate deflected by the static force of a solenoid. Figure2.4 shows the top view on Z displacement alone. From these figures we noticed that that the location of greatest motion (red) and the location of greatest tilt

(right of red, before the gray) are not the same, nor are they exact inverses of each other. Later results proved that this is actually an useful attribute: To change the pose of a test object, the impact plate must impart both a vertical 'kick' sufficient to get it into the air, and also a rotational gradient 'kick' sufficient to cause a $< 45°$ rotation of the object before ground contact. The region with the greatest gradient provides the best chance to re-orient an object.

To confine the parts onto the impact plate, a white 3D-printed hexagonal "corral" is mounted through the six threaded steel rods (Figure 2.6). The corral is 85 mm tall and tapers from ∼250 mm diameter at the top down to a rounded hexagon ∼180 mm diameter at the impact plate level. The corral is spaced ∼5 mm above the impact plate to allow the impact plate to flex freely upward when an impulse is delivered. To ensure parts recognition in the computer vision (CV) system, a piece of paper with the same color as the corral was added to cover the PET bowl bottom.

Figure 2.5: Thumper in calibration mod

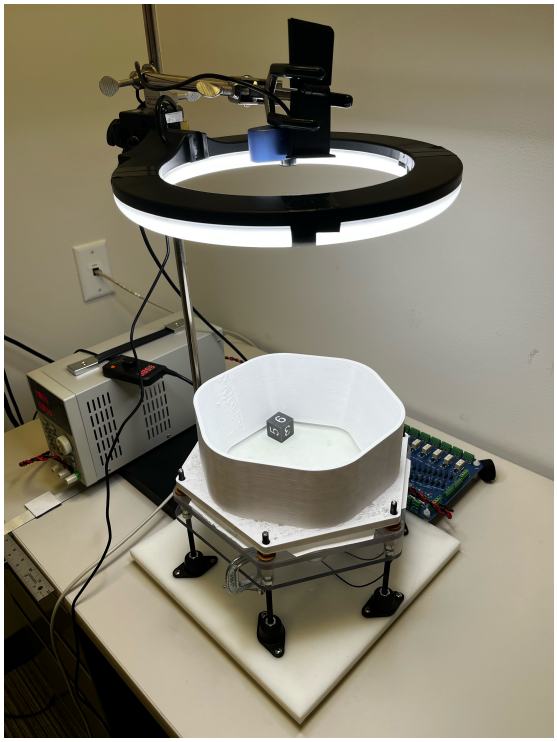


Figure 2.6: Thumper in operation mode with a 6-sided die

About 300 mm above the corral, An 640p HD webcam[5] is mounted on an adjustable

[5]Tripp-Lite AWC-002

metal stand, staring straight down at the center of the bowl. The webcam provides ∼30 frames per second to the CV system that locates the test objects and determines the test object pose. Just below the webcam, a ringlight is mounted on the same metal stand. In addition to providing ample lighting intensity for CV, the ringlight also reduces the influence of shadows from test objects. To calibrate the absolute position of the camera (and determine the camera's intrinsic and extrinsic parameters parameters), we used a checkerboard attached to a 3D-printed mount that positions directly against the vertical rods in a three-point kinematic arrangement, as shown in Fig. 2.5. Details will be discussed in the next section.

For naming convenience, in this thesis we will use the term "Thumper" to indicate the entire apparatus including the host PC running Debian Testing, the Arduino Mega, the PowerFET array, the 48v power supply, the seven solenoids, and the custom bowl, ringlight, and camera assembly. We will also use the term "thumper" with a number to indicate one of the seven sets of the PowerFET, solenoid, and striker heads. The term "solenoid" will refer to the electromagnetic coil, magnetic core, and striker head.

3. OBJECT STATE ACQUISITION USING CV

The key to a reliable control strategy is to have fast and robust sensors. Commercial 640p Webcams have the advantage of cheap cost and comprehensive open source support comparing with some of the higher end products. Instead of sending each frame of the camera into a carefully trained convolution Neural Network to figure out the useful information, we directly computed the state of the object using OpenCV tools. By mounting the webcam directly above the impact plate surface with its lens facing down, It is easy to obtain states (s, x, y, θ) of large enough items such as a die or a wooden square nut with sub-millimeter accuracy.

3.1 Camera Calibration

The camera as the eye of thumper provides state information in the pixel frame. Because the relative position between the camera and Thumper bowl is not fixed, it is important to perform camera calibration before each run. By obtaining the camera specific metricise via camera calibration, One can accurately map points from the pixel frame to the object frame in millimeters. This study employed Zhang's method [11] in OpenCV's Python library.

$$s \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \end{bmatrix} \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix} \tag{3.1}$$

Figure 3.1 shows the (6×8) calibration checkerboard with 30 mm edge checkers. By pasting the checker sheet on top of a flat glass board, we can safely assume that all points on the checkerboard are on the same plane. On this board, as the relative locations of each checker vertices are known and can be found using *cv.findChessboardCorners()*, we calculated the intrinsic parameters of the webcam using *cv.calibrateCamera()*. To ensure all the characteristics of the webcam are captured, we took 31 pictures of the checkerboard at different possible positions of the image with different distances and tilting angles.

Along with the 3-by-3 intrinsic parameter matrix and 5 distortion factors, *cv.calibrateCamera()* also provide extrinsic parameters (R_{chkr}, p_{chkr}) of each of the 31 checker-

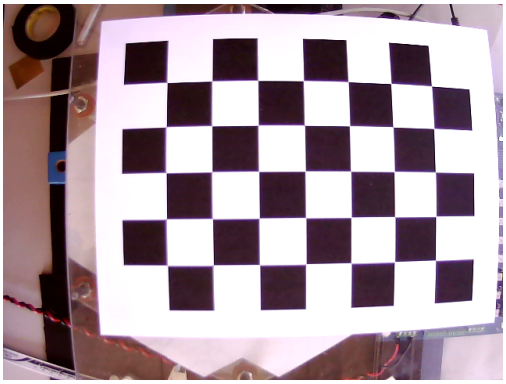


Figure 3.1: Camera intrinsic parameter calibration checkerboard

board pictures. They can be used in *cv.projectPoints()* to project known corner locations from the object frame (checkerboard perspective) to camera frame (pin-hole perspective) and to image frame (pixel locations). By calculating the 2-norm difference between the projected pixel locations of the corners and the actual ones from the picture, we found the projection error of the intrinsic parameter and distortion factors to be 0.0276 pixels. The calibrated result of the intrinsic parameters can be represented in a matrix formate:

$$\mathbf{A} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 678.65 & 0 & 336.67 \\ 0 & 677.29 & 273.54 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.2}$$

Once the nonlinear relationship between the picture frame and the camera frame is calibrated, we want to know camera's location in the Thumper bowl's frame. This is where we needed to find the extrinsic parameters or the frame transformation matrix between Thumper's impact plate and the camera on top. The idea is to have fixed reference points on the Thumper impact plate to be viewed by the camera on top. Specifically, to fully recover the transformation, 12 points are needed to solve for the homogeneous transformation matrix as shown below:

$$\mathbf{T_{CO}} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \end{bmatrix} \tag{3.3}$$

The solution we choose was to 3D print a special checkerboard with a mechanical locking mechanisms added. The mechanism places the checkerboard at the exact position of

the thumper bowl when the thumper is in calibration mode. While there is slight bending of the 3D printed material over time, it is later found that the featured accuracy between the projected and the true object frame points averages from 0.12 to 0.20 millimeters, which make this an acceptable method for extrinsic parameter calibration.

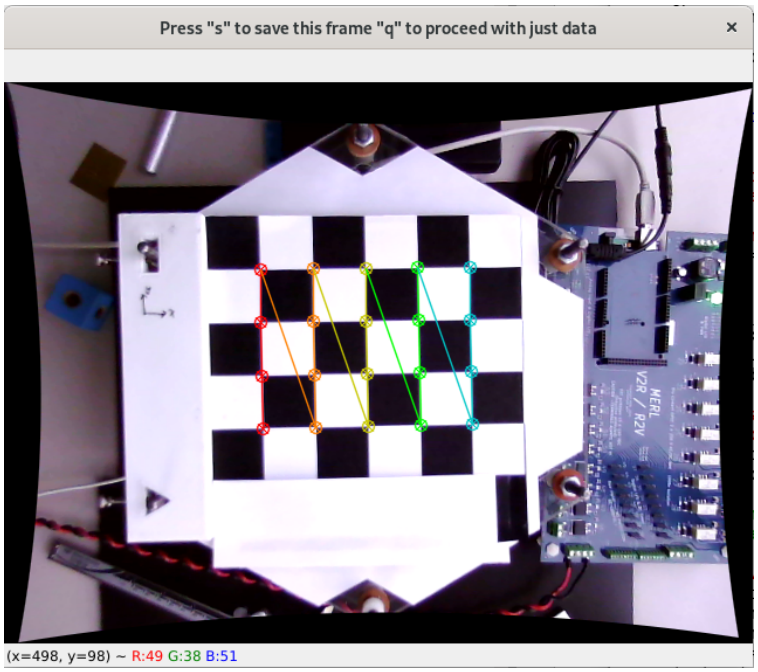


Figure 3.2: Extrinsic parameter calibration board with locking mechanisms

The extrinsic calibration process start by placing the thumper bowl under the overhanging camera and attaching the calibration board as in Figure 3.2. The true coordinates of each checkerboard corner are collected in both millimeters (object frame) and pixels (image frame). With the knowledge of the intrinsic camera parameter of f_x, f_y, c_x, c_y in Equation 3.1 and corresponding distortion factors, it is possible to solve for $\mathbf{T_{CO}}$ (Equation 3.3) using *cv.solvePnP()*.

With the full camera parameters available, we projected the object frame axis as well as the thumper bowl perimeter to the image frame as shown in Figure 3.3. Following the OpenCV convention, x, y, z correspond to $Blue, Green, Red$. We have also set the length of each axis arm to be 60 mm which corresponds to the length of two checker squares. By projecting the thumper bowl perimeter, we can visualize and confirm that the checkerboard corner points have a rich coverage within the thumper bowl's effective region.

So far, we have enabled a one-directional frame transformation (projection) from the

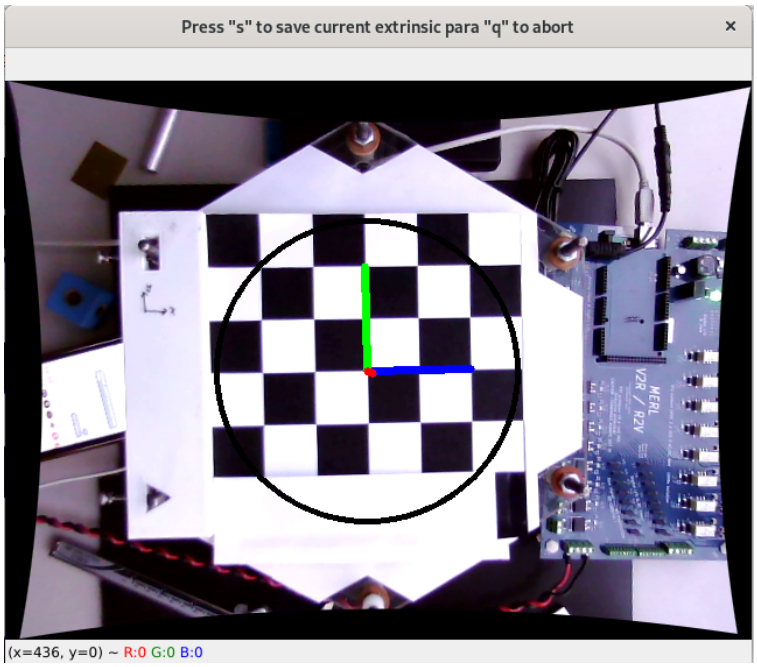


Figure 3.3: Projection of the object frame axis and thumper bowl circumference

world frame to the image frame. The transformation in the opposite direction (3D reconstruction) is normally a harder problem as additional depth information are needed to convert 2d image to 3d world objects; however, as the target object moves only on the thumper impact plate, we can simplify this image frame (pixel) to object frame (millimeter) transformation to a linear transformation by rearranging the terms in Equation 3.1:

$$
\begin{aligned}
s \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} &= \mathbf{A}\mathbf{T}_{\mathbf{CO}} \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix} \\
\begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix} &= \mathbf{T}_{\mathbf{OC}}\mathbf{A}^{-1} s \begin{bmatrix} u \\ v \\ 1 \end{bmatrix}
\end{aligned}
\tag{3.4}
$$

Note that both the intrinsic matrix $\mathbf{A}$ and the extrinsic matrix $\mathbf{T}_{\mathbf{CO}}$ are invertible matrices.

Lastly, to examine the accuracy of the above transformation, we wrote a leave-one-out error calculation function that uses 19 checkerboard corner points to calculate the transformation and test on the left out point. The function then iterated across all 20 points on the checkerboard so that each point has been left out once. The output of the function is shown in Figure 3.4. The error from transformation was on average 0.146 millimeters across the thumper bowl with diameter of 175.88 millimeters. Therefore we could safely utilize the readings from the camera for learning and control purposes.

```
Place excaliboard at 0 degree orientation and press "q"
        collecting extrinsic parameters from the excaliboard
***start excaliboard center correction
Loaded True center of the bowl in pixel:
 [314.566 243.853]
 expected array | leave-one-out iterated coordinates
 [[-60.    -30.    -59.247 -30.079]
 [-60.      0.    -59.282  -0.246]
 [-60.     30.    -59.306  29.833]
 [-60.     60.    -59.507  59.898]
 [-30.    -30.    -29.633 -30.018]
 [-30.      0.    -29.57   -0.214]
 [-30.     30.    -29.56   29.768]
 [-30.     60.    -29.533  59.581]
 [  0.    -30.     -0.012 -29.871]
 [  0.      0.      0.072  -0.121]
 [  0.     30.      0.119  29.636]
 [  0.     60.      0.18   59.502]
 [ 30.    -30.     29.427 -29.848]
 [ 30.      0.     29.519  -0.066]
 [ 30.     30.     29.646  29.641]
 [ 30.     60.     29.625  59.402]
 [ 60.    -30.     58.875 -29.903]
 [ 60.      0.     59.115  -0.008]
 [ 60.     30.     59.121  29.778]
 [ 60.     60.     59.179  59.506]]
****************************
 total reprojection error in (mm):  0.14595053100280245
```

Figure 3.4: Leave-one-out error function output

3.2 Image Preprocessing

Once the camera was capable of capturing accurate position information from Thumper's coordinate system using images, the challenges towards the object state acquisition were to extract the object from the background and identify its center of mass, perimeter, and pose. The former came from image preprocessing, and the latter involved object specific pose identification. During this study, we chose two test objects with different geometric characteristics: a cubic die and a wooden nut as shown in Figure 3.5. (describe the requirment of very high robustness against unpredictability and fast calculation speed)

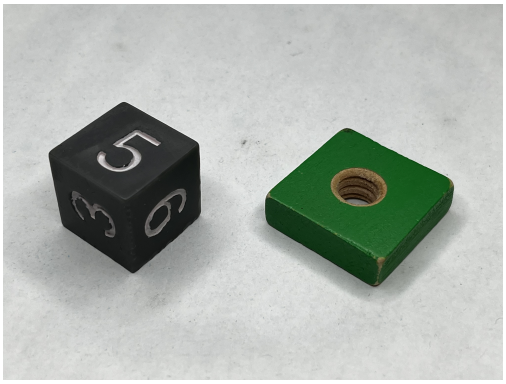


Figure 3.5: Test objects of a cubic die and a wooden nut

While each object had a different pose identification method, they shared the same preprocessing pipeline using functions in OpenCV. The 640p webcam hanging on top provided a 30 fps video stream which was captured into individual frames by *cv.VideoCapture()*. Each captured frame was stored as a 3-channel 640×480 pixel array with integer values between 0 and 255. The frame was then undistorted using the intrinsic camera parameters learned in previous section. The region of interest of the undistorted image frame was sent to an edge extraction pipeline shown in the chart in Figure 3.6.

Bilateral filter is an edge preserved noise reduction filter, and edges provide key information about the perimeters of the object [12]. Instead of using Gaussian blur filter that smoothen an image regardless of neighbor color intensity, bilateral filter allow the noise to be removed while maintaining edges that exhibits a large change in color intensity. a) and b) in Figure 3.7 are images before and after the bilateral filter.

After removing potential noises that can cause false contour, the next step was to simplify the filtered image into a binary image highlighting all the edges found in that image. A naive approach would be setting a threshold where pixels were polarized to either

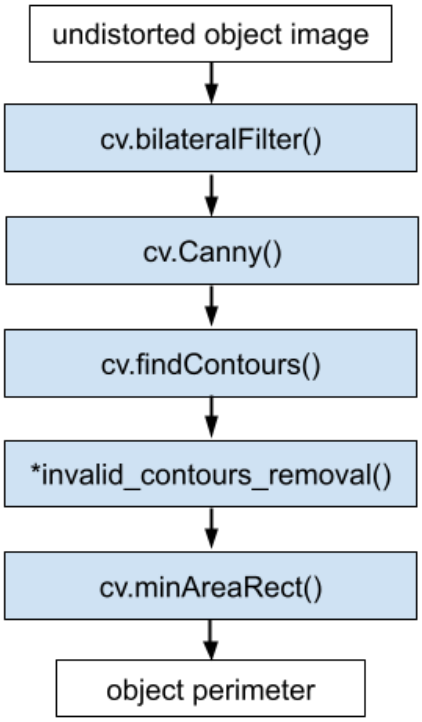


Figure 3.6: Image preprocessing flow chart
**Note: contains sub-functions*

0 or 255 based on their individual comparisons to the threshold. Due to the fixed threshold value, this method can produce inaccurate results due to shadows. Canny Edge Detector is a more adaptive alternative that uses change in color intensity between pixels call intensity gradient [13]. Instead of the using a fixed threshold, user defines an upper and a lower threshold of the intensity gradient of the image. While intensity gradients outside of the upper and lower threshold are directly polarized, anything in between will be treated as an edge (255) if it is connected to a pixel that is above the upper threshold. The effect of the Canny Edge Detector is shown in c) of Figure 3.7.

Binary image above contains all the key pixels that marks the perimeters of the object of interest. OpenCV contains a function called *cv.findContours* that connects these loose points in clockwise and form a contour object. Such an object is supported by powerful tools such as shape matching and shape fitting. In this study, we used *cv.minAreaRect* to find the best fitting rectangle to inscribe the test objects who effect is shown in Figure 3.8.

Under ideal circumstances, the job of image preprocessing would conclude at this point, and the fitted rectangle would be sent to object specific pose identification module. In reality,

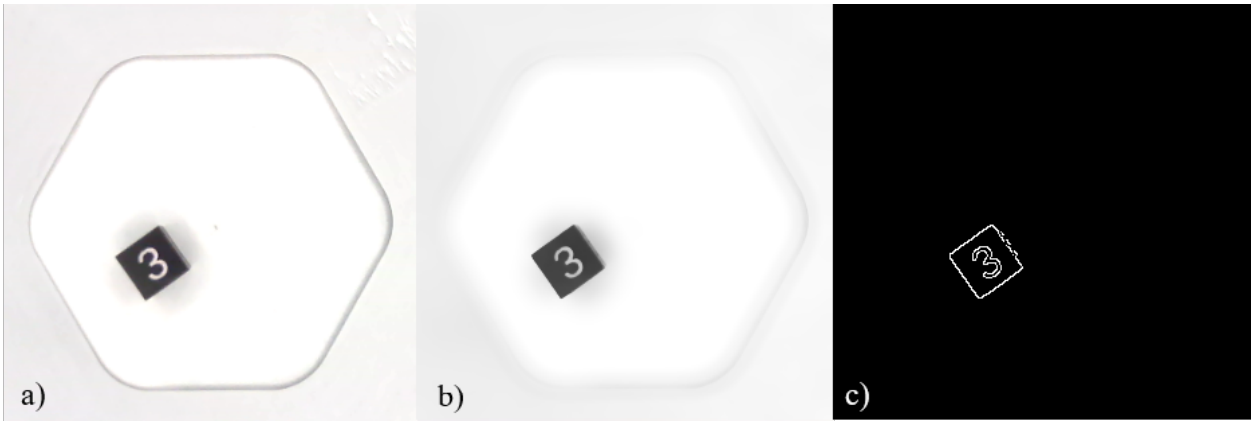


Figure 3.7: a) Undistorted image, b) bilateral filtered image, c) Canny threshold result

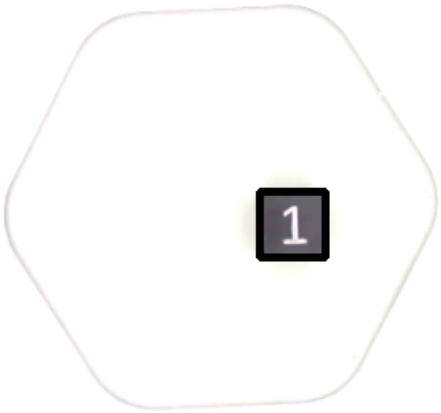


Figure 3.8: Fitting rectangle around the die

after thousands of times moving the test object inside the Thumper bowl, we found almost 20% of all rectangles to be falsely fitted like Figure 3.9. Further debugging revealed that most of the false case were due to the object touching the wall of the bowl, resulting in dark shadows being included as part of the edge. However, as contours created by the shadow are usually not enclosed, we created an additional function that removes unenclosed contours and other invalid contour using the internal hierarchy structure of the contour object. At the same time, we also conditioned the remaining contours using factors such as rectangle area and edge length to determine the validity of the contour. During the most recent run of 65,000 samples, only 50 was misfitted, rendering the fitting accuracy at 99.92%.

Figure 3.9: Falsely fitted rectangle

3.3 Object Specific Pose Identification

The preprocessing pipeline served as a cropping tool that output a sub-image of the object of interest. In addition, the x, y coordinates (pixels) and the rotation θ (degrees) of the fitted rectangle were provided by *cv.minAreaRect* function in the image frame. Using the image frame to object frame transformation described in Equation 3.4, the above mentioned states could all be represented in millimeters in the object frame where the center of the Thumper bowl was the origin. To determine the 3D orientation of the die and the nut, we developed a separate function for each of the test object.

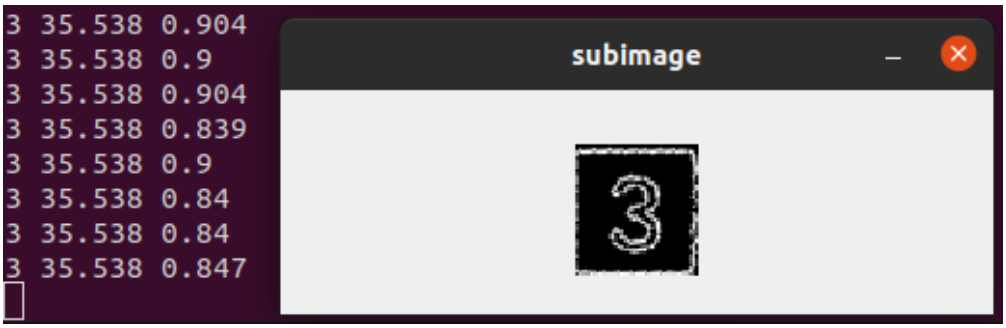


Figure 3.10: Die sub-image (right) with identification results prompt (left) in [*face, oridentation, template_matching_score*]

For the 6-faced die, the most computation efficient way of determining which number faces up was to apply template matching on the cropped sub-image. As shown in Figure 3.10, the location of the number on the die is not perfectly centered in the sub-image. What's more, the perimeter of the die in some of the sub-images are incomplete, which

creates great inconsistencies for template matching. To ensure a high robustness of the die face recognition function, we wrote a cropping function that "eats" away the perimeter of the sub-image. When there's only padding left surrounding the face number, the largest rectangle that inscribes all the pixels of that face number was extracted .

With the cropping tool ready, we created a template library by taking six pictures of the die, one for each face, and the die was placed arbitrarily but close to the center of the thumper bowl. Under each picture, we manually labeled the corresponding number of the face as the file name. During the face recognition process, the sub-image of a die with unknown face would be compared with all templates in the library using *cv.matchTemplate()* with *cv.TM_CCORR_NORMED*, the normalized cross-correlation algorithm [14]. The value of the largest correlation was used as the matching score between the unknown face and templates. The third column in the result prompt of Figure 3.10 shows matching scores of the last 8 frames of the die placed as in Figure 3.7. After testing out with several experiments, we found a threshold of 0.65 that produced the lowest false negative rate of less than 1 in 1000 and nearly zero false positive rate.

The pose identification of the wooden nut was much simpler. There were only two visually identifiable 3D pose for the nut: standing and laying. When the nut was standing, the top view appears as a thin rectangle; when the nut was laying down, the camera saw a square with a small circle inside. A naive approach to distinguish between this two pose was just to compare the ratio between the width and the height of the perimeter. With a threshold of 1.5, the function worked fairly well when the nut was within 60 millimeter radius of the camera optical axis which was near the center of the Thumper bowl. In reality, the object doesn't always stay close to the center. When the nut was standing near the wall, the deviation from the center made the top view from the camera looking like parallelograms of various size. Such an inconsistency was compounded with the shadow the nut casted to the wall, making it hard to determine whether the nut was still standing or undesirably leaning against the wall like in Figure 3.11.

To improve the identification function, we added an additional criteria that used the fitting rectangle area. With the dimensions of the object and the transformation from object frame to image frame in Equation 3.1 both available, we calculated the expected top view areas of the nut in standing pose and in laying pose as number of pixels. Using these values as centers, we set a lower and an upper bound for either pose of the nut. During the experiment,

Figure 3.11: Nut leaning against the wall

if an unknown pose satisfied one of the area bound and the edge ratio threshold, it would be identified as either standing or laying. If it failed either of the criteria, it would be categorize as an unknown pose and await later manual identification. Although such classification achieved a very low false positive rate with the trade off of more human intervention, it was proven to be beneficial when we discovered some unexpected pose of the nut which led to further improvement of the identification function. In a recent experiment using the wooden nut, only about 125 out of 60,000 (0.21%) samples were identified as unknown poses, and in most of those poses, the nut was leaning against the wall. Figure 3.12 shows the result of the identification function.

Figure 3.12: Nut pose identification result

3.4 State Acquisition under Motion

The computer vision tools discussed in previous sections embodied a robust pipeline for single frame object pose identification. During real time experiments, the 640p webcam could collect on average 30 frames per seconds. When analyzing the object pose from a video stream, unfinished movements of the object could create temporal uncertainties towards its state identification. An open-loop approach to the state extraction problem would be to wait for a fix amount of time (i.e. 2 seconds) after each impulse manipulation for the object to land and enter a motionless state before recording. This approach was guaranteed to work, but it was also too slow ($3600\frac{sec}{hr} \times 1/2\frac{sample}{sec} = 1800\frac{sample}{hr}$) in terms of data acquisition. An alternative approach would be adding a motion detection feature using frames from the webcam.

The initial motion detection function compared the difference in individual pixels between the newly collected frame to its previous frame, and if the sum of difference was lower than a certain threshold, the state would be recorded and the next manipulation signal would follow. During real time implementation, we realized that the webcam has an auto exposure system that could not be turned off. This resulted in overall brightness of the frame to change from time to time which made this pixel intensity based method less reliable. Later, we made another attempt by using the canny threshold result (Figure 3.7 c) instead of the original frame image (Figure 3.7 a). The corresponding motion detection function was much more robust since the difference between the new and old binary images were localized to object edges. Equipped with the motion detection function, the data acquisition rate of the CV system reached $3000\frac{sample}{hr}$. During the random manipulation experiment, impulses that created very little motion of the object would be followed by the next impulse immediately, but when the nut was wavering to stand up, the system would wait for the motion to complete.

4. LEARNING TO CONTROL A SIX-FACED DIE

In the past two chapter, we described an Arduino controlled Thumper bowl with seven independent firing solenoids and an OpenCV backed vision system that is capable of identify the state of the objects in the bowl at a frame rate of 30 fps. The former provides under-actuated controls of the object state whereas the latter ensures robust observations of the object state. In this chapter, we will introduce the mind that can utilize the hand and the eye—a learning-based object-specific control policy.

The idea of using a six-face die as the first test object wasn't immediately apparent. Since the purpose of this device lies in industrial assembly tasks, we thought of using nuts and bolts as manipulation targets. There were, however, great difficulties to even get started with the training due to the lack of knowledge in the manipulator's capability and its vast control space. As a result, we decided to start with manipulating a canonical 3D object — a cube, to explore the controllability of the manipulator. Inspired by a casino game called Craps, we added numbers from 1 to 6 onto the cube design to form a die. The die has edge length of 25 mm and weights about 20 gram. To ensure an uniform weight distribution, the part was printed using a Formlabs resin printer with the Grey Resin [6]. Lastly, we used white nail polisher to highlight the shape of the numbers. To use this die as a medium to understand the capability of the Thumper bowl, we designed the following task:

> Die Task: *Given any starting state of the die* $(x_{ini}, y_{ini}, \theta_{ini}, s_{ini})$ *and a desired face number* $s_d \neq s_{ini}$*, create a policy that flips the die to the desired face number with the least amount of firings.*

As mentioned in the beginning of Chapter 2, the optimal control policy for the die will be learned from experiment data. Given any state of an object to be manipulated, the corresponding control policy should provide the optimal command tuple (Thumper#[7], firing duration[8]), or, a list of command tuples that will lead to the desired facet. To evaluate the

Portions of this chapter are to appear as: C. Kong, W. Yerazunis, and D. Nikovski, "Learning Object Manipulation With Under-Actuated Impulse Generator Arrays," *2023 Am. Control Conf. (ACC)*, San Diego, CA, USA

[6] purchased from Formlabs, RS-F2-GPGR-04

[7] Thumper#$\in \{0, 1, 2, 3, 4, 5, 6\}$ as shown in Figure 4.1

[8] firing duration $= [9ms, 20ms] \in \mathbb{Z}$ is the ON time of the current in a solenoid. It is a way to quantify the impact of the kick.

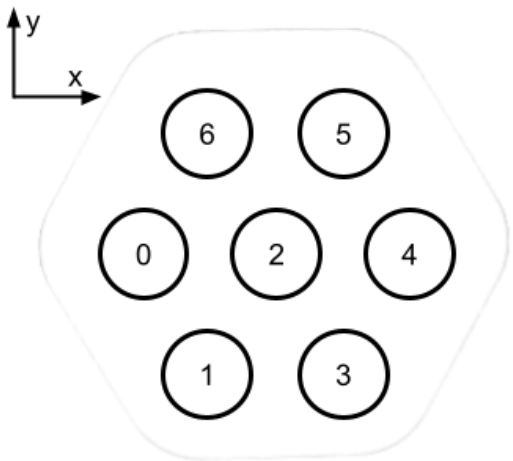


Figure 4.1: Thumper# Layout

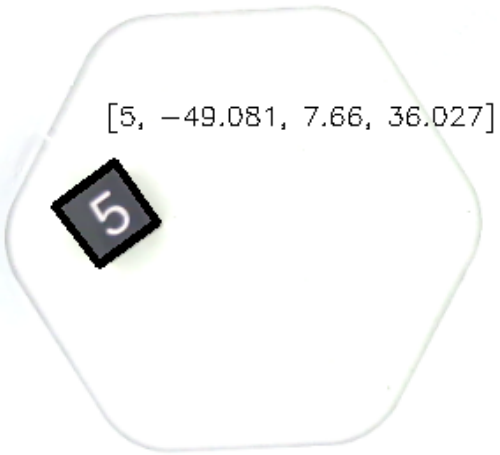


Figure 4.2: Nut pose identification result

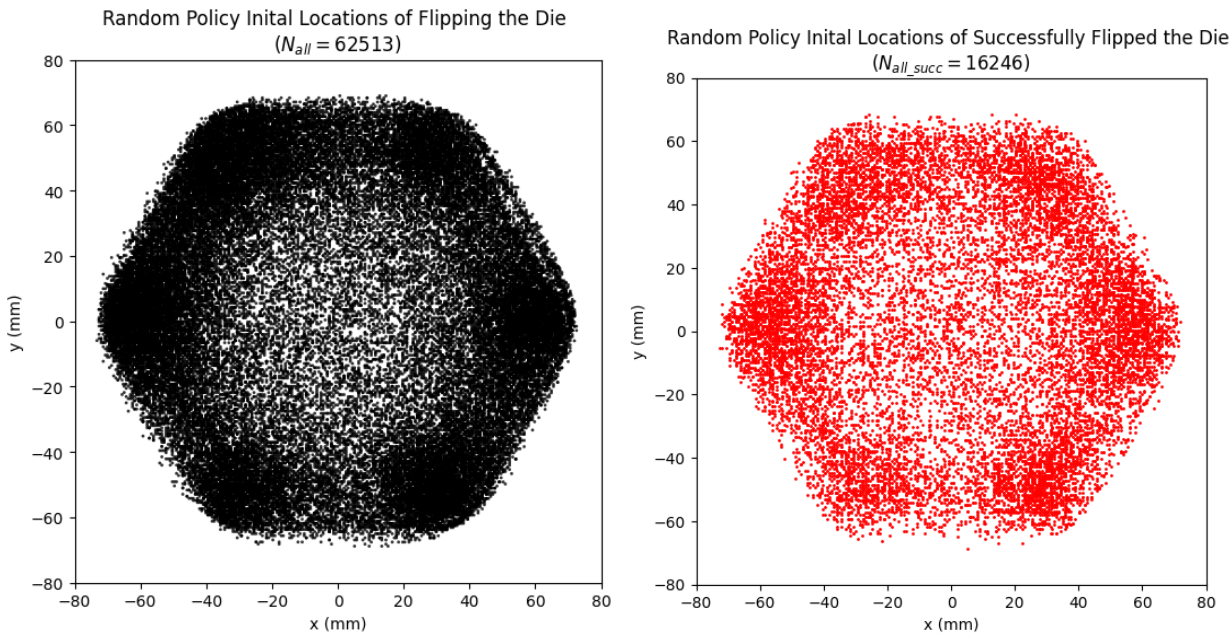
performance of the learned policy, a benchmark random firing policy is used. Such random policy will randomly select a solenoid with a random firing duration regardless of the state of the object; the vision system then log the object state after the firing. This policy is also used to create training data for the candidate ML algorithms.

4.1 Training Data Collection and Overall Thumper Data Analysis

The first and most important step of applying machine learning on this unknown device was to determine the format of features and labels. While the choice of ML algorithm was still unknown, it was crucial to ensure that the recorded data can be generalized to any possible learning algorithms — k-nearest neighbors, support vector machines, deep neuron networks, or even reinforcement learning. As a result, the data are collected in the following format:

Table 4.1: Die Experiment Sample Data Storage Structure

exp id	x_ini	y_ini	θ_ini	status ini	Thumper#	firing duration	x_f	y_f	θ_f	status_f
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
n	x_n	y_n	θ_n	s_n	u_{t_n}	u_{d_n}	x_{n+1}	y_{n+1}	θ_{n+1}	s_{n+1}
$n+1$	x_{n+1}	y_{n+1}	θ_{n+1}	s_{n+1}	$u_{t_{n+1}}$	$u_{d_{n+1}}$	x_{n+2}	y_{n+2}	θ_{n+2}	s_{n+2}



(a) (b)

Figure 4.3: Die Random Policy Full Data (a) vs. Only Succeeded Data (b)

Using this data format, with the above mentioned random policy experiment setup, we ran a 65,000 sample experiment that lasted almost 20 hours. Taking the raw experiment data, two filtering criteria was applied: 1) sample with unrecognizable state ($s_n = -1 || s_{n+1} = -1$), and 2) sample whose coordinate is outside the maximum radius of the bowl (175.88 mm). After the filtering, about 62,500 samples remained. Figure 4.3 (a) shows a scatter plot of all the initial locations of each sample. The data distribution across the impact plate top was not even: the die appeared near the six peripheral solenoid much more frequently than the center solenoid. We tested the center solenoid's impulse by swapping it with one of the peripheral one. It showed no difference, and we had also later observed a tendency for the die to travel toward the side wall of the thumper bowl. This was the same case for flipped or facet rotated samples in Figure 4.3 (b). These samples were a subset of samples in Figure 4.3 (a) that ended on a difference facet comparing with their initial facet ($s_{n+1} \neq s_n$). These samples are considered "successful" because they represented the result from thumper's manipulation to the die. On the other hand, the subset ($s_{n+1} = s_n$) was considered to be "failed".

With the idea of successful samples and failed samples, we decided to inspect how the success rate was influenced by the two controlling factors in the command tuple: thumper# and impulse duration. From Figure 4.4, we see that 1) with the random firing policy, most

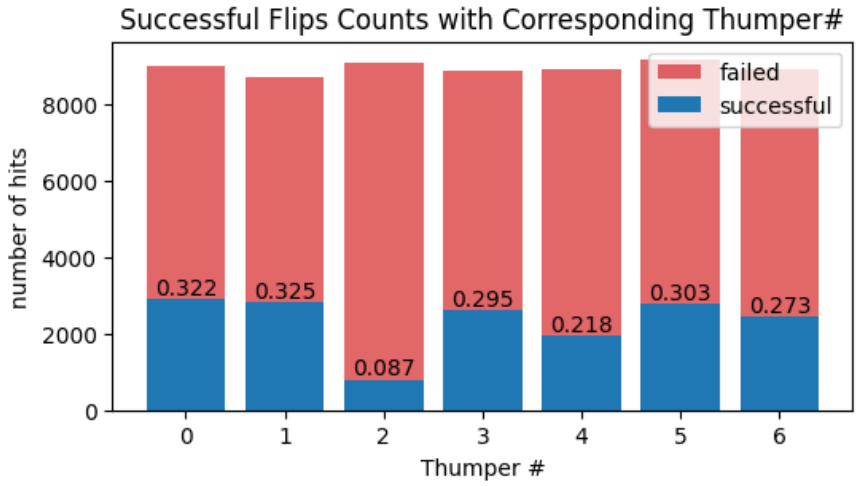


Figure 4.4: Die Random Policy Flip Counts/Rates vs. Thumper#

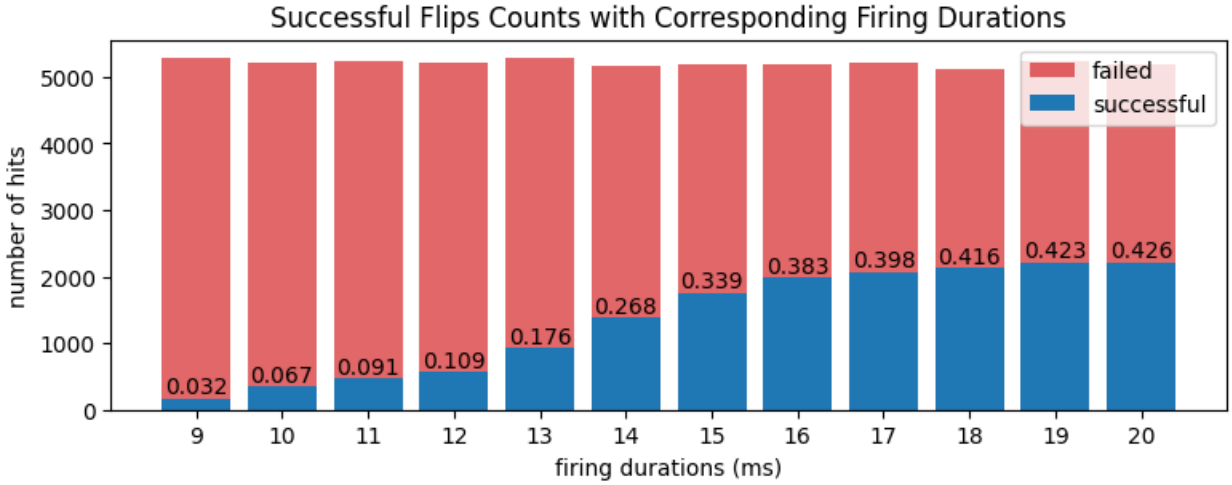


Figure 4.5: Die Random Policy Flip Counts/Rates vs. Firing Durations

solenoids have about 30% chance to change the facet of the die except the center solenoid — Thumper2, and 2) all solenoids had similar total firing numbers at around 9000 samples. The significantly lower successful chance of Thumper2 can only be explained by its location and the complex contact dynamics of the impact surface.

By looking at the number of successful flips in the firing duration's perspective (Figure 4.5), we see a gradually rising chance from 3% to about 43%. The growth seems to plateau after $>$ 18 ms firing duration of the electric current through solenoids. This means that higher firing duration does not further improve the ability to manipulate die's facet. Based on the statistic that describs the overall performance, One might argue that lower

impulses are not necessarily effective in terms of successful manipulation of facet. In the next section, evidence from individual solenoids will show that lower impulse manipulations are indispensable actions for locations closer to the source of impulse.

(a) (b)

Figure 4.6: Die Random Policy Thumper0 Data (a) vs. Thumper0 Succeeded Data (b)

4.2 Individual Solenoid Data Analysis

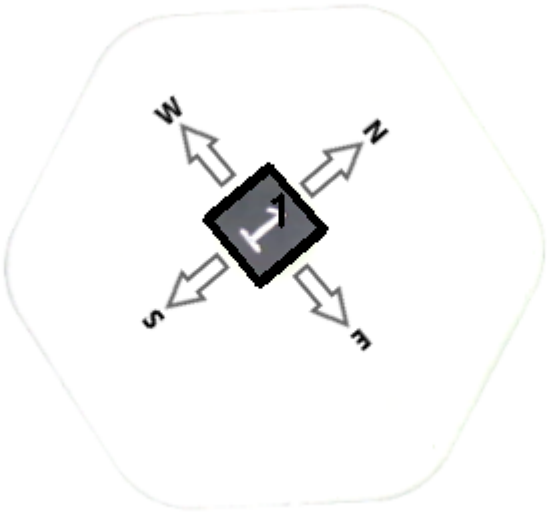
By looking at the result from just one solenoid — Thumper0, something more interesting about thumper's mechanism emerges. First of all, by comparing the data between Figure 4.6 (a) and Figure 4.3 (a), we were ascertain that samples drawn for Thumper0 was quite representative of the true distribution since the smaller distribution from just one solenoid resembles the shape of the overall data. Figure 4.6 (b) shows locations of the die that had successful flip due to Thumper0. The scatter plot provides the following observations:

1. a tight cluster at around $(-60, 0)$
2. two group of loose cluster gathering at the top and bottom right, almost symmetrical to the x-axis
3. a clearance area to the right of the tight cluster and another one between the two loose cluster

Observation 1) isn't so surprising since Thumper0 is located right below the coordinate $(-60, 0)$ according to the schematics in Figure 4.1. The following two observations, however, convey important messages about the controllability of each solenoid. The successful flip plot of all other solenoids can be found in the appendix Figure A.1. Except the center solenoid, all six peripheral solenoids share almost identical distributions following the observations above but rotated following their positions with respect to the center of the thumper bowl. We drew a hypothesis based on the FEM analysis plot in Figure 2.3: Locations of successful flip should match the corresponding region of the steepest slope in the FEM analysis. Based on this hypothesis, we further surmised that the direction of the flip should point towards the normal vector of slope surface.

Starting face \ Landing face	1	2	3	4	5	6
1	x	E	N	S	W	+
2	W	x	N	S	+	E
3	S	E	x	+	W	N
4	N	E	+	x	W	S
5	E	+	N	S	x	W
6	+	W	N	S	E	x

(a) *



(b)

Figure 4.7: Die Facet Rotation Table (a) and Schematic (b)

Note that "+" means the bottom face, and "×" means no change in facet.

To examine the hypothesis, we put together a technique to represent the flip direction of the die at each location of Figure 4.6. This technique uses three features as described in Table 4.1: the initial facet s_n, orientation θ_n, and the final facet s_{n+1}. The idea is that the facet rotation of the die is interchangeable. We summarized a facet rotation table based on the number configuration of the die in Figure 4.7 (a). Suppose the die has a starting face# 1 as shown in Figure 4.7 (b), by flipping north "N" (body frame), the die rotates around a

vector that is collinear with "W" for 90 degrees, following the right hand rule. According to the rotation table, it will land on face# 3. Therefore, this flip direction will be represented as an arrow rooted at the center of the initial location of the die, and points towards the direction of "N." The length of this arrow will just be an unit length.

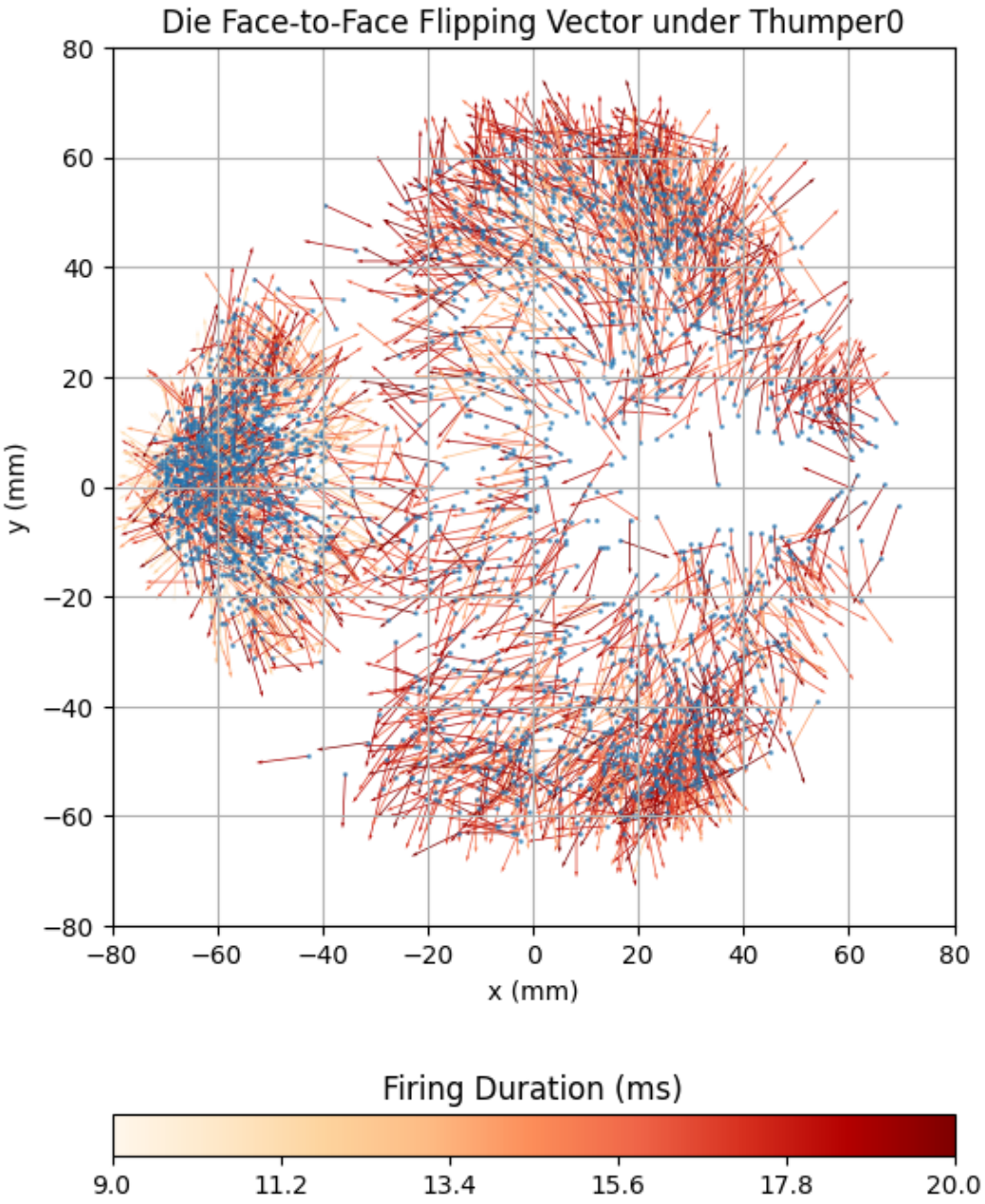


Figure 4.8: Thumper0 Die Flipping Direction Map

Using this technique, we plotted the flip direction of the die from firing Thumper0 in Figure 4.8. This quiver plot conveys a great amount of information regarding the con-

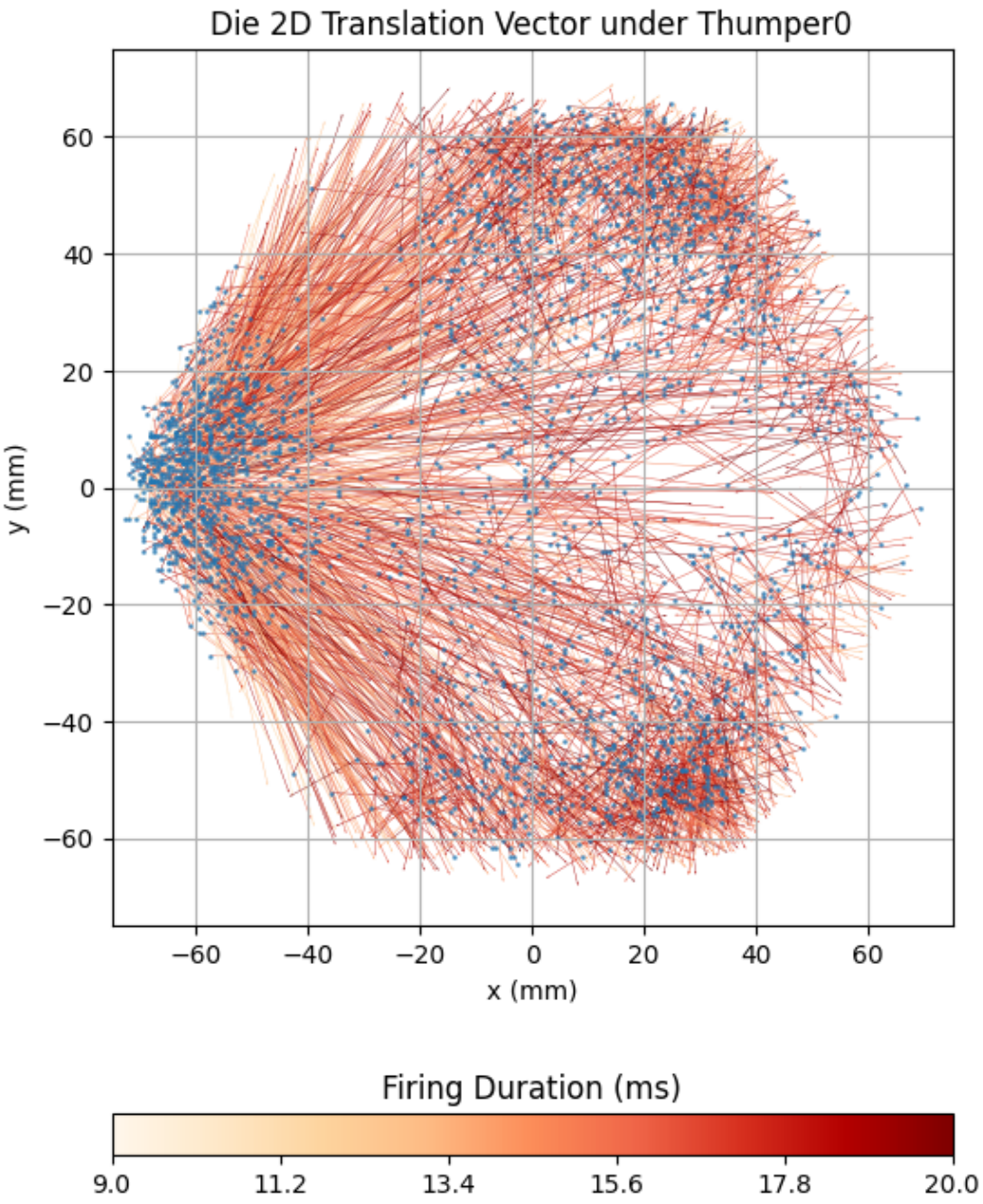


Figure 4.9: Thumper0 Die 2D Translation Map

trollability of Thumper0. The dense cluster or "island" described in observation 1) indeed produces outward pointing flips; however, the flipping directions of the two looser cluster have majority of their arrows pointing towards the $+y/-y$ direction. As the location gets closer to the island, the direction of arrows turn more and more towards the island. The two clearance areas as mentioned in observation 2) serve as separation lines where the die flips on opposite directions on each side. On top of this, the impact of each firing has also

been color coded based on the "Impulse Duration" bar. When the die is close to the impact source (i.e. $(-60, 0)$), lighter impact have higher success rate to flip the die. The two further cluster regions are dominated with stronger impulses where arrows' color get darker. As a result, this flip direction plot visualized a ballpark control space (the island, two clusters) and the uncontrollable region (two clearances) of Thumper0. With all seven solenoids working together, the uncontrollable region of a single solenoids might be in the control space of other solenoids.

In addition to the flip direction plot, we extended the idea of quiver arrows and created a 2D Transition map that shows the initial (x_n, y_n) and final location (x_{n+1}, y_{n+1}) of each successfully flipped die using Thumper0. As shown in Figure 4.9, distance and direction of the translation are represented in terms of the length and direction of the quiver arrows. Colors of the arrow inherent the same meaning as in Figure 4.8 — firing duration which correspond to the intensity of the firing solenoid. The "island" on the left side project the die away to the right half of the thumper bowl. Meanwhile, the $+y/-y$ clusters result in shorter trajectories. Lots of the arrows end at around the boundary of the bowl. It is still unclear if the thumper wall also plays a role in manipulating the die. In addition to the translation patterns, this plot also provides an insight into chained control policies where trajectories of the die can be predicted and used to make n-horizon policies. This topic will be discussed with details in the future work section.

Back to the hypothesis mentioned earlier, these two plots (Figure 4.8 and 4.9 have shown enough evidence to reject the first postulation drawn from the FEM figure. It seems that explanations behind the distribution of successful flips rely on more than just static FEM analysis. Inspired by the flip direction plots of the center solenoid (Thumper2) in Appendix A.2 (c) and A.3, a possible future direction could be using a dynamic FEM that shows the shock wave propagation, and exploring how the wave interact with the six fixed constraints imposed by the clamping nuts on the edge of the thumper bowl.

4.3 Manipulation Repeatability Analysis

When discussing the hardware design requirements in the beginning of Chapter 2, we mentioned that the Thumper Bowl was designed to meet two criterion: 1) possibility to manipulate the facet of parts, and 2) consistency in performing such manipulation. Section 4.2 has shown the possibilities and different ways for Thumper to flip the die using one of

the solenoids in various firing durations. Before deciding which machine learning algorithm to use, it is important to confirm that the successful manipulations recorded in the 62,500-sample database can be reproduced.

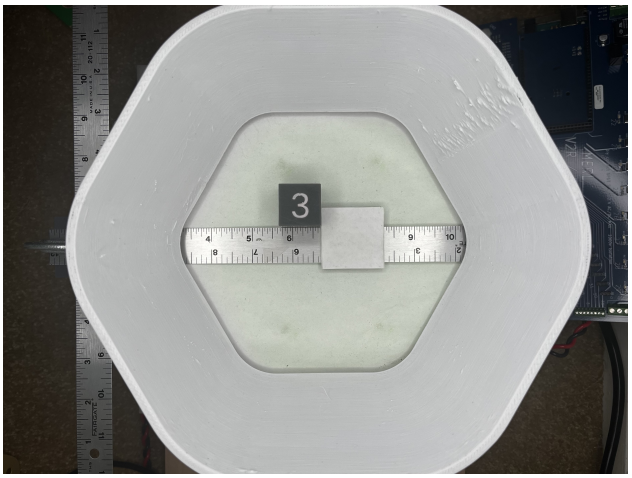
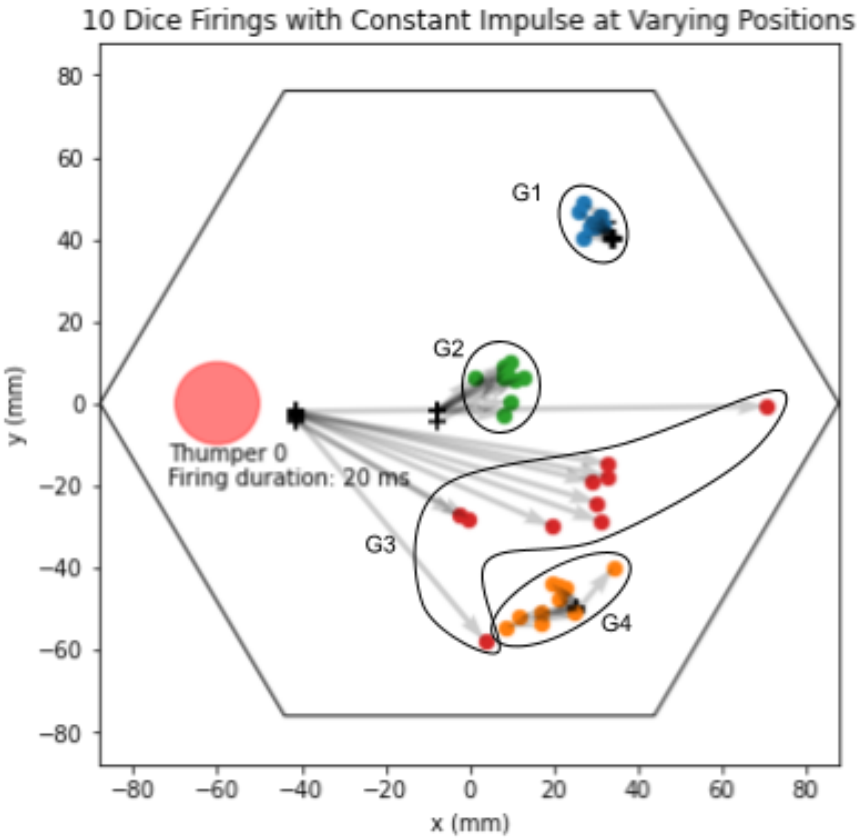


Figure 4.10: Repeatability Testing Jig and a Die in the Thumper Bowl

To assess the manipulation repeatability, we made a kinematic jig to reproducibly place the die at several "Successful" locations in the bowl with number '3' facing up and oriented at zero degrees as shown in Figure 4.10. We then positioned the die to each location 10 times with sub-millimeter precision (as verified by the CV system).

The result is shown in Figure 4.11. Figure 4.11 (a) shows the xy location of the die during four experiments with varying jig-set starting positions and fixed impulse location and duration. The starting locations of the four experiment groups are marked with black "+" signs as G1, G2, G3, and G4. Figure 4.11 (b) shows the detailed firing input and the resulting facets. Four observations are made from this Figure:

1. Location G1 (shown in blue) showed that the system was highly reproducible, with little variance (less than 10 mm) in xy location and 100% transitioning from face "3" to face "6".
2. Location G2 (in green) was also highly reproducible (about 15 mm), and a 100% chance of transitioning from face "3" to face "5".
3. Location G4 (in orange) showed a mild spread (about 20 mm) but a bifurcating face transition (seven to face "1", three to face "5").
4. Location G3 had the highest spread (about 70 mm) and a chaotic result in face transition (six samples rotated to face "1", two to face "2", and one sample each to face



(a)

	starting face	thumper fired	firing duration(ms)	landing face of ten tries
G1	3	0	20	[6, 6, 6, 6, 6, 6, 6, 6, 6, 6]
G2	3	0	20	[5, 5, 5, 5, 5, 5, 5, 5, 5, 5]
G3	3	0	20	[1, 1, 1, 1, 1, 1, 2, 2, 3, 5]
G4	3	0	20	[1, 1, 1, 1, 1, 1, 1, 5, 5, 5]

(b)

Figure 4.11: Thumper Manipulation Repeatability Experiment Results

"3" and "5")

The result from Figure 4.11 shows that it is possible to reliably control the landing face of the die with the correct control input, at least from some regions of the bowl. For some positions the transition is well defined and the final location after an impulse is contained to a small area (such as G1, G2 and hence highly predictable), and for other positions (G3, G4)

the final location after impulse is essentially chaotic. All of the samples in these figures used the same impulse solenoid 0, and always fired for 20 milliseconds. The observations from Figure 4.11 (a) and (b) pointed out a possible relationship between the spread of landing positions and the uncertainties in landing faces.

On top of what we directly learned from those observations, this experiment has also hinted that with any pair of the command tuple (Thumper#, firing duration), there is an inherent probability map of the flipping manipulation that can be called a "reliability map". This map can tell me, for example, that with (Thumper0, 20ms), the probability of getting repeated manipulation result at G1, G2 is nearly 100%. While it almost seemed like we have found a new quantity to be learned to enhance the performance of Thumper, the amount of sample needed to achieve such a data density—10 samples at the same exact state—is unimaginable.

Assume that we can accept a tolerance of a $5 \times 5 = 25$ mm^2 square in the xy feature space, 10 degrees in the θ feature space. With the bowl's impact surface area estimated to be $A = \frac{1}{4}175^2\pi \approx 24052$ mm^2, about $A/25 \approx 962$ distinct states of die's location can be represented on the surface. Since the die has 90 degrees rotational symmetry, a total of $90/10 = 9$ states are needed for the angle. Since the transition of numbers on the die is interchangeable based on the facet rotation table in Figure 4.7, only one state is needed. The total amount of states of the die observed on the impact plate would be $n_{state} = 962 \times 9 = 8658$. This is the number of states for each command tuple pair. The total amount of states across all command tuple (Thumper#, firing duration) is $N_{state} = 8658 \times 7 \times 11 = 666666$. Now if we want to have 10 samples in each states, we would need 6666660 samples. As mentioned in Section 3.4, the average running speed of Thumper is 3000 samples / hr. This means that 2222 hr or 92 days are needed to develop such sample size.

The hint from the observations and the calculation above showed that "reliability" as a desirable quantity is hard to fully realize, and the performance of the about-to-be-discussed machine learning algorithms can be compromised as the size of the current training data is only 1/100 of the desired sample size.

4.4 Algorithm Selections and Control Results

Up to this point of the chapter, we have proposed a control task for Thumper, set up an experiment to collect training data, and provided some insights about the capability

of Thumper for manipulating the die. All these information will now congregate and aid the process of finding a suitable learning control method for this problem setting. The learning object involves deriving a control policy $\mathbf{u} = [u_t, u_d]^\intercal = \pi(s, x, y, \theta)$ that selects which solenoid u_t to fire with duration u_d so as to maximize the probability of moving the object into a desired state. This desired state can be described in terms of one or more of the state components, for example changing the face the part is lying on, or also possibly bringing it to a desired position and orientation.

Let the Boolean function $g(s, x, y, \theta)$, provided by the user, indicate whether state (s, x, y, θ) is a desired goal state or not. A desired ML algorithm should learn probabilistic models $p = h(s, x, y, \theta, \mathbf{u}) = Pr[g(s', x', y', \theta') = True|s, x, y, \theta; \mathbf{u}]$ that predict the probability p of bringing the part into a desired configuration by firing the command tuple $\mathbf{u}$ when the part is in configuration (s, x, y, θ). Here, (s', x', y', θ') is the successor state after firing a solenoid according to $\mathbf{u}$. With a sufficiently accurate predictive model, It is possible to develop a greedy control policy by choosing the solenoid (and maybe duration) $\mathbf{u}^*$ that maximizes the probability of success: $\mathbf{u}^* = \arg\max_{\mathbf{u}} h(s, x, y, \theta, \mathbf{u})$. For a multi-step decision policy, it might also be advantageous to explicitly learn a model to predict this state, of the form $(s', x', y', \theta') = f(s, x, y, \theta, \mathbf{u})$ [10], [15].

This impulse-based manipulation of the die was first explored as a command domain question — what set of solenoid / impulse pair were actually useful in rolling the die. As discussed in Section 4.2, there are controllable and uncontrollable states for each sets of command tuple $\mathbf{u} = [u_t, u_d]$. Experiment in Section 4.3 has shown that the repeatability of the outcome changes drastically with the location of the die. While it is hard to make use of the knowledge in repeatability with the training data size, we designed a sub-task to first testify if we can improve the chance of effectively flipping the die with the already collected data.

4.4.1 Sub Task: Flipping the Die to any Other Face

In this sub-task, the goal was to roll the die to any face other than the one it was currently on, easily recognizable by the vision system as a change of the number on the topmost face. Accomplishing this task in a minimal number of attempts is equivalent to maximizing the probability of success in one attempt. When starting in state (s, x, y, θ) and ending up in state (s', x', y', θ'), the success criterion is $g(s', x', y', \theta') = True$ iff $s' \neq s$.

Given the face transition results as seen in Figure 4.8, it is clear that the system response is not linearly separable, with chaotic areas and significant dead zones. In addition, the choice of which solenoid to fire is clearly a categorical choice, the duration of impulse on each solenoid is continuous (at least as viewed on a millisecond scale). This requires a control policy that can yield simultaneously a multi-class classification (Thumper#) and a regression-style continuous-valued result (firing duration). Instead of attempting nonlinear model-based learning algorithms involving optimizing multiple hyper-parameters simultaneously, we chose to consider memory-based approaches.

Several informal tests were done with a k-nearest neighbors (kNN) classifier [16] with k varying from 1 to 24, but the results were not encouraging —the associated areas under the receiver-operator characteristic (AUROC) curves[9] were on the order of 0.7 at best [17]. Figure 4.3 and 4.6 both showed that the sample data set was strongly biased toward having the die near the edge of the corral, probably due to the die hitting the corral wall and losing energy in the partially inelastic collision. This effect (akin to thermally induced density gradients in a gas) caused significant depletion of the sample population in the bowl center. In these low density regions, the kNN diameter was expanding to 10-20 mm. As we found in further testing using a kinematic jig to reproducibly place the die in a controlled location, the regions of the bowl where movements were correlated and consistent are often smaller than 10mm (Section 4.3). If those regions happened to be low density as well, then the effective area of the kNN would become much larger than the correlation area and the kNN policy could behave no better than random chance.

We found significantly better results with a radius neighborhood (rN) classifier [18]. The rN classifier included all points within a given radius r in the voting set rather than just the k nearest points as in a kNN; voting and final selection of which solenoid to fire proceeded similarly to the kNN and yielded the categorical output choosing which solenoid to fire. The firing duration was then chosen to be the mean of the set of successful activation impulses on that solenoid. Since we had access to the number of neighbours of each sample, we could keep track of when a sample was having too few neighbors or no neighbors at all and took action towards it.

Like kNN, rN relies on some distance metric to determine whether a sample (x, y, θ)

[9]The ROC curve describes the performance of a binary classification model for multiple thresholds at the same time. The AUROC curve ranges from 0.5 (random guessing) to 1 (correct on all test set).

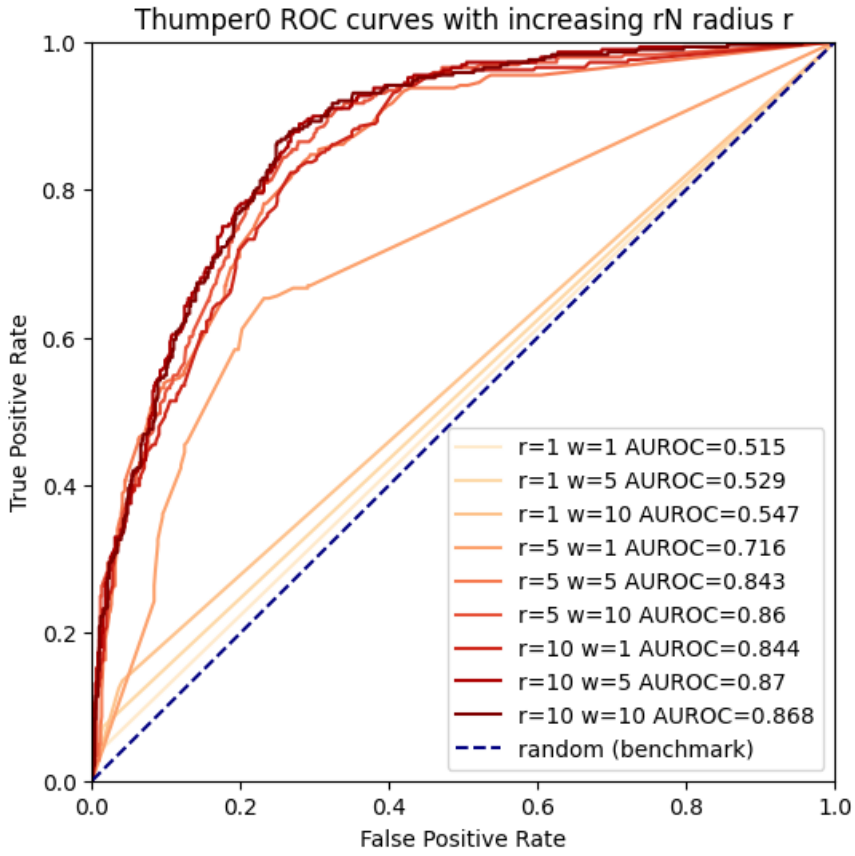


Figure 4.12: ROC curves for determining r (radius of neighborhood) of the rN classifier

is close enough to a prior observation (x_0, y_0, θ_0). As θ inherited a different unit than x and y, we made use of a single distance metric that scalarizes the two distances in position and angle as follows:

$$D[(x_0, y_0, \theta_0), (x, y, \theta)] = \|(x_0 - x, y_0 - y)\|_2 + w|\theta_0 - \theta|$$

where w is a tuned conversion factor.

To determine an effective radius r and a suitable conversion factor w for the rN classifier, we used 10-fold cross-validation on each of the seven rN classifiers. For each solenoid and its underlying classifier, we took test data from the train-test split and swept a threshold a from 1.0 to 0.0. E.g., when $a = 0.7$, for any query in the test split, 70% of its neighbors (from train split) within r have to meet the success criteria of face different, $g(s', x', y', \theta') = True$ iff $s' \neq s$, for that query to be predicted as successful. Predictions of all queries were then

compared with the ground truth label, and an entry of true / false positive rate (TPR / FPR) was plotted. The resulting ROC curves of one of the classifiers is shown in Figure 4.12. While increasing the radius of the neighborhood improves performance by including more data, radius beyond 5 mm yields little if any improvement. The value $r = 5$ with $w = 5$ was commonly agreed by all seven classifiers from the corresponding AUROC curve value.

We then performed an rN multi-class classification using these parameter settings and the 62,500 training samples (about 9,000 samples per classifier). Given an arbitrary state of the die (s, x, y, θ), the controller will apply all 7 classifiers on this state and then fire the solenoid whose corresponding classifier provides the best probability to roll the die to a different face. 2500 samples were collected in this test experiment. The result of this task is shown in Figure 4.13 with an average single-shot success probability of 0.753, versus 0.260 for the random policy — an improvement of almost three times.

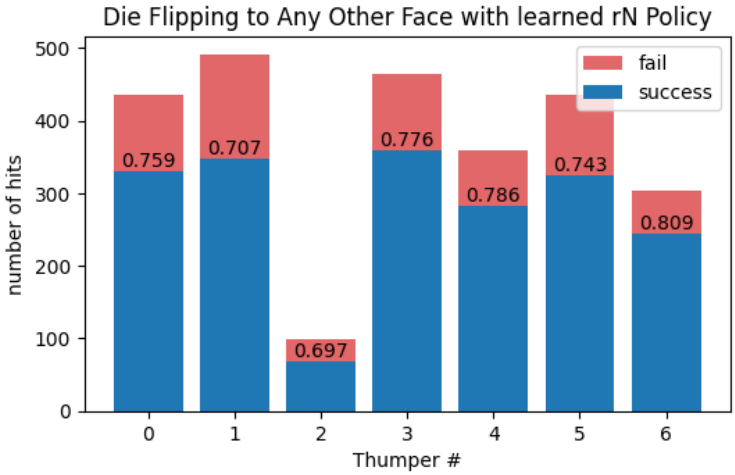


Figure 4.13: Die face changing (success) counts and rates on each solenoid using the rN classifier

4.4.2 Main Task: Flipping the Die to a Specific Target Face

The next more complicated control problem was to learn how to roll the die to a chosen face different from the one it was currently on, so direction of rolling became significant.

The task here was to achieve a series of 2,000 randomly chosen target values for the upper die face (with no sequential repeated faces) allowing up to 10 impulses to achieve the desired die pose. This emulates the challenge of feeding properly oriented parts to a manufacturing robot.

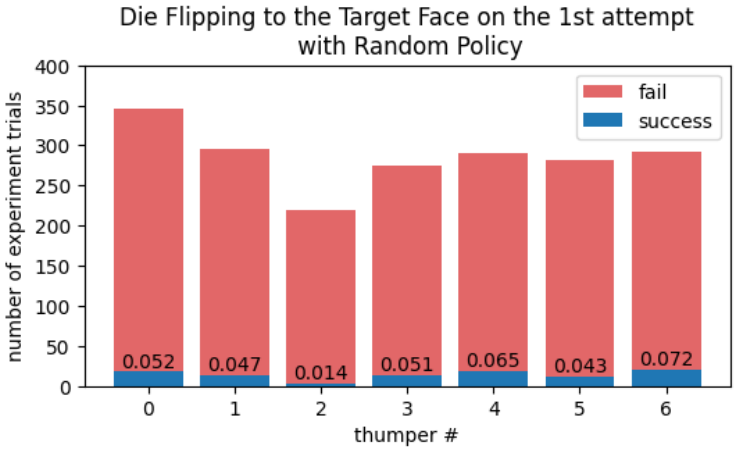


Figure 4.14: Number of failures (red) and successes (blue) in achieving a targeted goal face for each solenoid

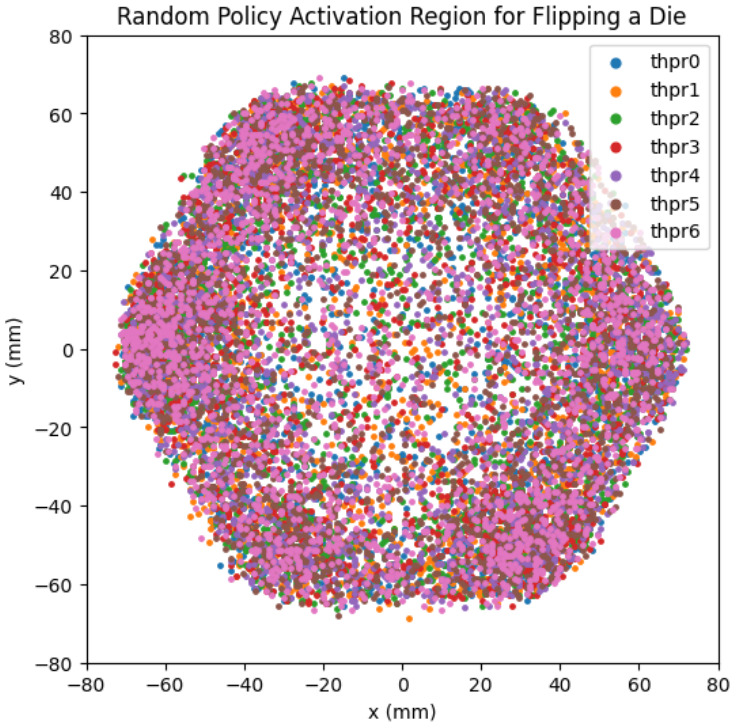


Figure 4.15: Random Policy: Die positions and the solenoid fired.

The first policy tested was the purely random-choice policy, which served as the experiment's control group. This resulted in an overall 5.1% success rate for rotating the die to a chosen face. The per-thumper activations and success rates are shown in Figure 4.14. As before, the die's initial position density variation strongly favors the corral wall and avoids the center. Since this is the random policy, we expected to see a uniform distribution of initial

positions versus thumper as shown in Figure 4.15. As we applied seven different color code associated to each solenoid, one can then expect that for an effective policy, there should not only be some degree of improvement in the number of successful flipping, but the activation region of each solenoid should be somewhat region-specific by color.

We are now in a position to consider a data-driven approach to approximating $h(s, x, y, \theta, \mathbf{u})$ — the function that predicts the probability p of bringing the part into a desired configuration given the state (s, x, y, θ) and an impulse $\mathbf{u}$; We have the entire 62,500 ground-truth data points for use as the base data for the rN policy.

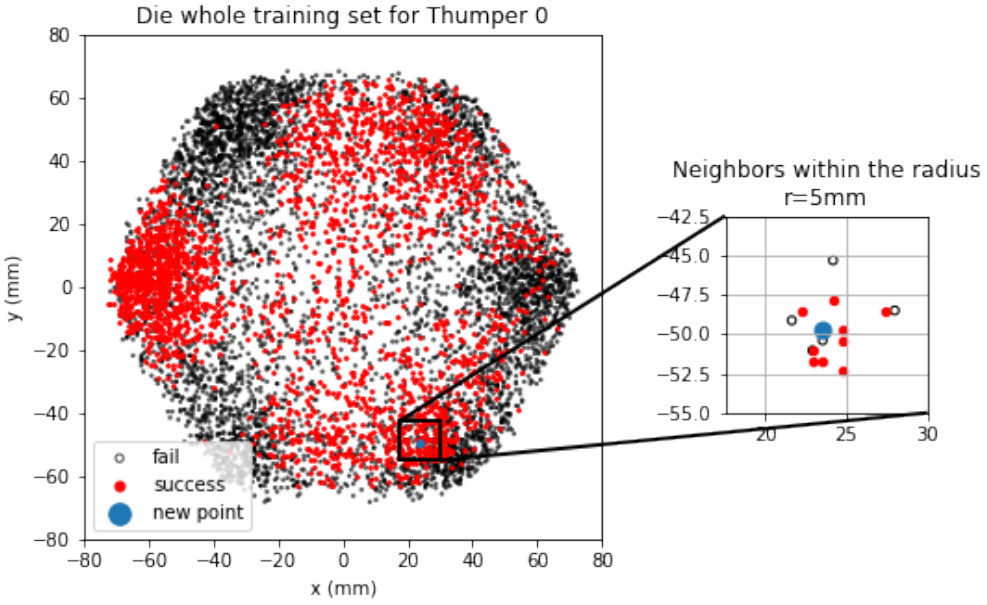


Figure 4.16: Thumper0 rN classifier accessing the neighborhood of a sample at [23, -49, 196]; inset shows the location and flipping results of each neighbor

For each of the seven solenoids, we formed a list of all (s, x, y, θ) examples within the $r = 5$ radius (Figure 4.16). Based on these training examples, we calculated the success probability for each of the seven solenoids and select the one with the highest success probability. To determine impulse duration, we took the mean of the successful impulses for that solenoid. In the case of a tie between two solenoids, we chose one at random from the tied candidates. An example of the decision neighborhood for a die at [23, -49, 196] is shown in Figure 4.17.

Next, to access the flipping facet of those successful samples (red) in the inset plot, we used the die transition table in Figure 4.7 (a). As shown in Figure 4.17, the six down

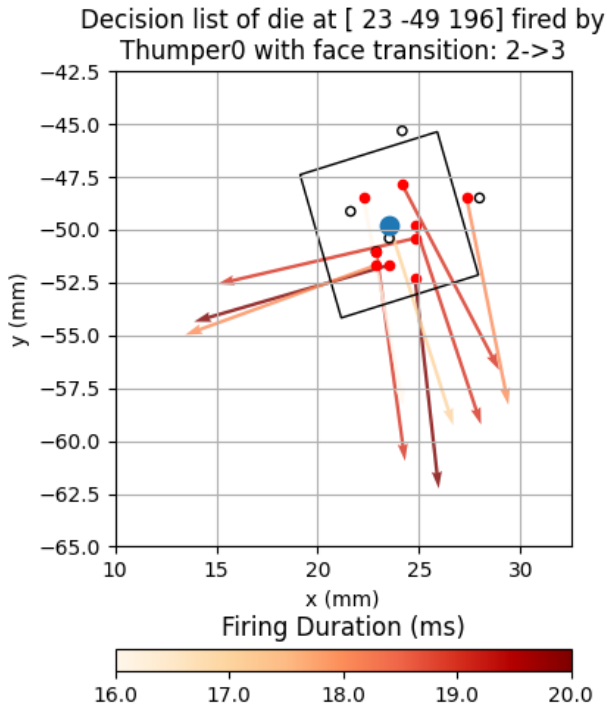


Figure 4.17: Sample decision list map of the die from Figure 4.16 at [23, -49, 196]; the neighbors within the radius r and their rolling directions are shown.

pointing arrows are transitions to face 3 while the three left pointing arrows are to face 6. If the goal state is face 3, we would chose Thumper0 with the mean firing duration - 17 ms as indicated by the arrow color. We fired Thumper0 at 17 ms, and the result was as predicted: the die landed on face 3. Now if the goal state is face 1 (pointing right) or face 4 (pointing up), we would examine the decision list of other 6 solenoids to find the best solenoid number and impulse duration pair.

As we are only choosing the best firing command without looking more than one step ahead, this is characterize as a greedy 1-step horizon approach to solve the under-actuated control problem.

Using this policy, we ran a 2,000-random-goals experiment. Figure 4.18 shows a scatter plot of the XY positions of the die, with the color of each dot indicating the particular solenoids chosen by the rN policy to have the best chance to rotate the die to another face. This plot forms a clear comparison with Figure 4.15 where the fixture of the firing positions of each solenoid are evenly mixed.

The final results for this policy are shown in Figure 4.19, with the rN policy achiev-

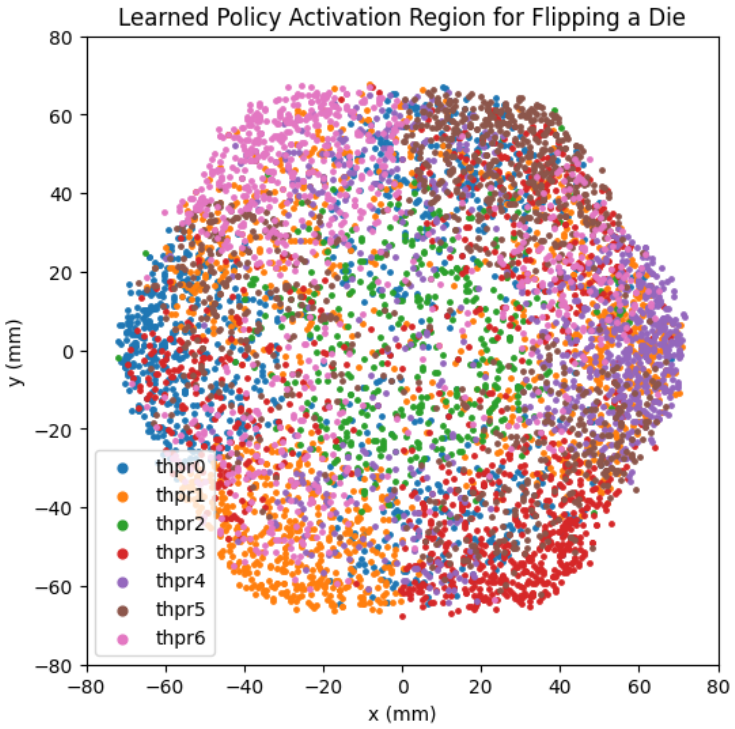


Figure 4.18: Die positions and the solenoid fired by the learned rN policy seeking a particular target face. The Voronoi-like segments are impure because the target face varies.

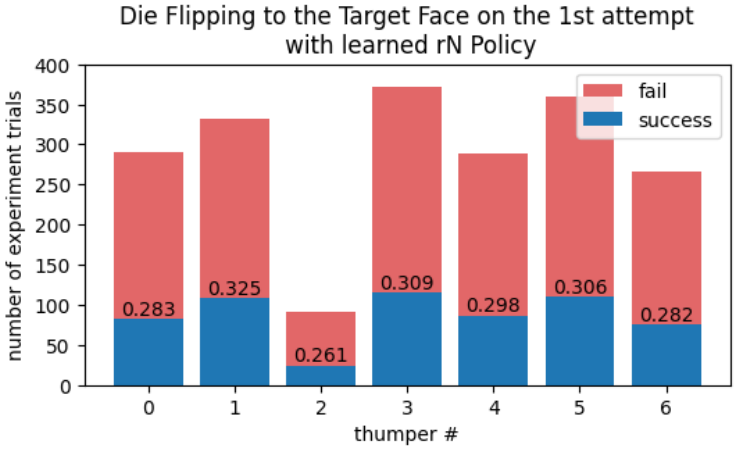


Figure 4.19: Number of impulses fired and successes. sorted by solenoid; the overall average success rate on the first impulse is 30.6%. Note the low density on the center solenoid (#2) is correctly accommodated by the rN policy.

ing the chosen goal state 30.6% of the time on the first impulse, beating the benchmark random policy by a factor of ∼6 times for single impulses. This indicates a very successful

manipulation of the die with an assumption-free memory-based approach.

5. LEARNING TO STAND A WOODEN SQUARE NUT

While the impact-based Thumper as a nonprehensile manipulator has limited actuation on parts, the die experiment has shown how complex manipulations can still be obtained using a learning-based method. When considering manipulation tasks in manufacturing settings, it is possible to have a mixture of high / low precision parts to be assembled together. Therefore, it is important to assess the robustness of Thumper when dealing with less uniform parts such as a square nut from a children's assembly set shown earlier in Figure 3.5.

Comparing with the precisely 3D-printed die, the nut is imprecise in almost every dimension. Measuring with a high quality digital caliper, the nut has dimensions of $\sim$35mm $\times$ $\sim$34mm $\times$ $\sim$10mm. The flat profile is actually a trapezoid as the side lengths vary by $0.5-1$ mm. As the nut is made of wood, density is not well controlled, nor is the location of the center of mass a constant; mass distribution and size of the nut will vary with humidity. Some of the edges are distinctly rounded rather than sharp. These conditions of the nut has led to several issues:

1. As the nut is rather thin (about 10mm thick), the desired goal state of the nut standing on edge is metastable and is statistically rare (thermodynamically unfavorable), even if there was adequate kinetic energy supplied in the solenoid impulse. This leads to heavily imbalanced training classes, with a lot more failures than successes.
2. Since the nut is slightly asymmetric, the force needed to stand the nut is different on each edge; however, it is hard for the vision system to identify such asymmetry on every location of the impulse bowl. This ambiguity leads to some degree of inaccuracy in the training process.
3. We also found that the apparatus must be well levelled and not tilted, as a tilt as small as 0.4° will cause the nut to "migrate" to the lowest corner of the apparatus.

Of these issues, the asymmetry issue was underestimated in the initial exploration; the consequences of this will be described in more depth below.

Portions of this chapter have been submitted to: C. Kong, W. Yerazunis, and D. Nikovski, "Stochastic Control of Object Pose With Under-Actuated Impulse Generator Arrays," *2023 IEEE Int. Conf. Mach. Learn. Appli. (ICMLA)*, Jacksonville, Florida, USA

5.1 Training Data Collection and Exploratory Verification

Having built the necessary infrastructure for operation, we began training data collection using a random firing policy where for each sample, we logged a start state (x, y, θ, s) where s has only two option "standing" and "laying", and fired a randomly chosen solenoid with randomly picked impulse duration (uniform distribution, ranging from 8ms to 25ms), yielding a new pose (x', y', θ', s') for the nut. Starting with a 30,000 sample data set, we first verified the computer vision (CV) state acquisition result. The OpenCV pipeline can correctly identify 99.6% of the experiment cases, and the unknown 0.4% are mostly cases where the nut leans on the bowl's wall. The 99.6% identified cases has zero false-negative rate (falsely identified as laying down), and about 1% false-positive rate (falsely identified as standing up). The overall result distribution resembles the die experiment in Figure 4.3 where data clusters toward the Thumper bowl's wall. However, there was a severe class imbalance, as shown in Figure 5.1.

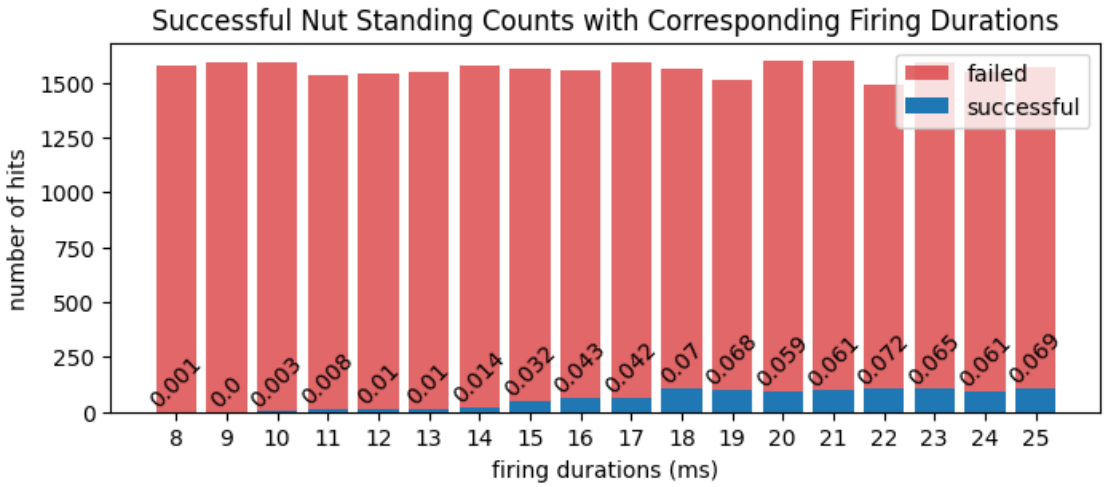


Figure 5.1: Nut standing (success) counts and rates versus firing duration

In the 30,000 sample database, there was about 2900 "successes" ($s' = 1$). These successes were distributed fairly evenly over the six peripheral solenoids, but the center solenoid had only seven successes out of $\sim$ 2900 impulses delivered- a small enough set that was dismiss as measurement error. The vast majority of the successful impulses in the exploratory set were uniformly distributed around the six peripheral solenoids.

Further analysis of the training data using Figure 5.1 showed a soft threshold behavior — for firing durations less than 14ms, the success in standing the nut up was essentially

zero. At pulse durations of 18ms or longer,the typical success rate over the range of firing durations was nearly constant.

Although the CV system could determine the (X, Y) centroid of the square nut to about one millimeter repeatability and the gross inclination of one facet of the cube between 0 and 90° to within one degree, the CV system was unable to resolve the position of the nut with respect to the eightfold corner symmetry of a flattened cuboid (that is, which of the eight corners of the flattened cuboid was in a particular corner of the image.)

To determine if this slight asymmetry in the flat nut would be significant, we used a three-point kinematic jig to place the nut at a fixed position (x, y, θ) laying down as shown in Figure 5.2 (a small dot of marker was added on one corner to allow a human to disambiguate the visual eightfold symmetry of the square nut).

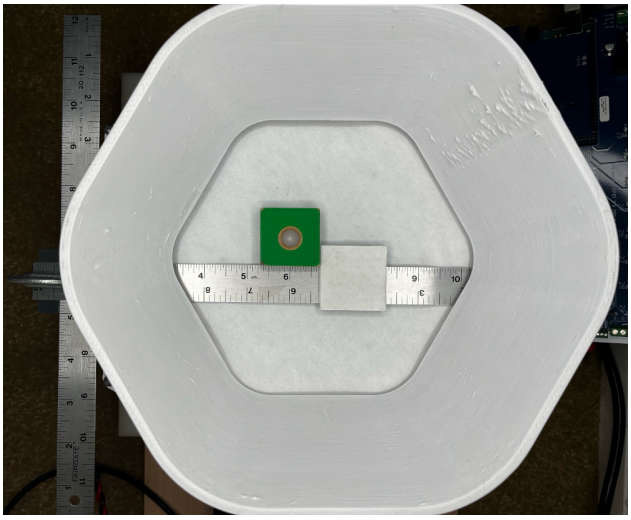


Figure 5.2: Nut positioned with kinematic jig for symmetry importance testing

With the jig, we conducted the repeatability test with firing durations varying from 15 ms to 25 ms in increments of 1 ms. For every firing duration, we repeated the positioning/firing cycle five times and recorded whether the nut remained laying or stood up. Using Thumper0, we would found a pose and a firing duration that provides a local maximum in the success rate, but when we rotated the nut 180 degrees on the Z axis and placed it back to its original (x, y, θ) location (representing the CV system's θ ambiguity), the best mode changed to a different firing duration, even without changing which solenoid was to fire. Similar changes were seen in the success rates and preferred impulse durations for all eight possible nut orientations, showing that there was a fourth significant state variable

representing the eightfold cuboid symmetry of the nut. This indicate that the variable s of the actual state of the nut (x, t, θ, s) is not fully observable.

This lack of observability of such state variable implies several obstacles to be overcome:

- Difficulty in establishing an effective regression algorithm to pick the best impulse duration for the nut.
- Difficulty in determining the most effective solenoid to fire
- Even assuming a perfect choice (100% success rate) of solenoid number and duration exists, there were only a 1 in 8 probability of selecting that perfect choice, so any rate better than 1 in 8 implies multiple nonzero-valued solutions for solenoid number and firing duration.

5.2 Determination of a Control Approach

While there are great uncertainties in controlling the wooden nut, the goal of the learning algorithm is just solving a binary classification problem: Given the initial state of a laying nut (x, y, θ, s) where $s = 0$ (laying), find the command input u_t, u_d such that the resulting state of the nut (x', y', θ', s') has $s' = 1$ (standing) instead of $s' = 0$. Using the same idea of learning a probabilistic model, we could consider two approaches:

1. Train seven separate binary classifiers — one per solenoid — that outputs a probability of the nut changing its status from laying to standing, given its current state (x, y, θ) (Die experiment)

2. Train one binary classifier that utilizes the 6-fold symmetry of the thumper bowl, combining all six peripheral solenoids together with suitable rotations to place the effective thumper at (-60,0). Output was also the optimal thumper to fire given the nut current state.

Approach 1 inherits similar learning model structure as the die experiment, but it has to deal with imbalanced training class data. For each of the seven solenoids, the exploratory set has ∼2900 solenoid firings, but only about 220 of the firings put the nut into the "successful" (standing) position. For a desirable a 1:1 ratio per classifier, we would need to have about 440 total samples per solenoid. Therefore, we down-sampled the unsuccessful firings randomly to obtain a 1:1 training set.

This opens the question of how well ~ 440 samples can cover the entire bowl. Given the diameter of the bowl at ~ 175.88 mm, we can approximate the area of the impact surface as a circle:

$$\begin{aligned} A_{bowlbottom} &= D^2\pi/4 \\ &= 175.88^2\pi/4 \\ &= 24295.33\text{mm}^2 \approx 243\text{cm}^2 \end{aligned} \tag{5.1}$$

A set of 440 samples can sample the 243 cm^2 bowl floor in X, Y at slightly better than a one sample per square centimeter density, which given the prior experience with the dice seems marginally acceptable.

Unfortunately, There is a third state variable - the rotation angle θ. As the angle ranges from $[0, 90]$ for a square shape, and binning the continuous variable θ into 10 bins, with a perfect distribution of starting locations, we would need $243 \times \frac{90}{10} = 2187$ or almost 5 times the current sample size to obtain a combined resolution of 1 cm and 10°, which edges into the domain of intractability.

Moving on to approach 2, While the state ambiguity and the class imbalance seem to render the control task exponentially harder than the previous die task, we found that the successful ($s' = 1$) location distribution of each peripheral solenoid exhibited very similar patterns — much similar than the die experiment — in the training data set. This observation was especially prominent when we looked at the location distribution from just one particular impulse duration. To confirm this observation, we conducted 20,000-sample random-firing exploratory run using just the firing duration of 20 ms — a strength that is not too strong but well into the successful plateau range in Figure 5.1. The result showed that:

1. The success rate of random policy was not seriously compromised with a reduced input space ($u_t \times u_d : 7 \times 17 \to 7 \times 1$, or $\to 6 \times 1$ if the center solenoid was considered ineffective)
2. The system did exhibit a sixfold symmetry with respect to the relative locations between the solenoid and the nut for a successful standing state (Appendix B.1)

This implies that the resulted manipulation induced by one peripheral solenoid can be generalized to other solenoids with the correct transformation. This 6-fold symmetry technique solves the lack of "standing" sample problem with the cost of limited input space. While

an additional downside of this approach was that the center solenoid Thumper2 would be completely left out due to lack of its standing class sample, the resulting shared training data pool has ∼2650 usable ground-truth samples, comfortably better than the desired 2187 samples mentioned from the previous calculations.

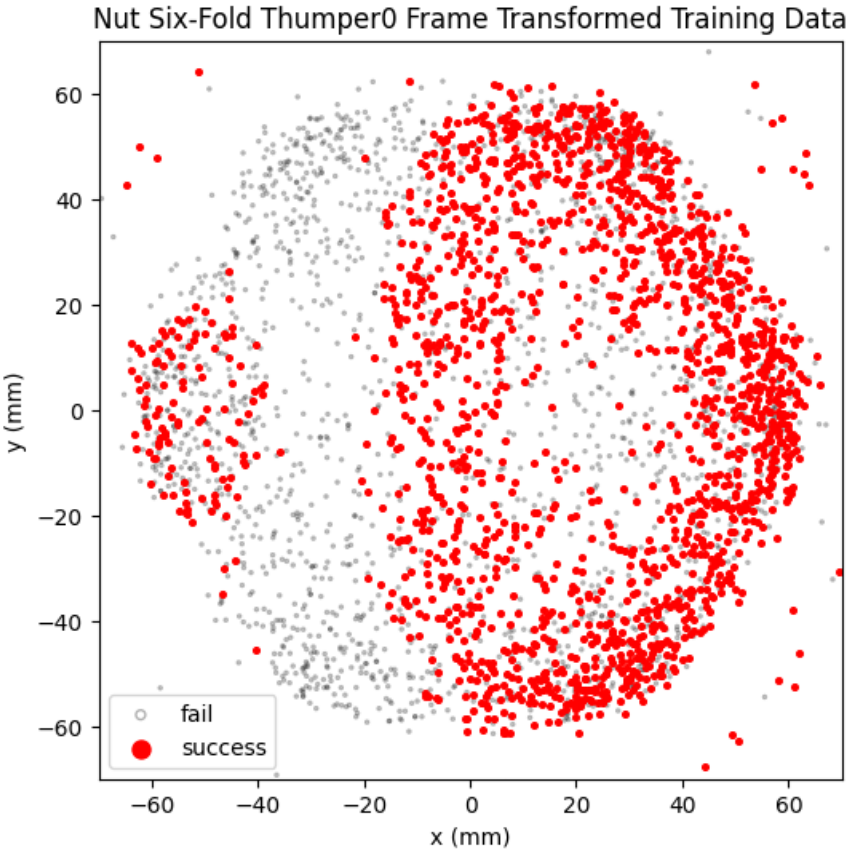


Figure 5.3: Transformed and Lumped Random Policy Initial Locations for Standing a Nut

For this to work, the entire physical system must first be symmetrical. As mentioned in Chapter 2, the solenoid mounting plate of the thumper system was made from polycarbonate plate and manufactured using a three-axis flatbed CNC mill with an accuracy of ∼50 microns. Using a 3D-printed checkerboard calibration plate, we found the geometric center of the bowl with sub-millimeter accuracy (Figure 3.3). We then set that point to be the origin of the world frame, and the frame of Thumper0 (shown in image) as the primary orientation on the negative X axis. Therefore, for any motion of the nut, we transformed its initial state

(x, y, θ, s) and final state (x', y', θ', s') from world frame to their own perspective frame[10]:

$$\begin{aligned} p_i &= \mathcal{R}_{io} p_o \\ &= \begin{bmatrix} \cos q_i & \sin q_i \\ -\sin q_i & \cos q_i \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \\ \theta_i &= \theta_o - q_i \end{aligned} \tag{5.2}$$

Where p_o, θ_o are initial / final states obtained by the camera in world frame and i is the frame id of six peripheral solenoid:

$$\begin{aligned} i &\in \{0, 1, 3, 4, 5, 6\} \\ q_i &\in \{0^\circ, 60^\circ, 120^\circ, 180^\circ, 240^\circ, 300^\circ\} \end{aligned} \tag{5.3}$$

As shown in Figure 5.3, instead of treating each sample as a manipulation from one solenoid, we propagated the motion to the other five. After down-sampling the "failures" (nut lying) to ensure a 1:1 ratio, we still have a six-times denser sample distribution which resulted a more defined decision boundary.

5.3 Algorithm Training and Control Result

Since the task is simply a binary classification, more algorithms can be considered comparing with the die experiment. Using the sci-kit learn python package, we trained the data in three classifiers — kNN, random forest (RF), and a radial-basis function SVM (RBF SVM) [19], [20]. We employed the same 10-fold cross-validation technique as the die task. The samples were randomly split into 10 bins; each bin maintains the outcome class ratio of 1:1. Given a classifier of interest, we used 9 bins to train, 1 bin to test; the AUROC curve of this train-test split was stored, and we repeated this process 10 times in total. In the end, the performance of this classifier, with a fixed set of hyper-parameters, would be expressed as the mean value of 10 AUROC curve values. In the end, the performance of this classifier, with a fixed set of hyper-parameters, would be expressed as the mean value of 10 AUROC curve values in Table 5.1.

[10]This is the inverse of the typical rotational transform matrix because the part coordinate is moving from world frame to a synthetic frame of reference

Table 5.1: Nut Experiment Trained Classifier Parameters

Classifier	Hyperparameters	AUROC μ	AUROC σ
kNN	k=24	0.715	0.020
RF	ntree=200 maxdepth=20 minleaf=10	0.745	0.017
RBF SVM	γ=1 C=10	0.728	0.022

Using the listed hyperparameter values, we ran an additional 20,000-sample testing experiment for each of the trained classifier. During the experiment:

1. The state of the nut (x, y, θ, s) was captured via the CV system.
2. It was transformed using Eq.5.2 to six orientations corresponding to six peripheral solenoids' frame of reference as in Eq.5.3.
3. The trained system was then called on each of the six transformed states.
4. The six resulting probabilities were obtained and compared.
5. Choose the result yielding the highest probability of success
6. Fire the solenoid with the highest probability of success
7. Capture the new state of the nut (x', y', θ', s') via the CV system.

Table 5.2: Nut Experiment Control Results per Classifier

Policy	standing (count)	standing (prob)	improvement
Random (control)	1817	9.08%	*control*
kNN	2793	13.9%	1.53 x
Random Forest	2673	13.4%	1.47 x
RBF SVM	2665	13.3%	1.46 x

The results of all three tested classifier were as follows in Table 5.2. Figure 5.4, 5.5, and 5.6 show the decision boundary plots of the corresponding classifier in the background where red regions are predicted to be successful initial locations for the nut to stand under a 20ms impulse at $(-60, 0)$. The scatter plot on top are the actual nut's firing locations taken using the results of the classifier.

At the first peek of all three plots, it is alarming to see how the decision boundaries of kNN is oddly shaped with the experimental result almost completely detached from the boundary. Meanwhile, the plot looks much more natural for the RF and SVM classifier. A possible explanation is how each algorithm fit the training data. kNN as a memory based

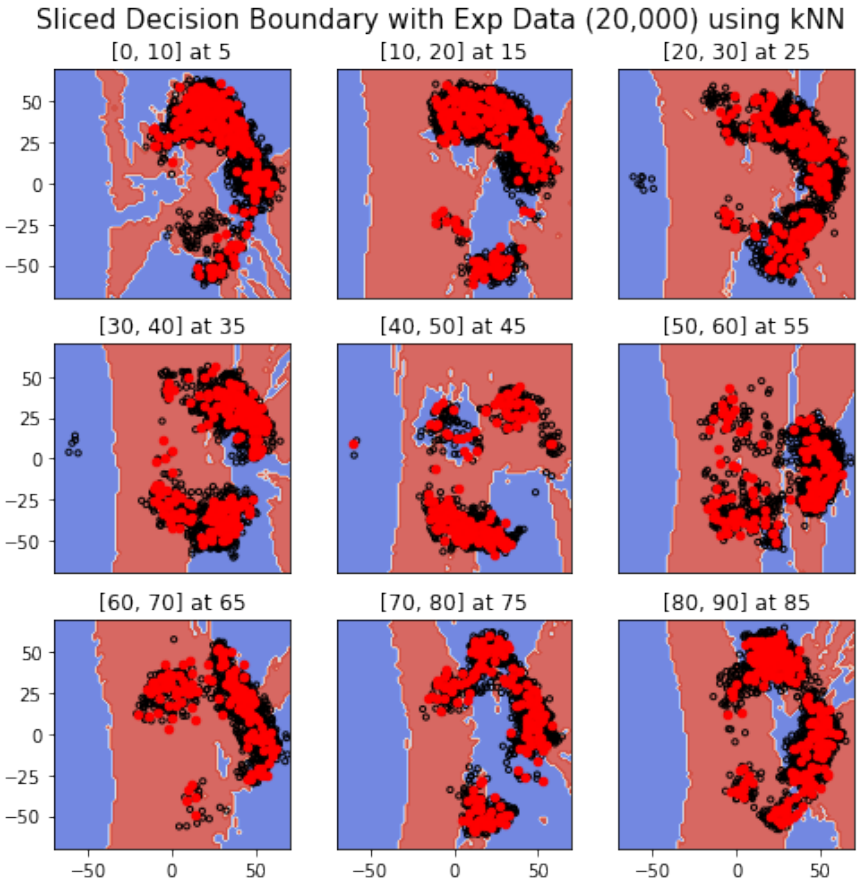


Figure 5.4: Flat nut kNN decision boundaries sliced at 5, 15, ..., 85 degrees in the background as θ of the experiment data varying in value from 0 to 10 degrees, 10 to 20 degrees, etc. up to 80 to 90 degrees; in each plot red dots are succeeded and black dots are failed cases.

method and can easily develop very step and discontinuous decision boundary if neighboring distance of a particular dimension become nonuniform. The conspicuous visual mismatch in Figure 5.4 is actually a result from fast changing decision boundary within even 10 degrees variation. With random forest (Figure 5.5), one can increase the number of tree to reduce over-fitting, and reduce the depth of the trees to increase smoothness. Therefore the cross-validation trained RF model has a shape that is close to the general distribution of the successful data distribution. Different from the other two methods, SVM takes assumptions of training data distributions. With the RBF Gaussian kernel, decision boundary are smoothened due to how distance are calculated. The relaxation parameter further flattened the decision boundary and resulted the current shape in Figure 5.6.

Comparing the stats listed in the training (Table 5.1) and result (Table 5.2), we found

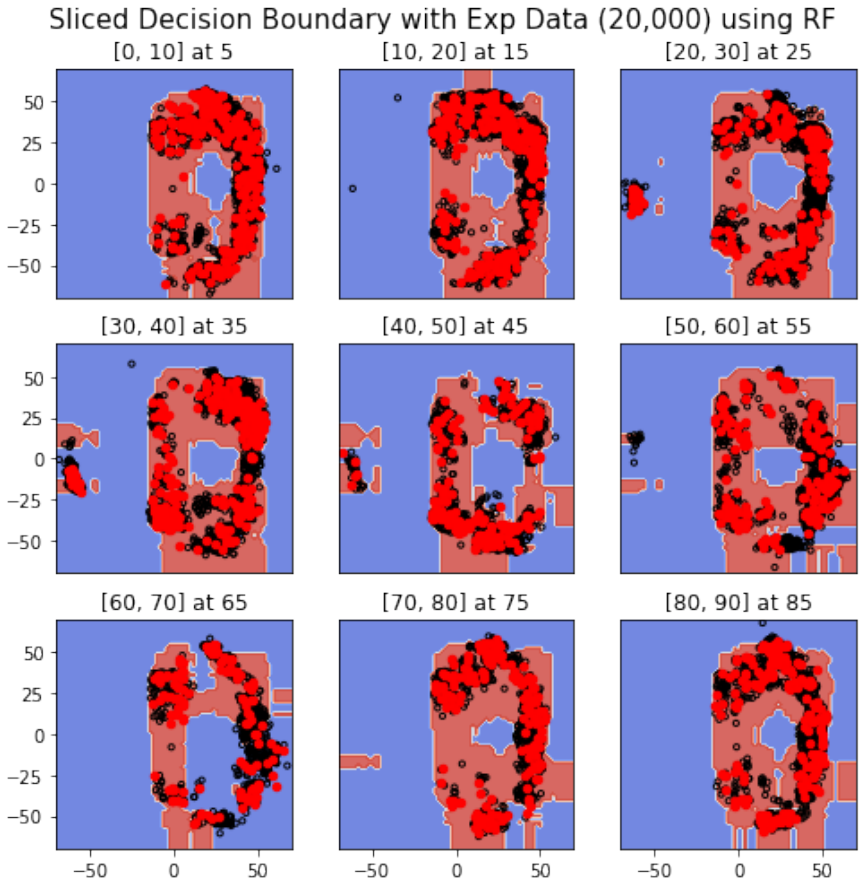


Figure 5.5: Flat nut RF decision boundaries sliced at 5, 15, ..., 85 degrees in the background as θ of the experiment data varying in value from 0 to 10 degrees, 10 to 20 degrees, etc. up to 80 to 90 degrees; in each plot red dots are succeeded and black dots are failed cases.

that the performance of the three binary classifiers are quite similar, albeit kNN achieved the highest performing result with the lowest AUROC during the cross-validation. In the last column of Table 5.2, we see that the improvements compared with the random policy was around 1.5 times at max. This was at odds with the live performance; specifically, using the kNN policy as an example, there seem to be regions in the thumper bowl (towards the center) where nut would stand instantly with less than three tries. Once the nut's position migrated towards the wall due to the bowl's structural tendency, it would take more than 10 tries to get to a successful hit, which is closer to how a random policy would perform. To further explore this previously undiscovered characteristic of Thumper, we payed a closer look to the temporal results of the testing experiments. Figure 5.7 shows the distribution of the nut location after the first failed attempt and after the sixth.

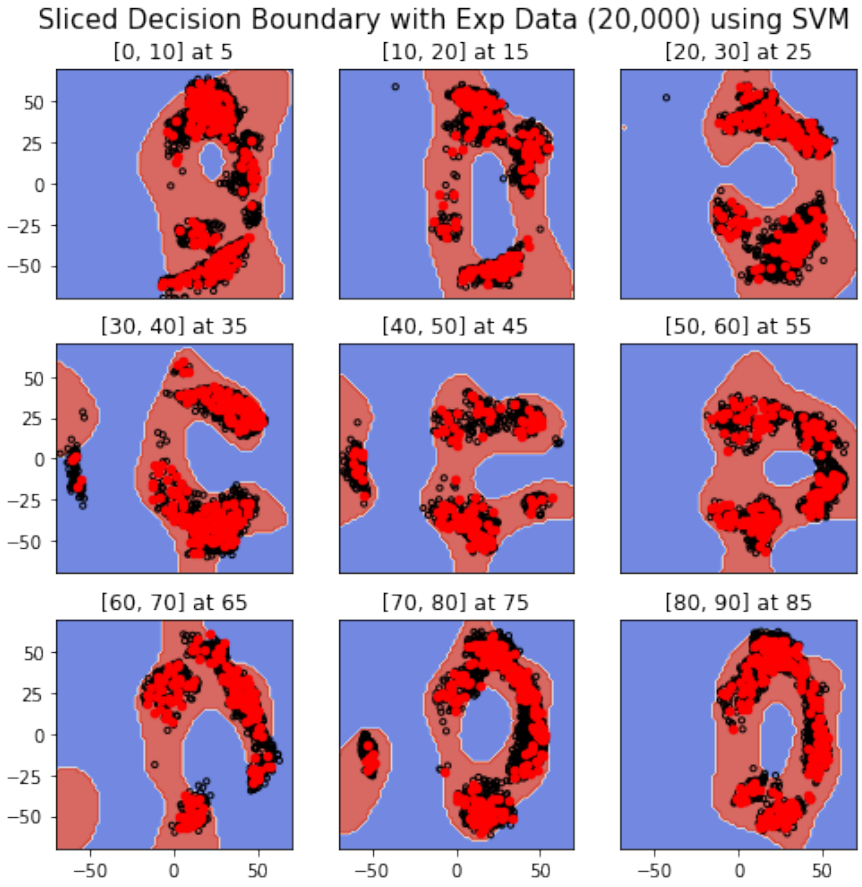


Figure 5.6: Flat nut SVM decision boundaries sliced at 5, 15, ..., 85 degrees in the background as θ of the experiment data varying in value from 0 to 10 degrees, 10 to 20 degrees, etc. up to 80 to 90 degrees; in each plot red dots are succeeded and black dots are failed cases.

We noticed an interesting phenomena here. First in Figure 5.7 (a), the position distribution after a first kNN-selected pulse that did not stand the nut up appears to be visually uniform, but after multiple impulses in, the still unsuccessful cases seem to form a distinct "attractor" region at the outer rim of the hexagonal corral wall. Such phenomena is in coherence with the live observation mentioned in the previous paragraph: the closer the nut is to the wall, the harder it is to stand (more N tries needed).

Along with this idea, we plotted the testing experiment success rate against the number of tries to stand the nut in Figure 5.8. The random policy exhibits a relatively consistent success rate around 0.09. The kNN (and other two policies) success rate exhibits a declining pattern, with the first impulse having a 0.211 success probability, the second impulse having an 0.192 probability, and so on. One possible explanation for this phenomenon is the evolving

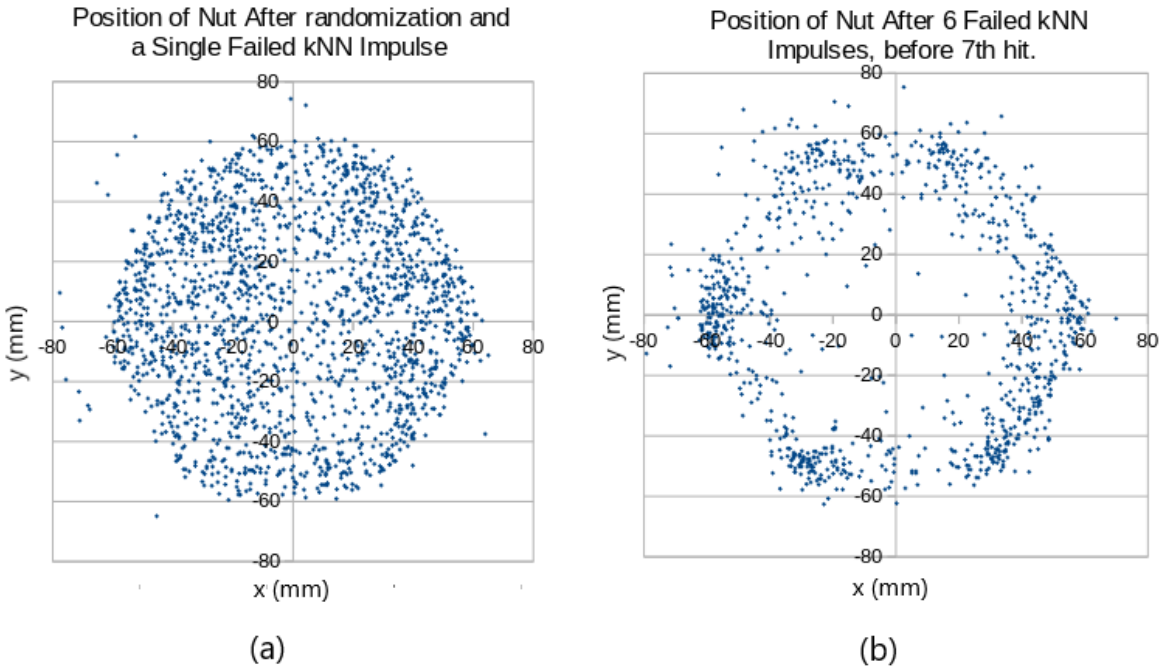


Figure 5.7: kNN testing experiment (x, y) positions of the flat nut after the first failed attempt (a), and the sixth failed attempt (b)

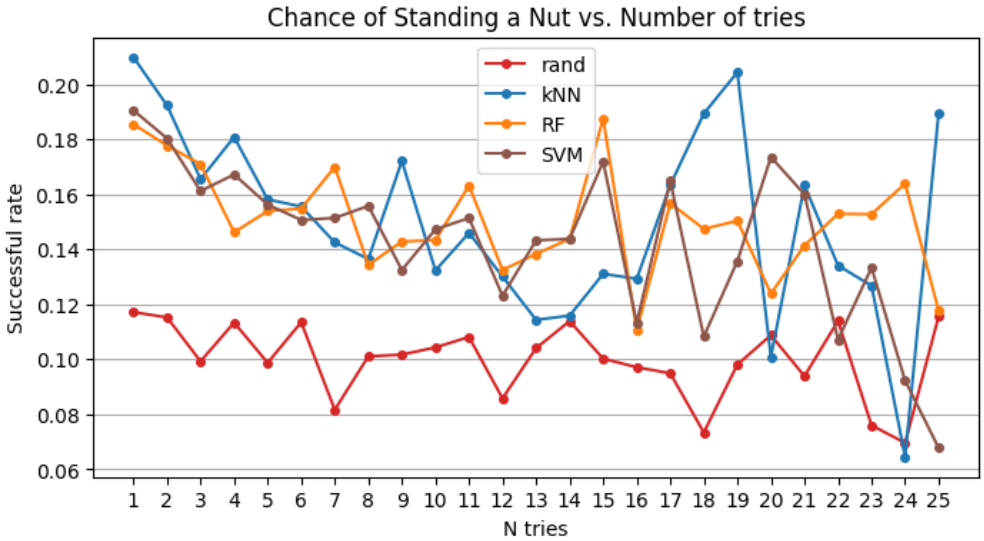


Figure 5.8: Success rates of each policies vs. the number of tries to stand the nut

distribution of the position of the nut after repeated failed attempts which was just discussed and shown in Figure 5.7. By averaging over all values of the success rate in each policy data, one can then obtain the same standing probabilities in Table 5.2.

The success rates plot in Figure 5.8 show the effectiveness of the control policy. Specifically, if the nut is placed around the center of the thumper bowl, the kNN policy can stand the nut with almost twice the chance of the random option.

6. WORKING WITH A ROBOT MANIPULATOR

In the past four chapters, we have included details about the design and control of Thumper in both hardware and software perspectives. Although Section 4.4.2 and 5.3 have shown promising result of Thumper manipulating parts into their desired goal states, the connection between Thumper and aforementioned robot assembly tasks in Chapter 1 (Figure 1.2) is not readily obvious. Therefore, we believe that by demonstrating the collaboration between Thumper and a 6-DoF industrial robot manipulator, a complete image of Thumper's industrial application would show.

6.1 Experiment Equipment and Design

With the addition of an ABB IRB 1200 robot manipulator (900 mm reach, 5 kg max load), we designed a proof of concept experiment with the setup shown in Figure 6.1. The manipulator was installed to the right hand side of Thumper in the camera's perspective. The object-of-interest for manipulation was chosen to be the 6-faced die. Due to the overall flat surface of the die, we mounted a 20 mm diameter suction cup onto the robot end-effector. The suction cup is attached to a spring loaded metal air tube; the spring ensures a soft contact between the robot tool tip and the die on the thumper impact plate. The content of the experiment is the following:

> *For each die part loaded into the Thumper bowl, manipulate the die such that the upward facing number ended up facing downward; once the manipulation is complete, the robot would take the die out.*

This experiment can be interpreted as a stamping task where the desired face to be stamped need to first be recognized by the camera mounted on the top and then manipulated to be facing down for a downward stamping motion. Such a task would be hard to accomplish with most state-of-the-art contact-rich manipulation techniques such as pick-and-place [8], pivoting [21], or rotating the object using tools [22], since training such a task from zero requires learning method that takes in large amount of states including the die's states, the force and torque detected on the tool center point (TCP), and the states of the robot end-effector which results in a long training time [23]. A common execution time for pick-

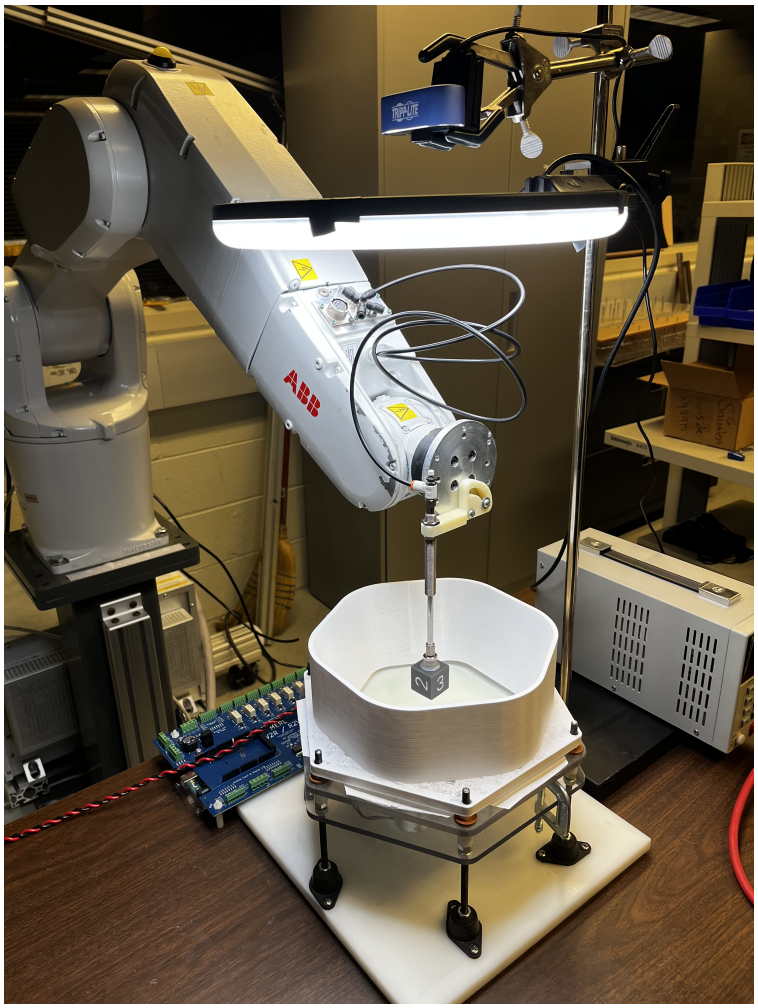


Figure 6.1: Thumper joined with an ABB IRB 1200 Robot Manipulator holding a die with a suction cup

and-place 180° manipulation using a robot arm would take around 15 seconds, pivoting with two fingers shortened the time greatly to around 5 seconds. Pivoting with a tool, however, would take more than 30 seconds.

An alternative approach would be to employ a more dexterous grasping tool such as Dactyl from OpenAI [24], or to utilize dual-arm manipulation such as the HERB robot [25]. These approaches would be more robust and faster but would cost greatly to develop and manufacture. Thumper facing manipulation task like this, however, would provide an economic solution with relatively fast (20 hours unsupervised) training time and execution time (less than 5 sec on average for the 180° rotation).

6.2 Method and Procedure

Taking a closer look of this experiment, the first half of which involved only Thumper can easily be achieved with some simple modifications on the 2000-random-goals testing experiment code in Section 4.4.2. The die's design followed the international casino standard where the opposing faces add up to seven. Therefore given the current die facet $s_{ini} = n$, the target facet would always be $s_{targ} = 7 - n$. Therefore, instead of randomly generating the next target facet, Thumper would send the current location of the part to the ABB robot, wait for the part to be retrieved, and start again once a new part arrives.

With the inclusion of an ABB robot, we made the decision of using a suction cup rather than a gripper due to the light weight and flat surface nature of the die. An extension on the suction cup mechanism ensured that the robot can reach positions closer to the white corral wall of the thumper bowl. When determining the placement of the robot, we realized that the previous spacing between the ring light / camera and the corral wall of thumper was too small for the robot end-effector and the long downward extending suction cup. While the tall metal stand shown in Figure 6.1 enabled a flexible adjustment of the ring light /camera height, the 640p webcam would have trouble recognizing the die's face number if the camera is too high up. As a result, we manually jogged the robot into its furthermost position and adjusted the height of the ring light / camera sets to their lowest possible positions.

Such a close operation distance between the robot arm and the ring light as shown in Figure 6.1 are discouraged under most circumstances and would require testing of robot control programs in a simulation to ensure safety. We used a simulation software built by ABB called Robot Studio. The library of Robot Studio contained the exact model of the ABB IRB 1200 robot used in this experiment. While reconstructing the entire experimental setup in the simulator with precise distance between Thumper and the robot measured was an option, we adopted an alternative approach where the robot's motion were planned based on a calibrated position of Thumper in the robot base frame.

6.2.1 Robot-Thumper Frame Calibration

To establish a relationship between Thumper's frame and the robot base frame, we adopted the following notation format. Let $(\mathcal{E}_P, \mathcal{O}_P)$ denotes Thumper's impact plate where $\mathcal{E}_P$ represents the orthonormal frame calibrated in Section 3.1 shown in Figure 3.3, and $\mathcal{O}_P$ represent the origin of Thumper's frame at the very center of the impact plate. Let $(\mathcal{E}_B, \mathcal{O}_B)$

denotes the robot base where the origin $\mathcal{O}_B$ is located at the bottom of the robot base with the z-axis of $\mathcal{E}_B$ pointing straight up. The direction of the x-axis of $\mathcal{E}_B$ is considered to be the front of the robot, and in Figure 6.1, it is pointing towards the direction of where Thumper is installed.

To find p_{BP}, the position of $\mathcal{O}_P$ in $\mathcal{E}_B$, and R_{BP} the transformation from $\mathcal{E}_P$ to $\mathcal{E}_B$, three points from Thumper were needed: 1) the point of origin $\mathcal{O}_P$, 2) a point on the x-axis of $\mathcal{E}_P$, and 3) a point on the y-axis of $\mathcal{E}_P$. Note that the goal is to represent $(\mathcal{E}_P, \mathcal{O}_P)$ in the $(\mathcal{E}_B, \mathcal{O}_B)$ frame. We needed vectors that represent 1), 2), and 3) in robot base frame's coordinates. Instead of taking measurements by hand, the most straight forward method is to utilize the forward kinematics of the ABB robot.

The 6-DoF robot arm has the following iterative formulation of frame rotation:

$$\begin{aligned}
\mathcal{E}_1 &= \mathcal{R}(\vec{h}_1, q_1)\mathcal{E}_0 \\
\mathcal{E}_2 &= \mathcal{R}(\vec{h}_2, q_2)\mathcal{E}_1 = \mathcal{R}(\vec{h}_2, q_2)\mathcal{R}(\vec{h}_1, q_1)\mathcal{E}_0 \\
&\vdots \\
\mathcal{E}_6 &= \Pi_{n=1}^{6}\mathcal{R}(\vec{h}_n, q_n)\mathcal{E}_0
\end{aligned} \tag{6.1}$$

In Equation 6.1, $\mathcal{E}_0 = \mathcal{E}_B$ is the robot base frame. $\mathcal{E}_1, \ldots, \mathcal{E}_6$ represents orthonormal frames of each link of the 6-linked ABB robot. Due to the linked connections between each robot joint link, each joint frame $\mathcal{E}_n$ can be represented in the base frame $\mathcal{E}_0$ in terms of propagating rotational operation. $\mathcal{R}(\vec{h}_n, q_n)$ is the rotational operation from $\mathcal{E}_n$ to $\mathcal{E}_{n-1}$. $\vec{h}_n$ is the rotational axis of the n_{th} robot joint, and q_n is the joint angle in radian. The corresponding rotation matrix projected in $\mathcal{E}_0$ can be expressed using the Rodriguez Rotation Formula, $\mathsf{R}(h_n, q_n) = e^{h_n^{\times} q_n} \in SO(3)$ where $h_n^{\times}$ is a 3-by-3 skew-symmetric matrix that functions as a cross product in a matrix multiplication fashion. To simplify the notation of frame rotation matrices, we defined the following representations:

$$\begin{aligned}
R_{01} &:= \mathsf{R}(h_1, q_1) = e^{h_1^{\times} q_1} \\
R_{02} &:= \mathsf{R}(h_1, q_1)\mathsf{R}(h_2, q_2) = e^{h_1^{\times} q_1} e^{h_2^{\times} q_2} \\
&\vdots \\
R_{06} &= \Pi_{n=1}^{6}\mathsf{R}(h_n, q_n) = \Pi_{n=1}^{6} e^{h_n^{\times} q_n}
\end{aligned} \tag{6.2}$$

Since the suction cup frame (tool frame) $\mathcal{E}_T$ inherent the orientation of the sixth frame $\mathcal{E}_6$, $R_{0T} = R_{06}$.

Following this representation, with joint angles between each link known, the location of the suction cup tip $\mathcal{O}_T$ in the robot base frame $\mathcal{E}_B$ can be solved at any moment as:

$$p_{0T} = p_{01} + R_{01}p_{12} + \ldots + R_{06}p_{6T} \tag{6.3}$$

Where $p_{i,i+1}$ are known lengths of each link represented in their own frame $\mathcal{E}_i$.

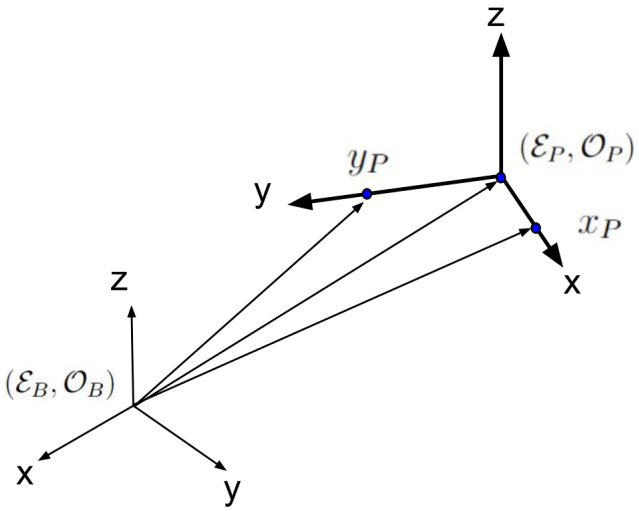


Figure 6.2: Defining Thumper frame $\mathcal{E}_P$ in the robot base frame $\mathcal{E}_B$ using three points $\mathcal{O}_P, x_P, y_P$

Now to sample the three point-of-interest to define the relative position and orientations between the Thumper frame $\mathcal{E}_P$ and the robot base frame $\mathcal{E}_B$, we simply put Thumper in calibration mode (Figure 2.5) and manually jog the suction cup of the robot p_{0T} to the origin $\mathcal{O}_P$, and two arbitrary points on Thumper's x, y axis shown in Figure 6.2, and we name them x_P and y_P respectively. Using Equation 6.3, three corresponding vectors in $\mathcal{E}_B$ are obtained as:

$$\begin{aligned} p_{BP} &= p_{\mathcal{O}_P} - p_{\mathcal{O}_B} \\ p_{Bx} &= p_{x_P} - p_{\mathcal{O}_B} \\ p_{By} &= p_{y_P} - p_{\mathcal{O}_B} \end{aligned} \tag{6.4}$$

Next, to define the orthonormal frame of Thumper $\mathcal{E}_P$, we calculated the unit vectors

in $\mathcal{E}_B$:

$$\begin{aligned} e_x &= \frac{p_{Bx} - p_{BP}}{||p_{Bx} - p_{BP}||} \\ e_y &= \frac{p_{By} - p_{BP}}{||p_{By} - p_{BP}||} \\ e_z &= e_x^{\times} e_y \end{aligned} \tag{6.5}$$

Three unit vectors in Equation 6.5 form an orthonormal basis of $R_{BP} \in SO(3)$:

$$R_{BP} = \begin{bmatrix} | & | & | \\ e_x & e_y & e_z \\ | & | & | \end{bmatrix} \tag{6.6}$$

With the matrix in Equation 6.6, any location of the die obtained from the Thumper frame can be represented in the robot base frame $\mathcal{E}_B$ as:

$$p_{Die} = p_{BP} + R_{BP} \begin{bmatrix} x \\ y \\ 25 \end{bmatrix} \tag{6.7}$$

where x and y are the die's location obtained from the camera, and 25 is the height of the die above the impact plate in millimeters.

Thus the completed calibration enabled the robot to fetch the die based on the locations provided by the camera of the Thumper Assembly.

6.2.2 Communication and Control

At this point, the ABB robot is aware of the location of Thumper in its own frame; however, to achieve collaboration, the robot need to know:

- whether Thumper has finished the manipulation task
- if done, where is the exact location of the die in Thumper frame after the manipulation

Thumper, on the other hand, need to know:

- if the robot is current busy (can robot run a new job)
- if the location of the robot arm is still in the camera's vision field (can robot buffer a job to be complete while the current job is finishing)

To establish communication between these two independent hardware, we used an open source interfacing software called Robot Raconteur (RR) [26]. RR holds driver program for numerous industrial robot manipulators including the ABB IRB 1200 driver module that can be executed with Python3 on my Linux computer that controls the camera and Arduino of Thumper (Figure 1.2). With the RR driver running in the background, the live states of the robot were broadcasted onto the lab's wireless network. Within the python control module of Thumper, we added an RR wire subscription so that direct commands obtaining and controlling the robot can be sent with reference to the vision information.

The RR driver for ABB robots supports various command mode from jogging to a preset joint configuration to accurate waypoints interpolations in joint space [27]. By including the robot definition tools from RPI's General Robotics Toolbox [28], we could jog the robot's suction cup p_{0T} in the Cartesian coordinate space of the robot base frame $\mathcal{E}_B$. Specifically, we first calculated the errors between the current suction cup spacial position $P(q_k) := (p_{0T}(q_k), R_{0T}(q_k))$ with joint angles q_k and the desired spacial position P_d, obtained the desired spacial velocity $V_d := (v_d, \omega_d)$ of the suction cup to reduce the error. Then, to solve for the robot motion needed to achieve V_d, we used quadratic programming to solve the following optimization problem:

$$\dot{q}_{k+1} = \arg\min_{\dot{q}} \|J(q_k)\dot{q}_k - V_d\|^2 \tag{6.8}$$

In Equation 6.8, $J(q_k) \in \mathbb{R}^{6\times 6}$ is the Jacobian matrix at the current joint angle $q_k \in \mathbb{R}^6$ that maps the joint angular velocity to the spacial velocity of the suction cup as:

$$V_k = J(q_k)\dot{q}_k \tag{6.9}$$

The resulted joint angular velocity $\dot{q}_{k+1}$ was then passed directly to the RR driver as velocity control command. The command control loop was set to run at 100 Hz which ensured an accurate velocity profile reducing the error to reach the desired suction cup spacial position.

While most motion plannings of such robot involves joint limit constraint setup, singularity analysis, and / or obstacle avoidance, the motion space of this pick-and-place task was relatively stationary with preset waypoints such as the home position, the picking up position, and the dropping off position. To deal with potential errors from the camera provided coordinates, a circular boundary was set along the perimeter of the corral wall during

the die picking command. Lastly, the control program was always tested in the ABB Robot Studio simulator before deployed to the actual hardware.

6.3 Execution and Result

The Experiment Control Program (ECP) started by loading camera calibration, die face recognition, and trained radius neighbor classifier parameters. With the Robot Raconteur (RR) driver running in the background, the ECP initialized a robot pick-and-place control program in parallel using python's Threading mechanism. After the robot is homed, the experiment began.

A Die was thrown into the Thumper bowl, and when its face number was determined by the over-hanging camera, an action according to the 1-horizon greedy policy was made to flip the bottom face of the die up. Once the target face was achieved, the coordinate of the die was sent to the robot control thread, and the robot start reaching for the die. The robot arm picked up the die, and as soon as the arm moved completely out of the camera's vision field, Thumper was then actively anticipating new incoming die.

Now, if the robot finished the dropping off and returned to its waiting position before the Thumper finishes its manipulation, then ECP would follow the same logic flow from the previous paragraph. In most cases, however, Thumper actually finished the manipulation of the second die before the robot arm was done with the previous task. In this situation, ECP would pulse Thumper, load the current die location into a buffer stream, and push the command right after the previous task finished. Note that currently Thumper support only one part at a time as the collision between parts was yet accounted by the the learned policy. An action clip series of the experiment is shown in Figure 6.3.

Table 6.1: Ten Experiment Result with Execution Time

exp #	01	02	03	04	05	06	07	08	09	10	Avg.
s_{ini}	5	4	2	1	6	2	3	3	6	2	—
s_{targ}	2	3	5	6	1	5	4	4	1	5	—
n_{tries}	4	4	1	5	3	9	6	3	2	9	4.6
t_{takt} (sec)	3.21	4.51	1.03	4.5	3.14	8.1	6.3	2.3	1.89	8.86	4.38

In Figure 6.3 (a), the experiment started with a die face number 5, and the goal was to get to 2; (b) shows the flipping right before reaching the target face. (c) and (d) shows

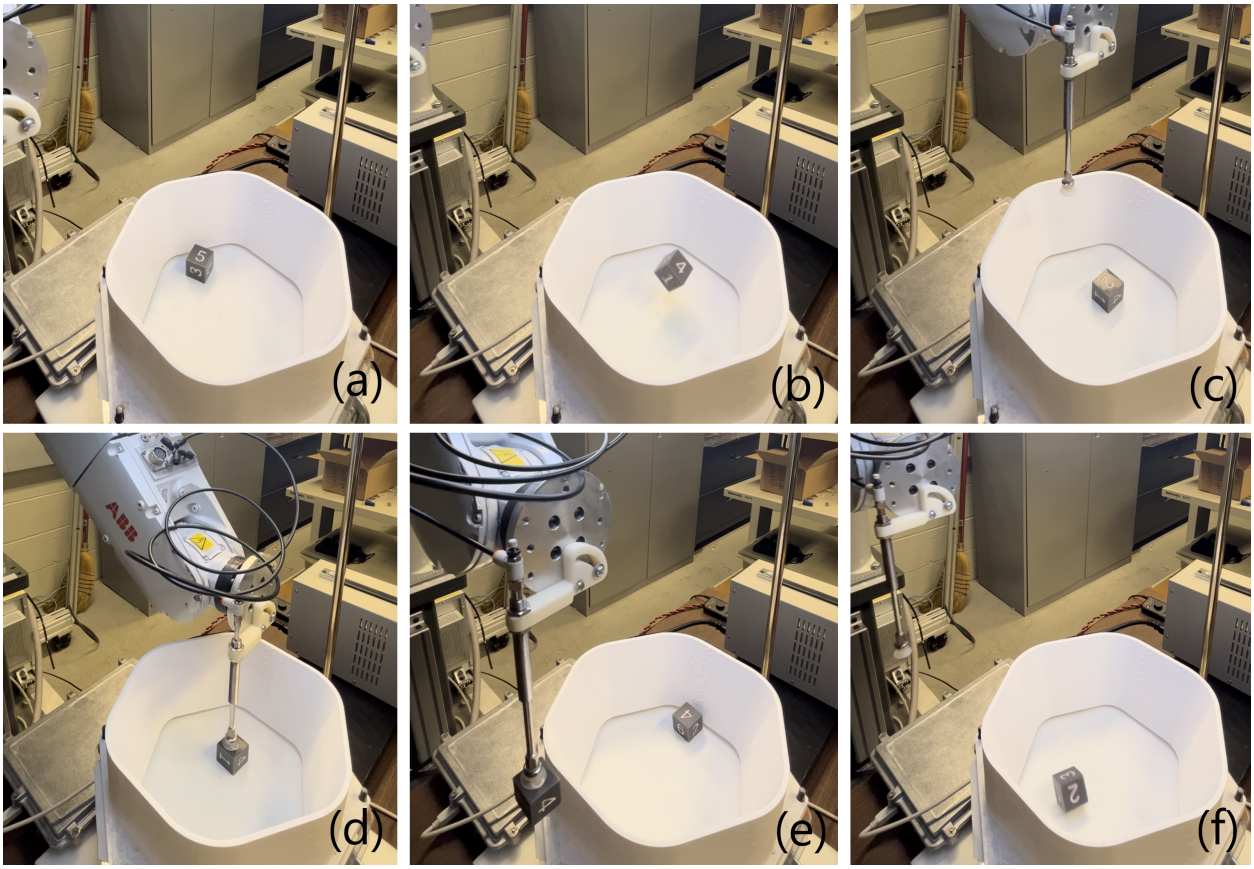


Figure 6.3: Exp "Flipping the bottom face up" action clips; (a,b) flipping from $5 \rightarrow 2$; (c,d) robot picking up the die; (e,f) flipping a new die from $4 \rightarrow 3$

that the robot has received the coordinate of the die and picked up the die with its suction cup. (e) shows that as the arm was carrying the die out of the camera vision field, a new die was loaded into the Thumper bowl. (f) shows the frame right before the die was flipped to the target face 3 from a 4, and meanwhile, the robot had dropped off the first die.

We continued this experiment till ten manipulations have been done. Results including the first two trials in Figure 6.3 are shown in Table 6.1. From the calculated average values of the ten trials at the last column of the table, we saw that the average execution time needed for flipping the die's bottom face up using Thumper was less than 5 second. This is a competent performance result comparing with other manipulation method for this kind of task demonstrated in [8].

7. CONCLUSIONS AND FUTURE WORKS

This thesis presented a working prototype of the nonprehensile impulse-based object manipulator. From the physical design of the assembly to the software tools used to control each component, the unique functionality of Thumper was discovered by directly interacting with objects under gravity. On top of the chaotic nature of impulse-based manipulation, the lack of simulation rendered the learning process slow and challenging. Such difficulty motivated the development of a fast and robust vision system whose functionality can be apply to a large variety of recognition problems beyond this thesis.

The two manipulation experiments on the six-faced die and wooden nut showed the effectiveness of Thumper for object orientation when driven by ML controllers, even with no prior knowledge of the actual contact physics. In addition to the already presented control result, the tuned Nearest Neighbor classifiers have demonstrated their robustness against changes in operation environment as similar statistics were obtained when the device was reassembled in a different lab space. The six-fold increase in the success rate of the die manipulation, the two-fold improvement in the success rate of standing the wooden nut, combined with the minimal need for manual supervision of the method (all training is self-supervised), could possibly result in a very fast and cost-effective method for part manipulation for robotic assembly.

Such idea was enhanced by the Robot-Thumper collaborative experiment in Chapter 6, where an example of the physical setup and communication strategy were demonstrated under the context of a common manipulation task on the six-faced cubical die. The resulted execution time was on pair with the best state-of-the-art pivoting approach.

7.1 Future Directions

As we conclude the project in terms of a master thesis, we see improvements and future research directions in the topic of learning algorithm and multi-object manipulation.

Portions of this chapter are to appear as: C. Kong, W. Yerazunis, and D. Nikovski, "Learning Object Manipulation With Under-Actuated Impulse Generator Arrays," 2023 Am. Control Conf. (ACC), San Diego, CA, USA

7.1.1 Extending to a RL-based Controller

When dealing with the initial data generated from random manipulation, instead of resorting to the trending "Black-Box" approaches such Neural Networks, we chose one of the simplest learning algorithms — Nearest Neighbors. This consideration was based on not only the memory-based vs. model-based comparison, but the characteristics of the data structure: low feature space dimension with relatively large sample size. We believe that the well received final performance of the die control experiment was a direct result of the learning algorithm; specifically, when we can't make any assumptions of the data distributions, the algorithm with the least number of hyper-parameters (Nearest Neighbors) was the safer choice.

However, during the actual performance of the learned 1-horizon greedy policy, we observed some inefficiencies during the manipulation: when the part was getting close to the edge of the impact plate, as the controllability of Thumper in rotation gets relatively low, lots of unsuccessful attempts were made. An alternative would be: first move the part to the center of Thumper where rotations are much easier, then perform the actual flipping action. Such alternative challenges the current 1-horizon greedy policy and expresses a great motivation in moving to a multi-horizon policy where the optimal policy considers future outcomes.

In fact, the initial data recording structure in Table 4.1 has hinted the connectivity between the die control experiment set up (Section 4.4) and the four fundamental elements for a Markov Decision Process based Reinforcement Learning [29]:

- $\mathbf{S}$, a set of states for the agent — $\mathbf{s} := (s, x, y, \theta)$
- $\mathbf{A}$, a set of actions that the agent can take — $\mathbf{a} := (u_t, u_d)$
- $P_{\mathbf{a}}(\mathbf{s}, \mathbf{s}')$, the state transitional probability under the condition of a certain action $\mathbf{a}$
- $R_{\mathbf{a}}(\mathbf{s}, \mathbf{s}')$, the immediate reward after such transition under the condition of action $\mathbf{a}$

To calculate an effective $P_{\mathbf{a}}(\mathbf{s}, \mathbf{s}')$, the sample size calculation made in Section 4.3 has also given an estimate that about 100 times more data (6,666,660 samples) would be needed to provide enough coverage on a state / action space discretization of

$$s \times (x \times y) \times \theta \times u_t \times u_d := 1 \times (962) \times 9 \times 7 \times 11 = 666,666 \text{ bins} \tag{7.1}$$

Meaning that Thumper will be running for 3 months straight. Since this approach might

surpass the physical strength of the device, alternative approach like SARSA or Q-learning that directly learns the value function can be an interesting direction. Another possible route is the kNN-TD approach in [30] which produces a probabilistic representation of the input state signal to construct robust state descriptions. This method seems to be able to incorporate the already tested Nearest Neighbor parameter and produce fast converging results.

7.1.2 Multi-Object And / Or Multi-Solenoid Manipulation

Another future direction is to simultaneously manipulate multiple parts, with more than one solenoid firing simultaneously. The challenge in this topic lies in multi-object localization and motion prediction, especially when two or more part is in contact with each other.

Given the wide range of motion of just one robot arm, a larger Thumper with alternative fixture / solenoid arrangements can also be explored, featuring multiple robot arms picking up multiple correct oriented parts at the same time. This way, Thumper can work with robot arms to perform singulation (separate lumped parts) and orientation at the same time.

LITERATURE CITED

[1] Saratha Electrical Works, "SS Vibratory Bowl Feeder," https://www.indiamart.com/proddetail/ss-vibratory-bowl-feeder-19401580833.html?pos=4&pla=n, [Accessed: 2023-02-28].

[2] G. C. Deveol, "Programmed article transfer," U.S. Patent 2 988 237, June 13, 1961.

[3] M. T. Sgriccia, "Feeder bowl," U.S. Patent 2 654 465, October 6, 1953.

[4] R. Song, F. Li, T. Fu, and J. Zhao, "A robotic automatic assembly system based on vision," *Appl. Sci.*, vol. 10, no. 3, 2020.

[5] D. Berkowitz and J. Canny, "A comparison of real and simulated designs for vibratory parts feeding," in *IEEE Int. Conf. Rob. Autom. (ICRA)*, vol. 3, 1997, pp. 2377–2382 vol.3.

[6] R. Horaud, F. Dornaika, and B. Espiau, "Visually guided object grasping," *IEEE Trans. Robot. Autom.*, vol. 14, no. 4, pp. 525–532, 1998.

[7] T. H. Vose, P. Umbanhowar, and K. M. Lynch, "Sliding manipulation of rigid bodies on a controlled 6-dof plate," *Int. J. Rob. Res.*, vol. 31, no. 7, pp. 819–838, 2012.

[8] A. Holladay, R. Paolini, and M. T. Mason, "A general framework for open-loop pivoting," in *IEEE Int. Conf. Rob. Autom. (ICRA)*, 2015, pp. 3675–3681.

[9] M. T. Mason and J. K. Salisbury, "Impedance control and internal model use during the initial phase of manipulation learning," *J. Acoust. Soc. Am.*, vol. 109, no. 5 Pt 1, pp. 2161–2175, 2001.

[10] C. G. Atkeson, A. W. Moore, and S. Schaal, "Locally weighted learning for control," *Artif. Intell. Rev.*, vol. 11, pp. 75–113, 1997.

[11] Z. Zhang, "A flexible new technique for camera calibration," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 22, no. 11, pp. 1330–1334, 2000.

[12] C. Tomasi and R. Manduchi, "Bilateral filtering for gray and color images," in *IEEE Int. Conf. Comput. Vis.* IEEE, 1998, pp. 839–846.

[13] J. Canny, "A computational approach to edge detection," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 8, no. 6, pp. 679–698, 1986.

[14] J. P. Lewis, "Fast normalized cross-correlation," in *Vis. Interface*, vol. 10, no. 1. Citeseer, 1995, pp. 120–123.

[15] M. I. Jordan and D. E. Rumelhart, "Forward models: Supervised learning with a distal teacher," *Cognit. Sci.*, vol. 16, no. 3, pp. 307–354, 1992.

[16] T. Cover and P. Hart, "Nearest neighbor pattern classification," *IEEE Trans. Inf. Theory*, vol. 13, no. 1, pp. 21–27, 1967.

[17] T. Fawcett, "An introduction to roc analysis," *Pattern Rcognit. Lett.*, vol. 27, no. 8, pp. 861–874, 2006.

[18] S. A. Dudani, "The distance-weighted k-nearest-neighbor rule," *IEEE Trans. Syst., Man, Cybern.*, vol. SMC-6, no. 4, pp. 325–327, 1976.

[19] L. Breiman, "Random forests," *Mach. Learn.*, vol. 45, no. 1, pp. 5–32, 2001.

[20] C. Cortes and V. Vapnik, "Support-vector networks," *Mach. Learn.*, vol. 20, no. 3, pp. 273–297, 1995.

[21] R. Antonova, S. Cruciani, C. Smith, and D. Kragic, "Reinforcement learning for pivoting task," arXiv:1703.00472 [cs.RO], Mar. 2017.

[22] Y. Shirai, D. K. Jha, A. U. Raghunathan, and D. Hong, "Tactile tool manipulation," arXiv:2301.06698 [cs.RO], Jan. 2023.

[23] Íñigo Elguea-Aguinaco, A. Serrano-Muñoz, D. Chrysostomou, I. Inziarte-Hidalgo, S. Bøgh, and N. Arana-Arexolaleiba, "A review on reinforcement learning for contact-rich robotic manipulation tasks," *Rob. Comput. Integr. Manuf.*, vol. 81, p. 102517, 2023.

[24] OpenAI, M. Andrychowicz, B. Baker, M. Chociej, R. Jozefowicz, B. McGrew, J. Pachocki, A. Petron, M. Plappert, G. Powell, A. Ray, J. Schneider, S. Sidor, J. Tobin, P. Welinder, L. Weng, and W. Zaremba, "Learning dexterous in-hand manipulation," arXiv:1808.00177 [cs.LG], Aug. 2018.

[25] B. Yang, P. E. Lancaster, S. S. Srinivasa, and J. R. Smith, "Benchmarking robot manipulation with the rubik's cube," *IEEE Robot. Autom. Lett.*, vol. 5, no. 2, pp. 2094–2099, 2020.

[26] J. D. Wason, "Robot raconteur® version 0.8: An updated communication system for robotics, automation, building control, and the internet of things," in *IEEE Int. Conf. Autom. Sci. Eng. (CASE)*, 2016, pp. 595–602.

[27] J. D., Wason, "Abb motion program exec," https://github.com/rpiRobotics/abb_motion_program_exec, 2022, [Accessed: 2023-02-28].

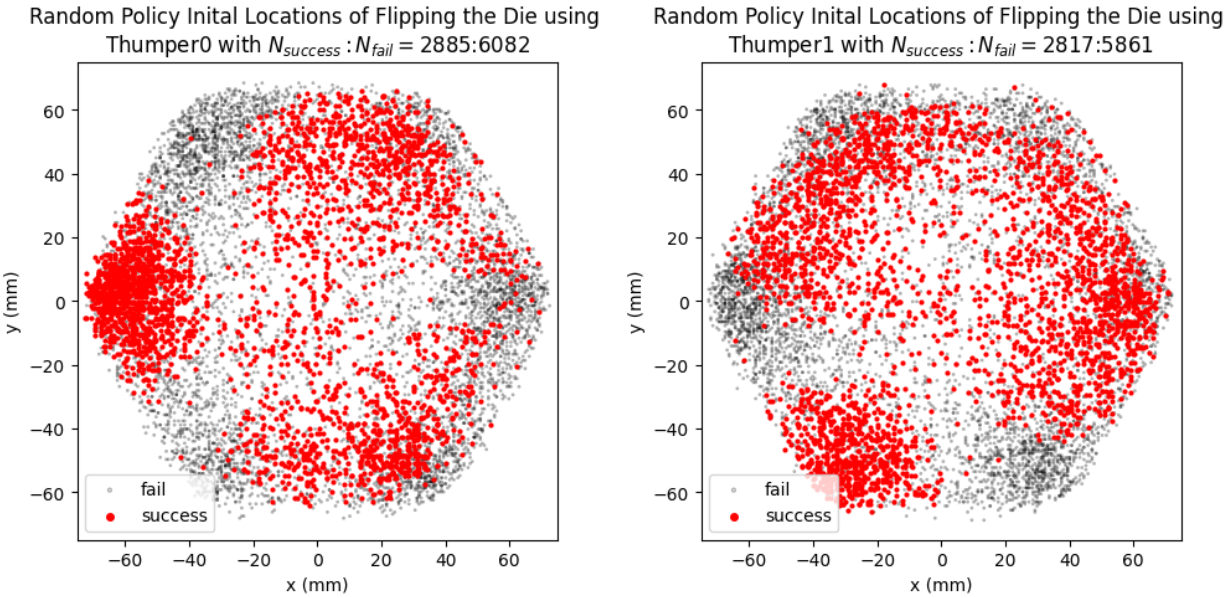
[28] J. D. Wason and W. Lawler, "General robotics toolbox," https://github.com/rpiRobotics/rpi_general_robotics_toolbox_py, 2018, [Accessed: 2023-02-28].

[29] R. A. Howard, *Dynamic programming and Markov processes*, 1st ed. Cambridge, MA, USA: Technology Press of Massachusetts Institute of Technology, 1960.

[30] J. A. Martín H, J. de Lope, and D. Maravall, "Robust high performance reinforcement learning through weighted k-nearest neighbors," *Neurocomputing*, vol. 74, no. 8, pp. 1251–1259, 2011.

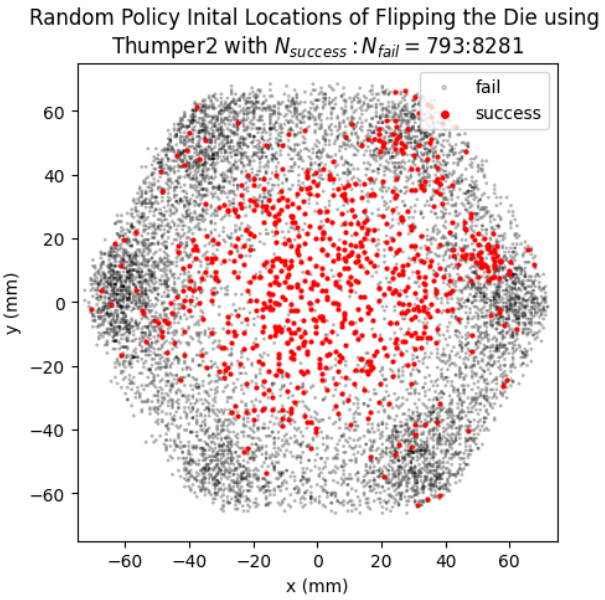
APPENDIX A
DIE FLIPPING EXPERIMENT FIGURES

A.1 Die Random Policy Experiment Data of Each Solenoid

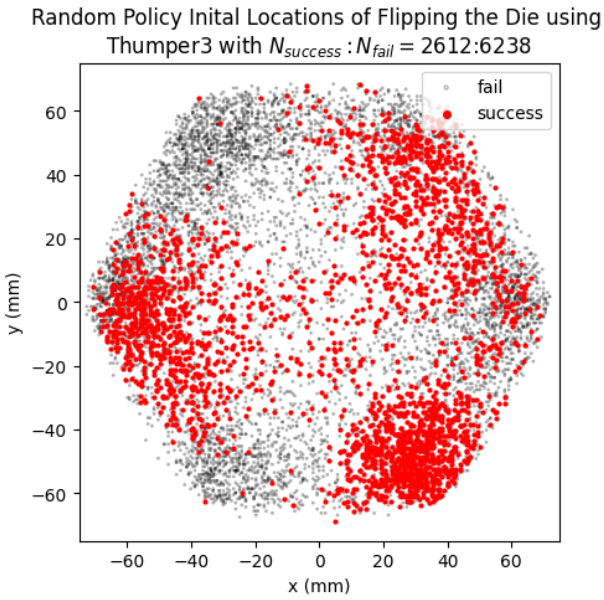


(a) (b)

Figure A.1: Flipping the Die using (a)Thumper0, (b)Thumper1

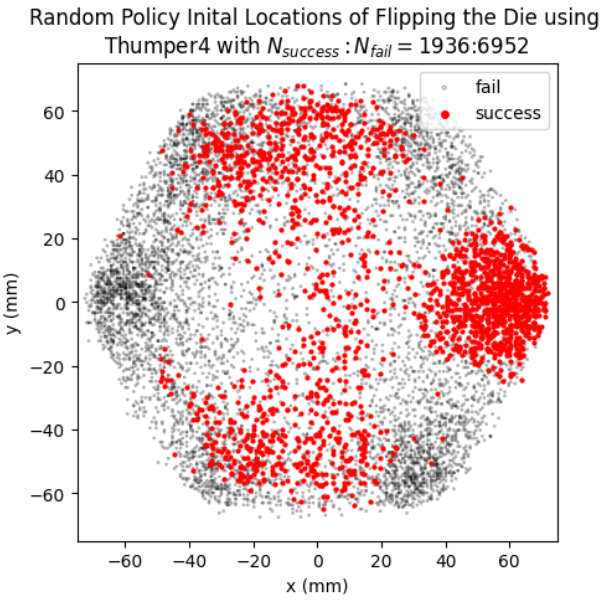


(c)

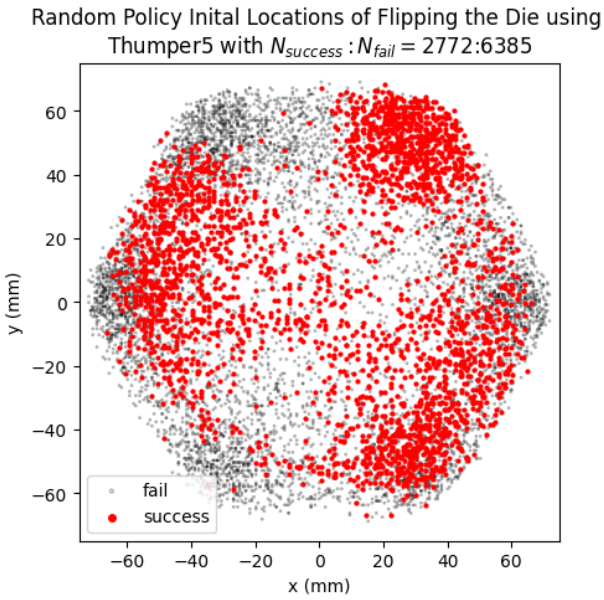


(d)

Figure A.1: Flipping the Die using (c)Thumper2, (d)Thumper3

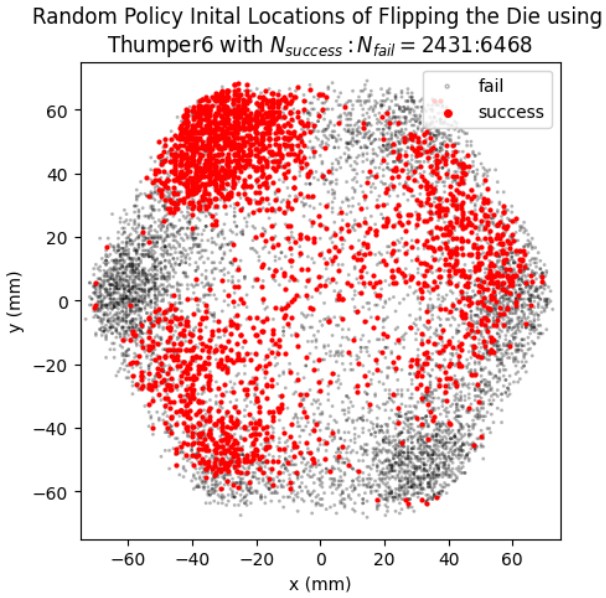


(e)



(f)

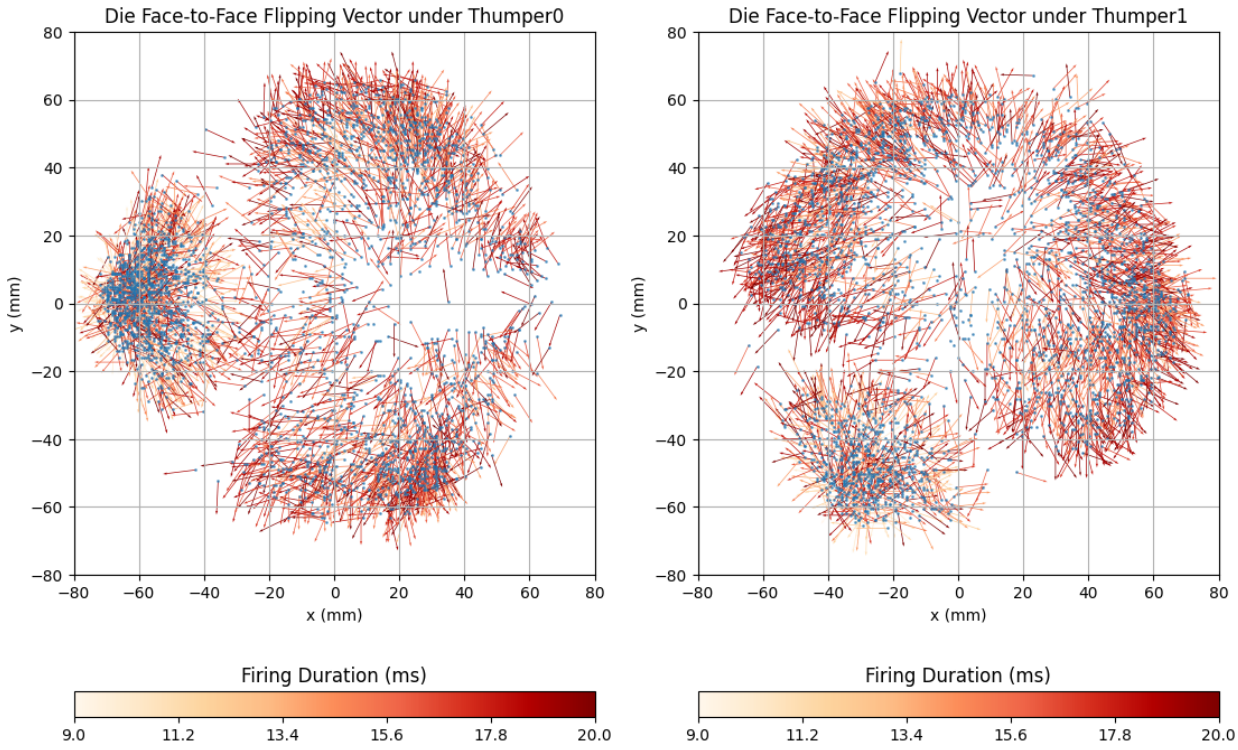
Figure A.1: Flipping the Die using (e)Thumper4, (f)Thumper5



(g)

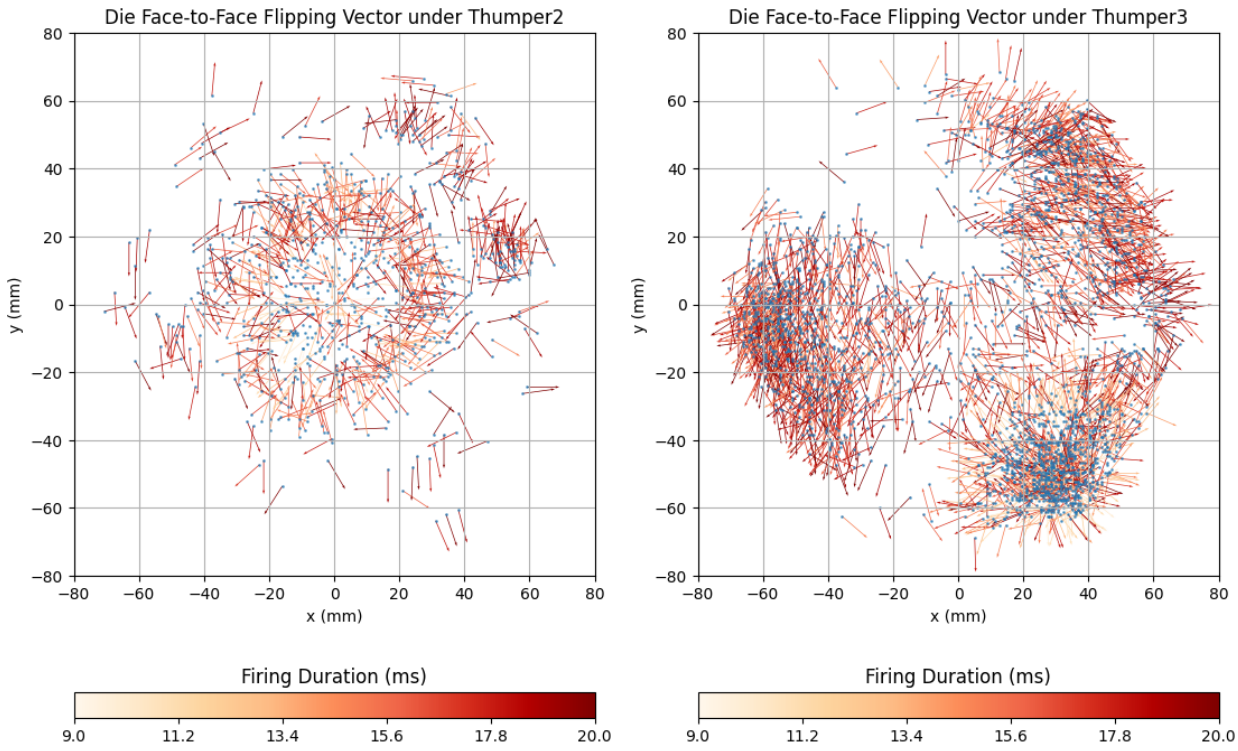
Figure A.1: Flipping the Die using (g)Thumper6

A.2 Die Random Policy Experiment Facet Transition Graphs of Each Solenoid



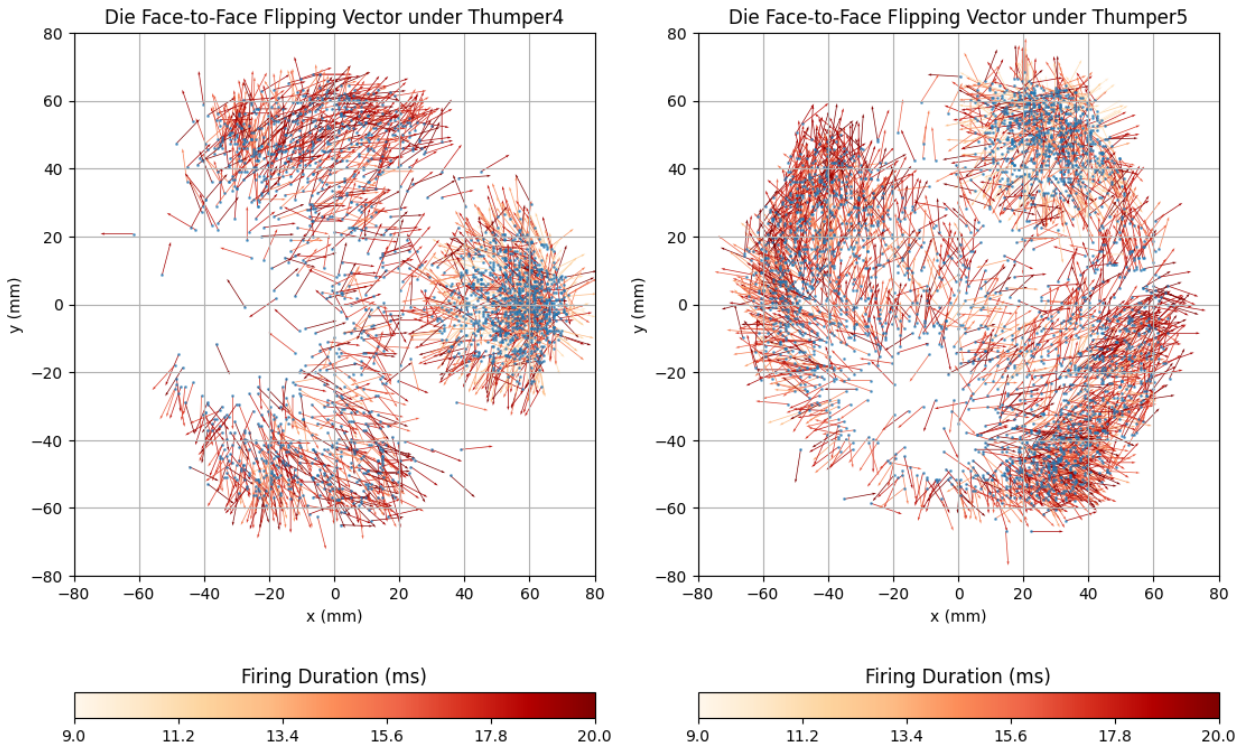
(a) (b)

Figure A.2: Flipping the Die using (a)Thumper0, (b)Thumper1



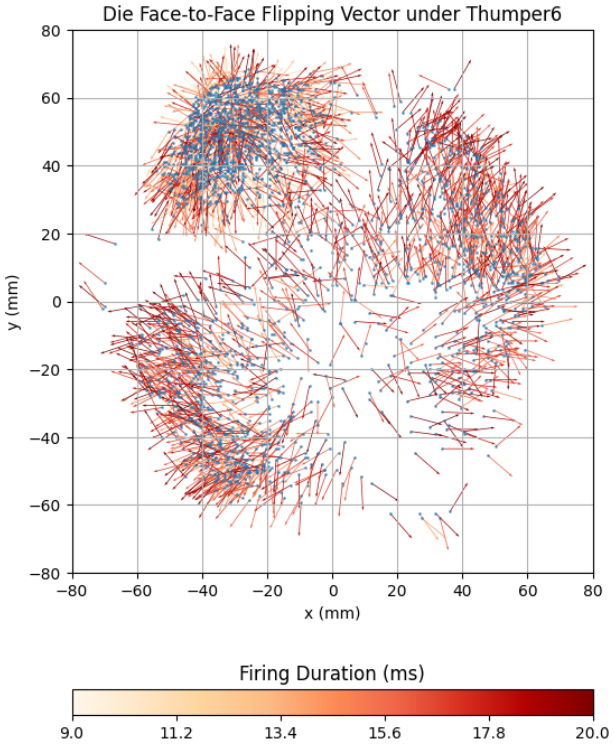
(c) (d)

Figure A.2: Flipping the Die using (c)Thumper2, (d)Thumper3



(e) (f)

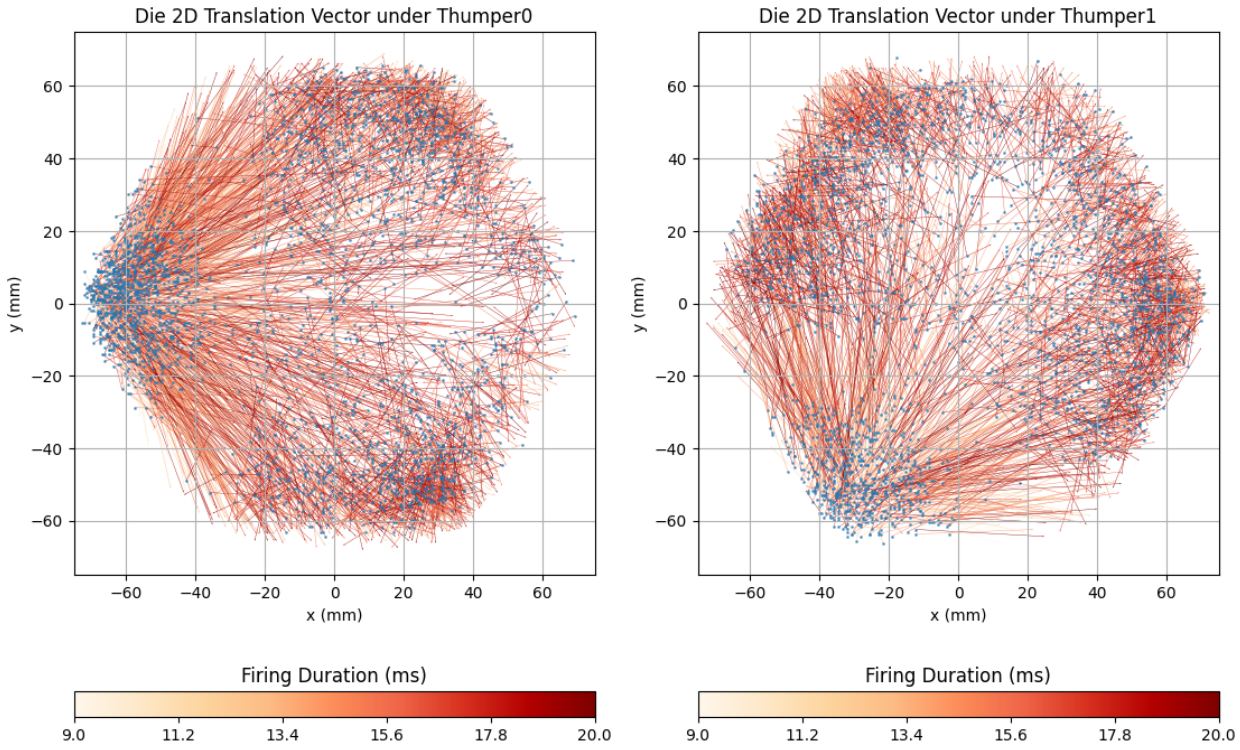
Figure A.2: Flipping the Die using (e)Thumper4, (f)Thumper5



(g)

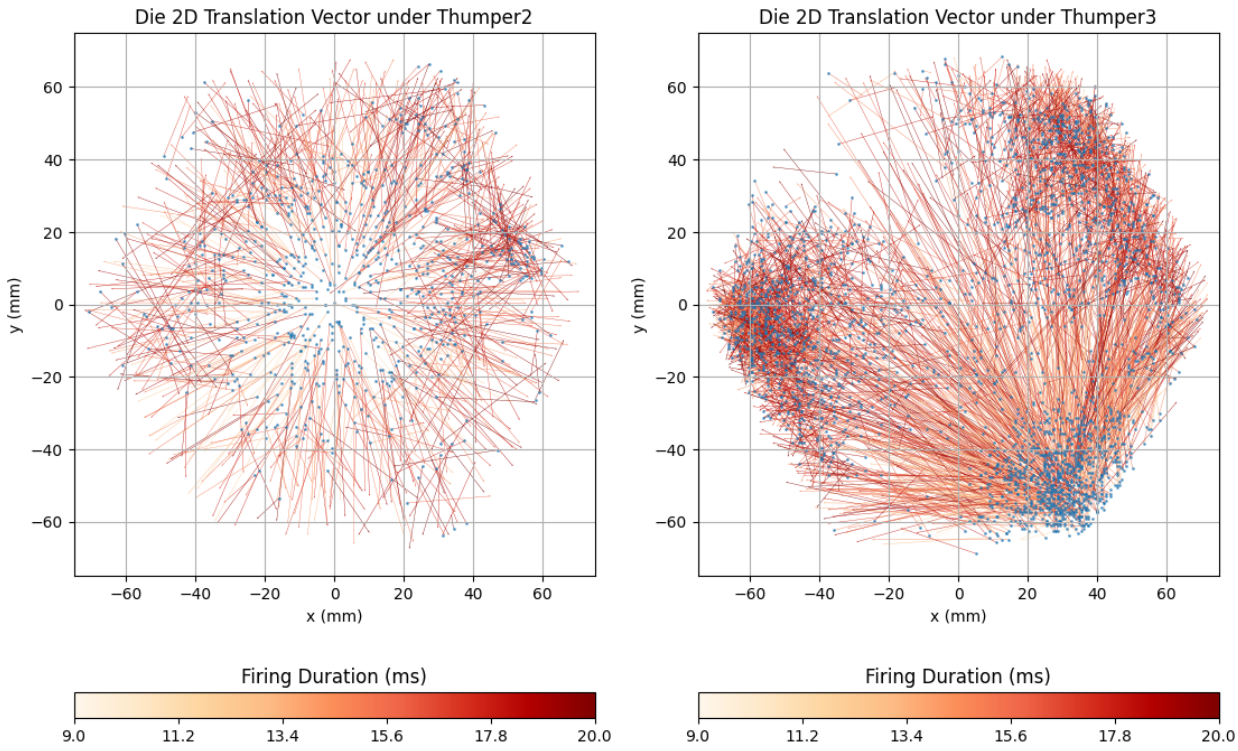
Figure A.2: Flipping the Die using (g)Thumper6

A.3 Die Random Policy Experiment 2D Transition Graphs of Each Solenoid



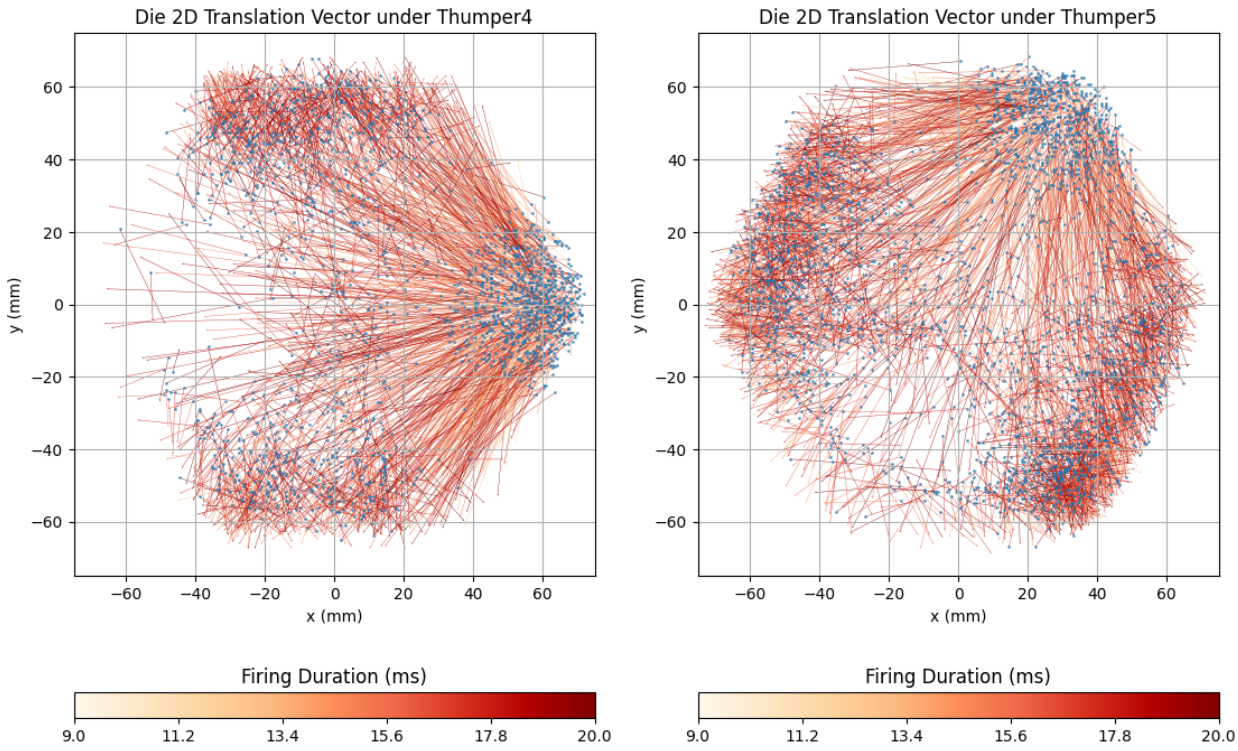
(a) (b)

Figure A.3: Flipping the Die using (a)Thumper0, (b)Thumper1



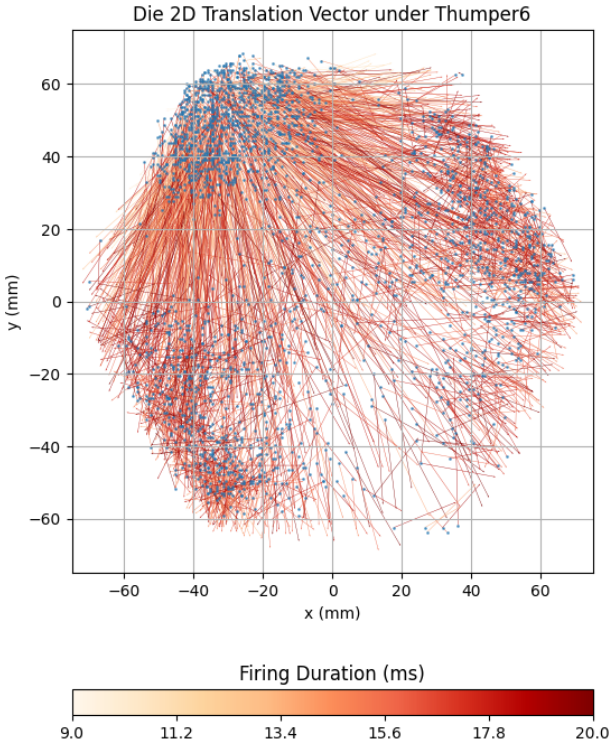
(c) (d)

Figure A.3: Flipping the Die using (c)Thumper2, (d)Thumper3



(e) (f)

Figure A.3: Flipping the Die using (e)Thumper4, (f)Thumper5

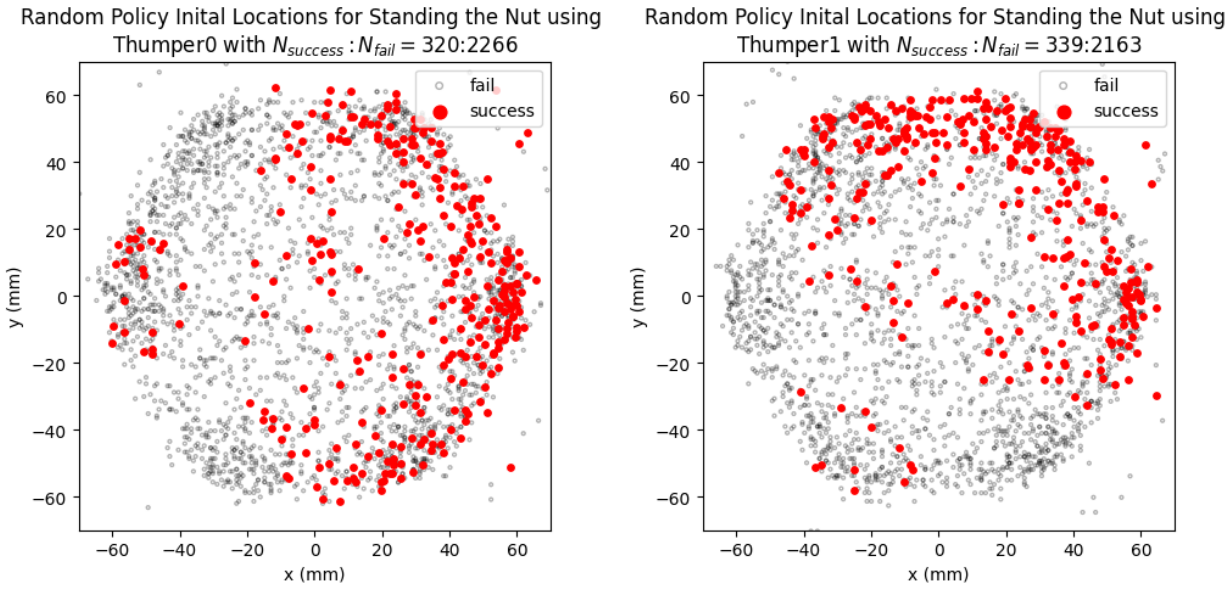


(g)

Figure A.3: Flipping the Die using (g)Thumper6

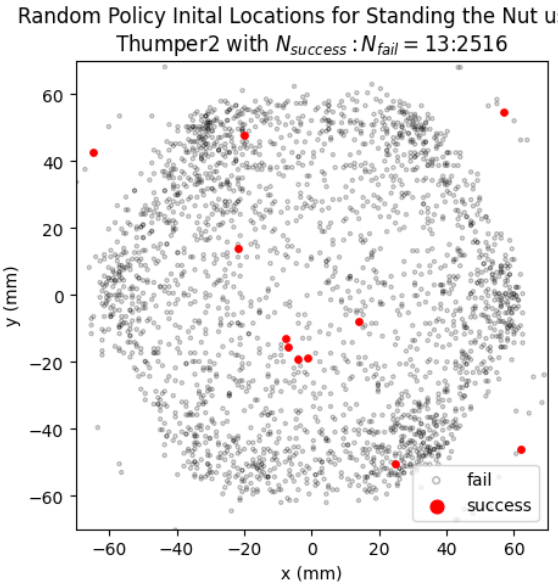
APPENDIX B
NUT STANDING EXPERIMENT FIGURES

B.1 Nut Random Policy Experiment Data with Fixed Impulse Durations

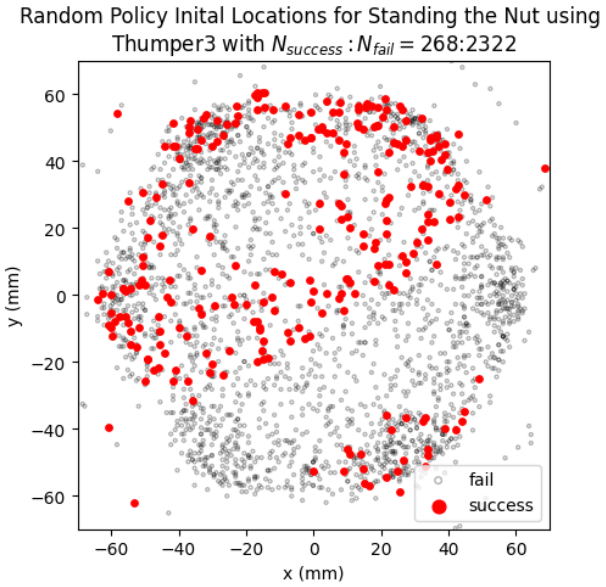


(a) (b)

Figure B.1: Standing the nut using (a)Thumper0, (b)Thumper1

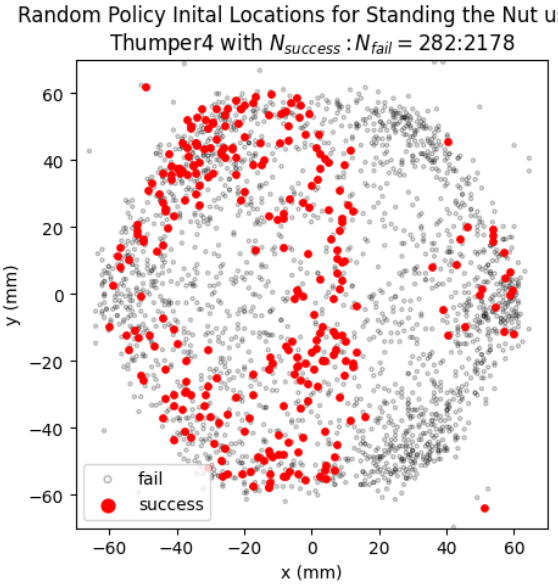


(c)

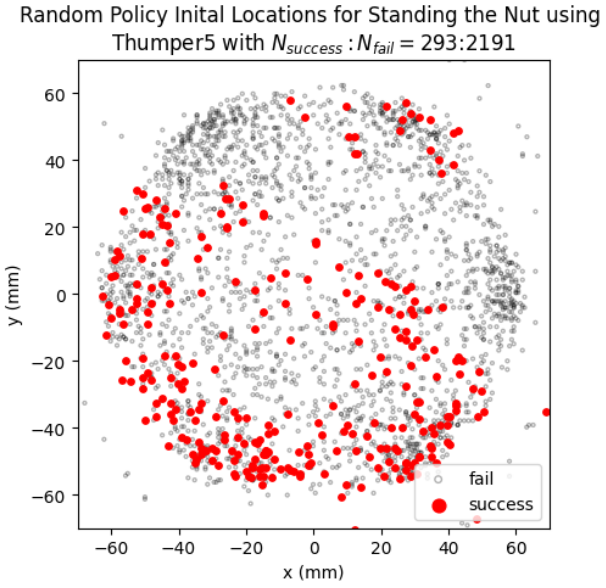


(d)

Figure B.1: Standing the nut using (c)Thumper2, (d)Thumper3

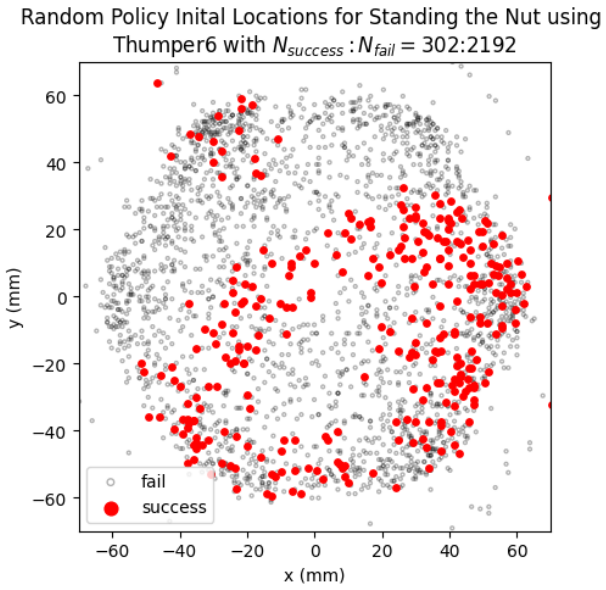


(e)



(f)

Figure B.1: Standing the nut using (e)Thumper4, (f)Thumper5



(g)

Figure B.1: Standing the nut using (g)Thumper6