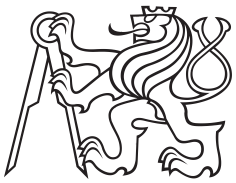
Master Thesis



Czech Technical University in Prague

F3

Faculty of Electrical Engineering
Department of Control Engineering

Autocompletion algorithm for simple trajectories

An algorithm that will automatically complete a simple demonstrated trajectory

TAN Wei Xin

Supervisor: Mgr. Radoslav Škoviera, Ph.D.
Supervisor–specialist: Mgr. Karla Štěpánová, Ph.D.
Field of study: Cybernetics and Robotics
Subfield: Cybernetics and Robotics
May 2019

MASTER'S THESIS ASSIGNMENT

I. Personal and study details

Student's name: **Tan Wei Xin** Personal ID number: **471886**

Faculty / Institute: **Faculty of Electrical Engineering**

Department / Institute: **Department of Control Engineering**

Study program: **Cybernetics and Robotics**

Branch of study: **Cybernetics and Robotics**

II. Master's thesis details

Master's thesis title in English:

Autocompletion algorithm for simple trajectories

Master's thesis title in Czech:

Algoritmus pro automatické doplňovaní jednoduchých trajektorií

Guidelines:

The goal of this thesis is to propose, implement, and test an algorithm that will automatically complete a simple demonstrated trajectory. The motivation for this algorithm comes from an assemly process aided by a robotic arm. For example, if a glue is to be applied to an object, the operator would simply start demonstrating where the glue should be applied. Instead of tracing the entire object the algorithm would recognize the trajectory pattern (e.g., wave-like) and the relation of the trajectory with respect to the object (e.g., 5mm inwards from the border of the object). The algorithm would then complete the trajectory, thus it can be used by the robot to apply the glue. The demonstrated trajectory should then be transferable onto other objects of the same shape.
The following steps should be carried out in order to complete the thesis:
1. Create or search for a suitable dataset containing trajectories and accompanying visual information.
2. Preprocess the trajectory data (e.g., filtering) and employ an existing trajectory classification algorithm to detect the trajectory pattern.
3. Preprocess the visual data and employ an existing object recognition algorithm to detect objects present on the scene and their properties (such as shape).
4. Develop an algorithm for autocompletion of the trajectory.
a. First, propose a simple path,
b. then, superimpose the demonstrated pattern on the path
5. Evaluate and demonstrate (i.e. in simulation or on a real robot) the proposed algorithm

Bibliography / sources:

[1] Thrun, Sebastian and Burgard, Wolfram and Fox, Dieter: "Probabilistic robotics.", MIT press, 2005.
[2] Berio, Daniel, Sylvain Calinon, and Frederic Fol Leymarie. "Generating Calligraphic Trajectories with Model Predictive Control." 2017.
[3] Faraway, Julian J., Matthew P. Reed, and Jing Wang. "Modelling three□dimensional trajectories by using Bézier curves with application to hand motion." Journal of the Royal Statistical Society: Series C (Applied Statistics) 2007
[4] Gallier, Jean, and Jean H. Gallier. Curves and surfaces in geometric modeling: theory and algorithms. Morgan Kaufmann, 2000.
[5] Duda et al. "Pattern classification." NY, USA 2001.

© ČVUT v Praze, Design: ČVUT v Praze, VIC

Name and workplace of master's thesis supervisor:

Mgr. Radoslav Škoviera, Ph.D., Robotic Perception, CIIRC

Name and workplace of second master's thesis supervisor or consultant:

Mgr. Karla Štěpánová, Ph.D., Robotic Perception, CIIRC

Date of master's thesis assignment: **28.01.2019** Deadline for master's thesis submission: **24.05.2019**

Assignment valid until:
by the end of summer semester 2019/2020

Mgr. Radoslav Škoviera, Ph.D.	prof. Ing. Michael Šebek, DrSc.	prof. Ing. Pavel Ripka, CSc.
Supervisor's signature	Head of department's signature	Dean's signature

III. Assignment receipt

The student acknowledges that the master's thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the master's thesis, the author must state the names of consultants and include a list of references.

.	
Date of assignment receipt	Student's signature

© ČVUT v Praze, Design: ČVUT v Praze, VIC

Acknowledgements

First and foremost, I would like to thank my supervisors, Mgr. Radoslav Škoviera, Ph.D. and Mgr. Karla Štěpánová, Ph.D., for providing me with the opportunity to pursue this topic. Without their prior work on the Asus XTION PRO 3D sensor, the HTC Vive virtual reality system and the trajectory dataset, this thesis would have not gotten to the level it is at right now. Their guidance has also been important in allowing me to finish this piece of work in a timely manner.

Besides that, I would also like to thank Markus '斯斯' Stroot for his ingenuity and his suggestions related to the field of computer vision – without him this thesis may have ended up stuck forever on the image processing part. Many thanks also goes Felix '菲菲' Staudigl for introducing me to the wonders of feature selection and for his infectious optimism which has kept me going even when all hope seemed lost. On top of that is Flavio '弗弗' Kreiliger who I have to thank for being one of the best workmates I've ever had as well as Jasna Petric for always being a fun person to work with.

I would also like to thank 熊貓 for leaving her comfort zone and staying by me for so long as well as for being willing to support me however she can. Finally, last but not least, many thanks go to my family for placing so much of their trust in me. Without their advice, love and support, I would have never made it as far as I have today.

Declaration

I declare that the work presented in this thesis was developed independently and that I have listed all relevant sources of information used within according to the guidelines on observing ethical principles in preparation of university theses.

Prague, 23. May 2019

.....

TAN Wei Xin

Abstract

Motion planning is one of the core problems that is being studied extensively by robotics researchers in the present day. Among the many techniques available, planning via automatic path/trajectory generation is one of the most widely used approaches. This thesis has implemented a system using tools from computer vision, computer graphics and supervised machine learning which can 'autocomplete' a user demonstrated trajectory segment on differently shaped blocks. This means that the final trajectory generated will be based on the shape of the block with the user's demonstration superimposed on it. The aim is for the trajectory to be utilized in planning the motions of an industrial robot. In the process of developing this system, this thesis provides a comprehensive review of the subject fields utilized and covers the basic intuition behind the algorithms used in the system.

The final results of this thesis show that it is possible to automatically generate smooth and continuous trajectories that are non-photorealistic using information from a human made trajectory segment. Although the system is functional, it should be considered as a proof of concept rather than as an industrial level implementation. There is much improvement to be made to this thesis' system before it can be considered fit enough to be deployed in an industrial setting.

Keywords: trajecory generation, digital image processing, shape recognition, computer graphics, non-photorealistic rendering, machine learning, trajectory classification

Supervisor: Mgr. Radoslav Škoviera, Ph.D.
B-611a
Jugoslavskych partyzanu 3
160 00 Praha 6

Contents

Figures

Tables

Chapter 1

Introduction

In the modern era, people have become increasingly reliant on automation and robotics which means that there is a need for these machines to have a wide variety of capabilities at their disposal. The number of human robot interactions have also been on the rise which means the design of future robotic systems will have to be more accommodating towards humans. In addition, there is also a need for robots that can perform highly monotonous work such as applying glue with a glue gun in order to free up people's time to tackle more difficult tasks.

1.1 Motivations

The planning of robot motion is a very large field. It has many existing solutions[LaV06] and yet, is still filled with challenges especially in an industrial setting[KC]. The classical approach to motion planning for industrial robots is designing joint position, velocity or even torque profiles using programming by demonstration (more well known as imitation learning)[BCDS08]. A human instructor moves the industrial robot in a desired manner and stores the corresponding actuator information thus 'teaching' the robot in a kinesthetic manner. This approach is especially ideal for repetitive tasks. If some sort of visual system for object recognition is included, the robot can behave flexibly under many different conditions.

The downside of imitation learning is that it is usually difficult to scale scale a task and/or transfer it to a different robotic setup. This means that every single case must be considered and that the human instructor must guide the robot through the entire process perfectly. Hence, human instructors have to be experts at the task being taught. Furthermore, imitation learning also requires them to come into physical contact with the robot. This type of human-robot interaction is not without its risks[KC05]. As such, all of this makes it more difficult for the robot to learn new processes. Due to these issues, industry has begun to utilize a number of different technologies such as virtual[FR99] and augmented[OGL08] reality. This is expected to reduce physical human-robot interaction as well as to reduce the reliance on expert human instructors.

Since the user is not expected to be an expert and human-robot contact should be minimized, there is a need for the robot to deduce the user's intentions. This means that not only should the robot determine what its operator would like it to do but also, that the robot should filter any 'distorted' commands. This is due to the fact that non-expert users should only be able to imagine a task in being performed in an 'ideal' manner. They should not have the skill to replicate their own thoughts exactly. This ability to 'autocomplete' a human operator's intentions would make an industrial robot more adaptive and useful since it would require less training and supervision.

1.2 Goals

This thesis aims to propose a system which will 'autocomplete' trajectories based on a human demonstrated one. This will allow an industrial robot with a glue gun attached to its end-effector to use these trajectories for path planning without the need for physical human-robot interaction. More specifically, with the use of a 3D sensor and a virtual reality system, the goals of this thesis are the following.

1. Detect the shape of a target using the 3D sensor
2. Classify the type of trajectories recorded by the virtual reality system
3. Determine the 'ideal'/non-distorted version of the trajectory classified in 2.
4. Generate a new trajectory that contains the trajectory found in 3. superimposed over another trajectory based on the shape detected in 1.

In the case of 2., the trajectories should be identified without needing the user to 'complete' it. This means that drawing any pattern along the path of a circle should only require one segment of the pattern for the system to 'autocomplete' the rest of trajectory. On top of that, the pattern need not be perfect as 3. should filter out any distortions e.g. a wave pattern with inconsistent periods and amplitudes should become a sine wave with constant periods and amplitudes.

1.3 Contribution of thesis

The main contribution of this thesis is a method for procedurally generating trajectories that contains complex curves superimposed on another simpler curve. The complex one is based on user demonstration while the simpler one is based on shape detection. This includes:

1. a computer vision algorithm that preprocesses visual as well as depth images and determines the shape/edges of the target from them
2. a supervised machine learning algorithm that preprocesses 3-dimensional trajectory data consisting of xyz coordinates and classifies them according to type

3. a computer graphics algorithm that generates 'ideal' versions of trajectories using mathematical equations and superimposes them onto another trajectory

This thesis is structured as follows. Chapter 1 lists the motivations for this thesis' topic and what this thesis hopes to achieve. Chapter 2 explores the relevant work in the areas that this thesis employs. Chapter 3 elaborates upon the methods used by this thesis to achieve its goals. Chapter 4 goes on to present the results of implemented system on the dataset collected for this thesis. Chapter 5 examines in more detail the system and results of this thesis. Chapter 6 analyzes the aspects of this thesis that could be developed further. Appendix A provides the pseudo-code that made this thesis' results possible. Appendix B contains a glossary of the acronyms and symbols used while Appendix C lists the references utilized in this thesis.

Chapter 2

Related Work

The work presented in this thesis is closely related to 3 distinct subfields. These are vision based trajectory generation, non-photorealistic rendering (NPR) and trajectory classification. All three have been researched extensively and there have been a number of works on NPR robotics[LMPD15][SSGG19]. These works have utilized ideas from all 3 subfields although mainly for painting/picture reproduction purposes. This thesis, on the other hand, is more focused on path/trajectory planning in an industrial setting. As such, this thesis draws upon many previously used approaches and integrates them to develop the final system.

2.1 Vision based trajectory generation

In robotics, trajectories are smooth functions of time which specify the robot's and/or its end-effector's position[LP17]. Very often, trajectories are designed to satisfy constraints such as positional waypoints or workspace/velocity/acceleration/torque limits. Trajectory generation also has to take into account obstacle avoidance. Since trajectories should be smooth, they are usually approximated by polynomial functions.

For useful trajectories to be generated, at least part of a robot's workspace must usually be known. This information can be obtained in many ways. One of the most common is through the use of vision systems. In the case of [ANSL03], their system was developed to automate the cutting of embroidered material. Using a digital camera equipped with an ambient light filter and a laser diode, they were able to detect a difference in light intensity where the seam of the embroidery occurred. They used this information to generate a trajectory which they smoothed using either a moving average filter or a Recursive least square filter. Their approach, however, was very specific to their task. The edges their system were detecting were of non-negligible thickness which is why their use of a laser diode was effective. Also, their camera was mounted close to the end-effector of the robot unlike in this thesis' case. The vision system used in this thesis was stationary.

Another related work is [GVPS12] where the system was developed for spray painting. In their case, a barrier sensor was used to obtain the edge and surface

primitives/lines of the object (the algorithm behind this process was not elaborated on). Then, an Eulerian trajectory was determined so that every primitive/line was visited once. Although their approach is particularly useful for generating trajectories with no overlap, it presumes that the object edges are already well known. Also, image processing was not discussed which is not the case for this thesis.

[MPW14] is another work of interest. They utilize image edge detection to generate a trajectory for applying sealant to engine blocks. Here, the authors use a single digital camera and apply Canny edge detection to the images that they acquire. Using integral projection functions, they then proceed to determine the engine block's outermost edge. After that, they filter it to remove jumps/skips between pixels before using it as a trajectory. This thesis uses a similar methodology for shape recognition although there was not as much control over experimental area's lighting.

Finally, there is [SL17] which was developed for drawing pictures using a differential drive robot. They also use Canny edge detection as the basis of their trajectory. They go further by approximating these edges with cubic B-splines which results in continuous joint positions, velocities, accelerations and even jerk. Although their work did not utilize other image processing methods besides Canny edge detection, their use of B-splines demonstrated its usefulness for robot trajectory generation.

2.2 Non-photorealistic rendering (NPR)

In traditional computer graphics, the aim of rendering is usually photorealism i.e. the reproduction of an image or scene such that it resembles the output of an actual camera. NPR, on the other hand, is more subjective in that it aims to create an artistic representation[SS02]. Although NPR encompasses many techniques ranging from classical graphics rendering to sketch-based modelling, this thesis will be primarily exploring the statistical and mathematical modelling approaches to curve synthesis – the creation of output curves that are similar but contain information that is not present in their inputs.

The multiresolution approach to curve synthesis is used by [FS94], where the idea of decomposing curves into multiple levels of detail using wavelets was introduced. [BSS07] improved and extended it further to cover different sketch styles. Curve decomposition is done by using two matrices known as analysis filters. One downsamples/coarsens the input curve while the other preserves the features removed by the previous matrix. Curve synthesis is then performed by utilizing another 2 matrices known as synthesis filters. They reverse the effects of the analysis filters. This process is done iteratively and generates what the authors call a 'filter bank'. The 'bank' contains filter matrices for every level of detail. Even though this method was considered, it was determined to be more complicated than necessary for the purpose of this thesis. Nevertheless, it is

possible that their approach may be quicker for longer and/or more complicated curves. Further studies will have to be done to ascertain this.

Another approach to curve synthesis has been explored by [HOCS02]. The authors also treat curves as decomposable into multiple levels of details like [FS94]. Feature extraction and reconstruction, however, is performed using local transformations. This synthesis of curve analogies is done by first finding the rigid transformation between curve A (base) and curve B (input). This transformation is then applied to curve A' (style) and the result is split into multiple segments. A cost function, $E(B')$, that characterizes the difference between:

- the shape of A' and B',
- the shape relationship of A to A' and that of B to B',
- the relative position and orientation relationship of A to A' and that of B to B',

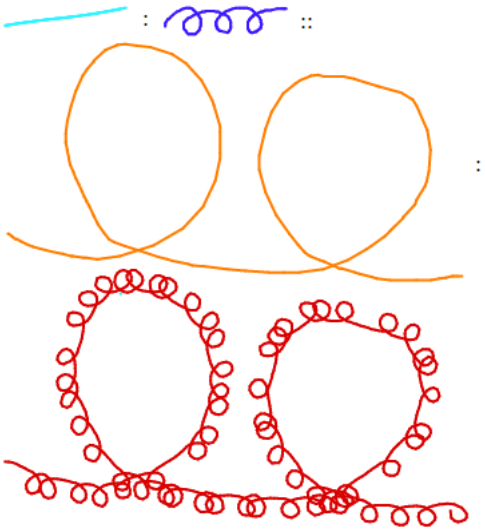


Figure 2.1: Example of curve results from [HOCS02], $A : B :: A' : B'$

is then defined. For every segment, a rigid transformation around the center of mass of the B' segment that minimizes $E(B')$ is computed. This thesis' approach is similar and has the advantage of allowing for patterns that had no overlap unlike in that seen in Figure 2.1.

There is also a statistical approach to curve synthesis as seen in the works of [KMM+02]. The authors use a Markov random field (MRF) to synthesize curves based on several user provided example curves. Another one is [SD04] where Hidden Markov Models (HMM) are used to synthesize appropriate curves onto new illustrations. In the case of [KMM+02], user provided curves are stored as a spline and vectors perpendicular to the spline. These are used to characterize the curve's features. The MRF is then used to map similarities of a curve segment

to the probability of that segment being added to the new, synthesized curve. For [SD04], their HMM is trained on multiple sets of detailed and non-detailed 'control' curves. This is to determine the correspondence between them. From that, detailed curves are then synthesized from non-detailed user drawn curves. The key to their approaches' success is the use of large training datasets which this thesis did not have. Also, the method used in this thesis only requires the user to demonstrate part of the desired trajectory. The system being developed by this thesis is not aiming to replicate the user's input completely but to generate an 'ideal'/non-distorted version of it. As such, statistical modelling was used not in the curve synthesis portion of the thesis. Instead, it was used in the trajectory classification portion which is elaborated in Section 3.4.1.

2.3 Trajectory classification

Much of the work on trajectory classification is centered around 2-dimensional cases. 2D trajectories are usually categorized based on spatial and temporal features i.e. position, velocity etc. Many of those approaches, however, can be extended to the 3-dimensional case. Though in 3D, the importance and relevance of the 2D features need to be re-evaluated. Also, new features need to be determined in 3D to aid classification. This is what happens in [BKS06] where the authors explore classifying 3D trajectories created from the Australian Sign Language (ASL). By developing affine-invariant features from trajectory coordinates, the authors were able to use HMMs to recognize ASL words not oriented in the same manner i.e. the signs for the same word could be rotated or stretched. These new features are the centroid distance function (CDF) and curvature scale space (CSS). Both were adopted by this thesis in order to improve classification results. Unlike [BKS06] though, the trajectory datasets collected for this thesis were constrained to the same area of the coordinate space since they were all performed on the surface of the targets. This meant that the xyz coordinates could be used together with the CDF and CSS for training the classifier used in this thesis.

TraClass[LHLG08], on the other hand, uses region and movement pattern clustering to tacklet the 2D trajectory classification problem. This framework allowed the authors to approach trajectory classification in a hierarchical manner. It sorted the detected features and gave preference to higher level ones for classification. The procedure behind TraClass can be described in 4 general steps

1. Each trajectory is split into line segments based on their change in direction using the minimum description length principle[GMP05]. Segments from similar trajectories as well as those from similar classes end up grouped together.

2. The groupings are used to identify regional clusters via a grid structure. A regional cluster is one where there are relatively many trajectories of one class in comparison to other classes. This is done recursively with 1. to ensure as many clusters as possible are found.

3. The groupings are also utilized to establish movement pattern clusters within a user-specified neighborhood. Each group must contain segments from the same class otherwise they are treated as noise. This is also done recursively with 1. to ensure as many clusters as possible are found.

4. Both clusters based on region and on movement pattern are merged for class identification. Regional clusters are given higher priority compared to movement pattern clusters.

TraClass' approach was effective at performing classification on 'top-down' trajectory data. Nevertheless, it was not suitable for this thesis' dataset as most of the 3D trajectories recorded did not have significant regional differences. Movement pattern based clustering may have been suitable but seemed unlikely. This is because TraClass' experiments were shown to work well on trajectories generated by different sources (e.g. deer vs cow) whereas the trajectories for this thesis were performed by one subject multiple times.

Another work dealing with trajectory classification is [PMF08]. The use of Support Vector Machines (SVM) for abnormal trajectory detection is examined by the authors. Here, they introduce an improved approach to tuning SVM parameters. SVMs have been previously used for trajectory classification and is also one of the classifiers used by this thesis. The authors' method of selecting optimal parameters, however, is more applicable for outlier detection. It is less relevant for this thesis compared to multiclass identification as there were more than 2 types of trajectories present in this thesis' dataset.

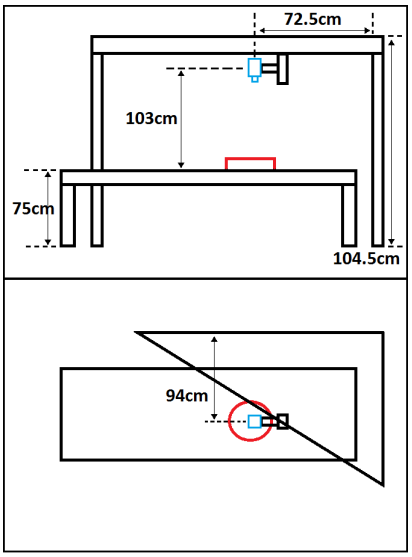
Similar to [BKS06] is the work of [FD09]. It addresses the classification of hand movement and orientation. In order to label analogous motions correctly, the authors of [FD09] did not take into consideration temporal information of the trajectory (otherwise fast and slow motions would be classified differently). They also normalized trajectory length so that it would not affect classification. Also, they smoothed and divided the trajectory into discrete parts. Then, they use spherical and cylindrical coordinates to compute orientation features. After building a learning table from those features, they then utilize it to develop a classifier based on Bayes' theorem. They also use Shannon entropy to further improve results. Although similar preprocessing was performed (such as data smoothing and resampling), it was determined that their method of calculating orientation features was not appropriate for this thesis. This is because trajectory coordinates in the dataset corresponded to the tip of a glue gun instead of the position and orientation of a human hand. Moreover, this thesis' dataset came with its own orientation profiles but it was determined that they varied too much to be useful for training the final classifier.

Chapter 3

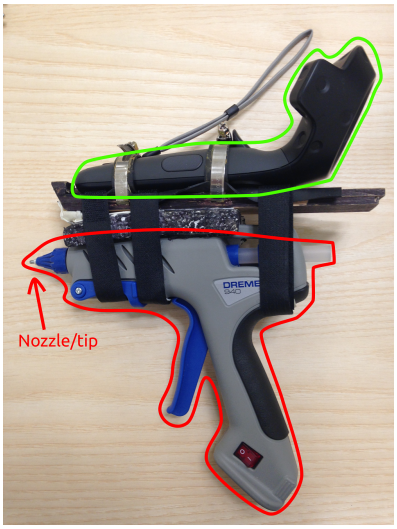
Materials and Methods

3.1 Dataset

3.1.1 Data collection



(a) : Diagram of data collection setup



(b) : HTC Vive controller (green) strapped to glue gun (red)

Figure 3.1: Setup of devices

The spatial and visual data used in this thesis was obtained with the use of a HTC Vive virtual reality system paired with an Asus XTION PRO 3D sensor. As seen in Figure 3.1a, the camera (light blue) was mounted over a target (red, left) on a table. This produced RGB and depth images as seen in Figure 3.2. The checkerboard pattern in the background was used for calibrating the HTC Vive's controller. The controller (green) was attached to a glue gun (red, right) as seen in Figure 3.1b. The position of the glue gun's nozzle was determined using a known transformation matrix on the controller's internal position. All of

the data collected was saved in rosbag files which this thesis utilized to develop the final system.[1]

3.1.2 Data types

```
path:        66_SmallCircle_line_ontop.bag
version:     2.0
duration:    24.0s
start:       Jan 28 2019 12:13:44.25 (1548674024.25)
end:         Jan 28 2019 12:14:08.23 (1548674048.23)
size:        874.6 MB
messages:    36916
compression: none [718/718 chunks]
types:       geometry_msgs/PoseStamped    [d3812c3cbc69362b77dc0b19b345f8f5]
             sensor_msgs/CameraInfo       [c9a58c1b0b154e0e6da7578cb991d214]
             sensor_msgs/CompressedImage  [8f7a12909da2c9d3332d540a0977563f]
             sensor_msgs/Image            [060021388200f6f0f447d0fcd9c64743]
             tf2_msgs/TFMessage           [94810edda583a504dfda3829e70d7eec]
topics:      /XTION3/camera/depth/camera_info                    717 msgs    :
             sensor_msgs/CameraInfo
             /XTION3/camera/depth/image_rect/                    717 msgs    :
             sensor_msgs/Image
             /XTION3/camera/rgb/camera_info                      715 msgs    :
             sensor_msgs/CameraInfo
             /XTION3/camera/rgb/image_rect_color/compressed      716 msgs    :
             sensor_msgs/CompressedImage
             /controller1                                       1440 msgs    :
             geometry_msgs/PoseStamped
             /end_point                                         1440 msgs    :
             geometry_msgs/PoseStamped
             /tf                                                7481 msgs    :
             tf2_msgs/TFMessage
```

Listing 3.1: Dataset as seen using rosbag info

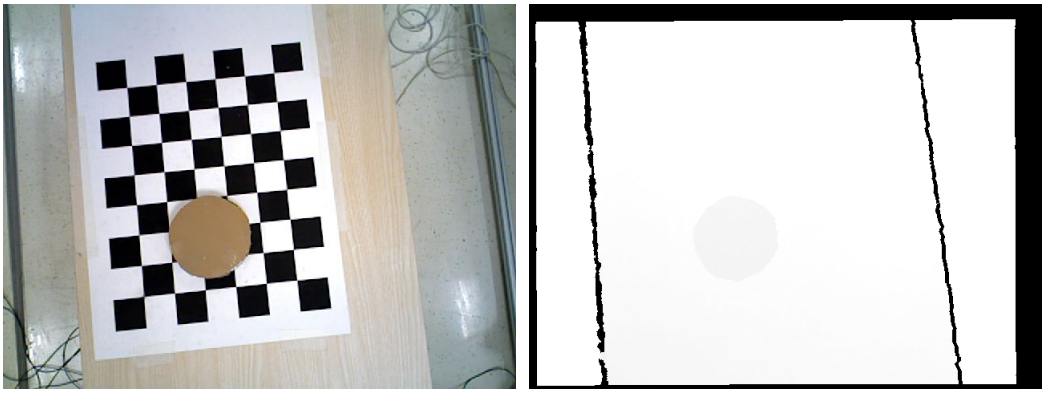
Listing 3.1 is an example of the contents in the rosbag files obtained from Section 3.1.1. Of primary importance are the topics

- /XTION3/camera/depth/image_rect/ which contains ROS messages related to the depth image being captured
- /XTION3/camera/rgb/image_rect_color/compressed which contains ROS messages related to the RGB image being captured
- /end_point which contains ROS messages related to the pose of the glue gun's nozzle

Unlike the pose information, only the first message from the depth and RGB topics was required. This is due to the assumption that the target would be static. Furthermore, it was assumed that there was nothing between the target and the 3D sensor that would obscure the target. This made shape identification easier as seen in Figure 3.2.

In comparison, every message of the glue gun's nozzle's pose information was required. Four types of user input trajectories were recorded – straight, dashed, low and high frequency wave trajectories. Each rosbag file containing 3 'laps' of each trajectory on six differently shaped targets.

[1] Rosbags are the primary mechanism in Robot Operating System(ROS) for data logging, http://wiki.ros.org/Bags



(a) : Raw RGB image for circular block

(b) : Raw depth image for circular block

Figure 3.2: Example of unprocessed visual information from dataset

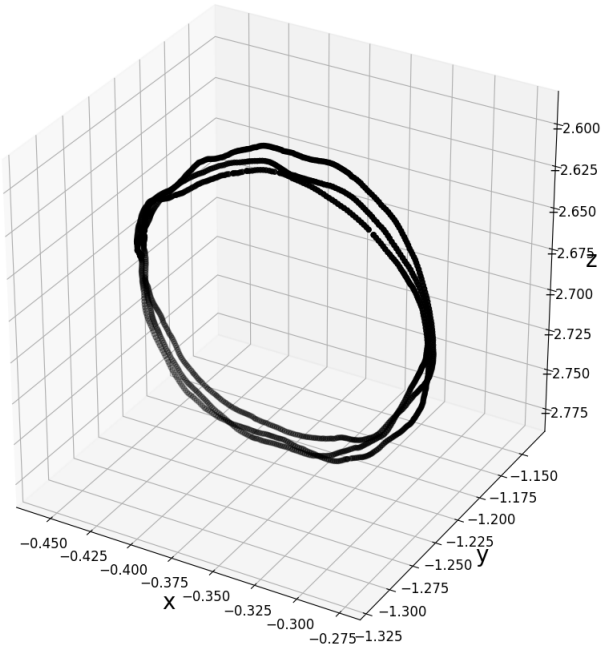


Figure 3.3: Example of unprocessed trajectory data from dataset

3.1.3 Trajectory segmentation

Trajectory type	Number of training samples	Number of testing samples	Total number of samples
Straight	29	29	58
Dashed	45	45	90
Low frequency wave	28	28	56
High frequency wave	54	54	108

Table 3.1: Number of trajectory samples after segmenting originals

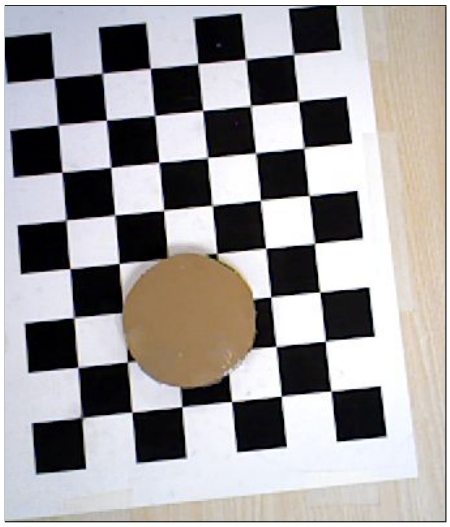
In total, there were 6 of each input trajectory type which gave a total of 24. If each were to be considered a single training sample, then the dataset would definitely be considered too small[Alp09]. As the assumption was that users would only need to demonstrate a part of the trajectory and the system should predict the rest, it made sense to split up the 24 trajectories into smaller segments. This should not affect classification as the amount of features that would matter should still be obtainable. As such, the number of trajectory instances was increased by segmenting the original user made trajectories by length. It was empirically determined that about 0.25m of a trajectory contained a sufficient number of features to determine trajectory type. This allowed for 312 trajectory segments/instances to be obtained from the original 24. The trajectories were then split into training and testing sets, half for each set with the ratio of classes in both sets being equal. The number of samples from each class can be seen in Table 3.1. Some classes have more samples due to the fact that some trajectory types, like the dashed and high frequency wave, were longer. This is because trajectory length was computed using the euclidean distance between each coordinate point.

3.2 Vision based trajectory generation

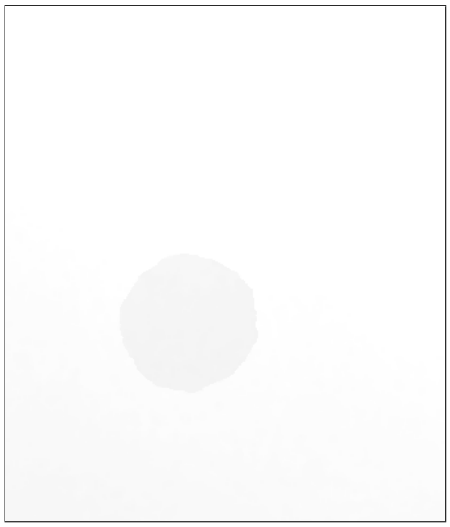
From the data seen in Section 3.1.2, it was clear some method was required to extract the shape of the target. In order to do so, a number of digital image processing techniques were used. These approaches and the basic intuition behind them are detailed below.

3.2.1 Not a Numbers (NaNs) in digital image processing

Since digital image processing methods are fundamentally the manipulation of numbers, it would be problematic if pixels in an image contained NaNs. This was the case as seen in Figure 3.2b where the black pixels mean that the Asus XTION PRO's depth sensor failed to assign a numeric value to them. Thus, although trivial, image cropping was crucial in making sure these pixels were removed from



(a) : Cropped RGB image for circular block

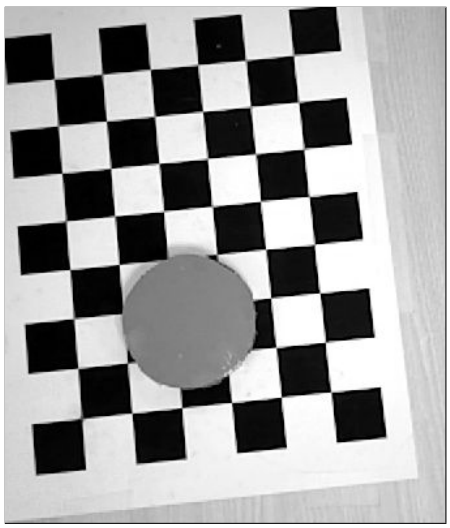


(b) : Cropped depth image with no NaNs for circular block

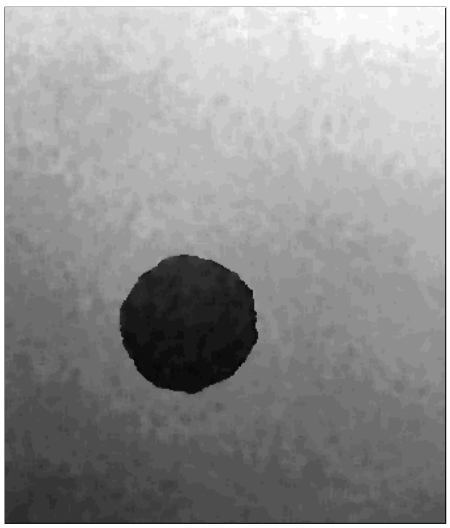
Figure 3.4: Example of cropped images with no NaN pixels

the image. This also meant cropping Figure 3.2a in the same manner to ensure pixel correspondence. Furthermore, a simple filter was applied to any NaN pixels where they were assigned the mean value of their 8 neighboring pixels. If other pixels in the neighborhood also had NaN values, those pixels were ignored in the computation of the mean.

3.2.2 Grayscale images



(a) : Grayscale conversion of RGB image



(b) : Grayscale conversion of depth image

Figure 3.5: Example of converting RGB and depth images to grayscale

Once NaNs were removed from the images, it became possible to apply further digital image processing methods to them. To determine the shape of the target, there was a need to detect the difference in magnitude of pixel intensities which should occur at the edges of a target[WR00]. This, however, is applicable only to grayscale images. While converting RGB images to grayscale is well-known[Sze10], converting depth images to grayscale is slightly different. It results in a rescaling of pixel values so that they fall within the grayscale range of 0 to 255. This effect can be seen in Figure 3.5b where the outline of the target becomes much more defined compared to Figure 3.4b. There is a limitation to how thick the target must be for this approach to work and this is discussed in Chapter 5.

3.2.3 Canny edge detection

In order to extract the features in Figure 3.5 for shape recognition, edge detection was utilized. One of the best performing algorithms is the Canny edge detector[MPW14]. Canny edge detection is a 5-stage process that uses filtering, intensity gradients as well as thresholding in order detect significant edges in a scene.

1. Canny edge detection first begins with the application of a Gaussian filter to remove noise from an image. A Gaussian filter is effective at this because it acts like a lowpass filter by decreasing the number of high frequency components in an image[NA12]. This results in blurring as seen in Figure 3.6. To perform Gaussian filtering in digital image processing, a Gaussian kernel must first be created before an image is convolved with it. This makes Gaussian blurring a linear filter. In 2 dimensions, the Gaussian function is expressed as seen in Equation 3.1. From that, a square Gaussian kernel of dimension m can be constructed as seen in Equation 3.2.

$$G(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} \tag{3.1}$$

$$\boldsymbol{K}_{ij} = \frac{1}{2\pi\sigma^2} \exp\Big(- \frac{(i-(k+1))^2 + (j-(k+1))^2}{2\sigma^2} \Big)$$
$$\text{where} \quad k = \frac{m-1}{2}, \quad i \geq 1, \quad j \leq (2k+1) \tag{3.2}$$

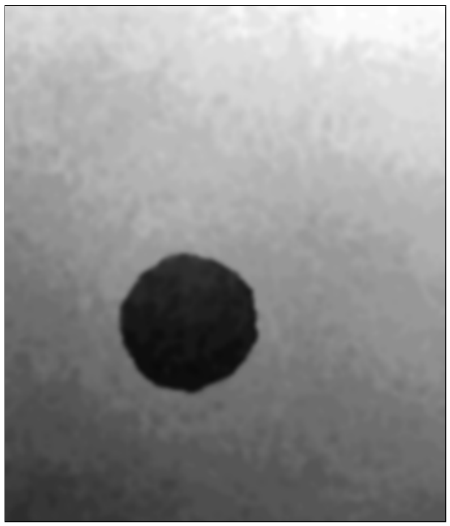
2. Next, the intensity gradient of the image is obtained. Using edge detection operators such as the Sobel operator, the edge gradient of the image can be obtained using Equation 3.3

$$I_{grad} = \sqrt{{I_x}^2 + {I_y}^2} \tag{3.3}$$

where I_x is the image's first derivative in the x direction while I_y is in the y direction. This results in thick edges as seen in Figure 3.7.

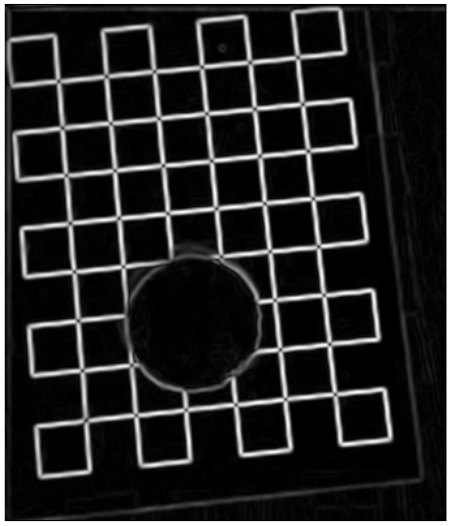
3. Although thick edges are easier to identify with the human eye, digital image processing considers this erroneous. This is due to the Canny's goal of one-to-one mapping of edge pixels to actual edges in the target image[Can87].

(a) : Gaussian blur applied to grayscale version of RGB image

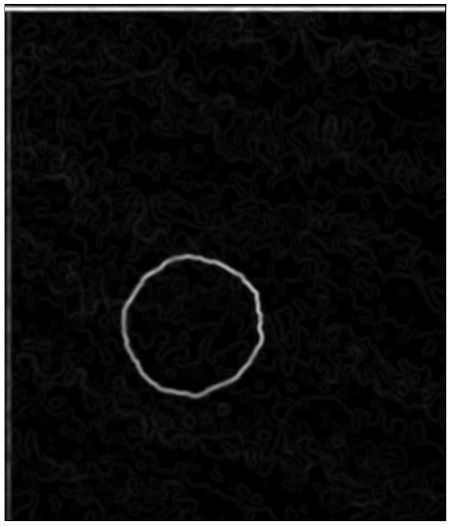


(b) : Gaussian blur applied to grayscale version of depth image

Figure 3.6: Example of Gaussian blur effect



(a) : Edge gradient of grayscale version of RGB image



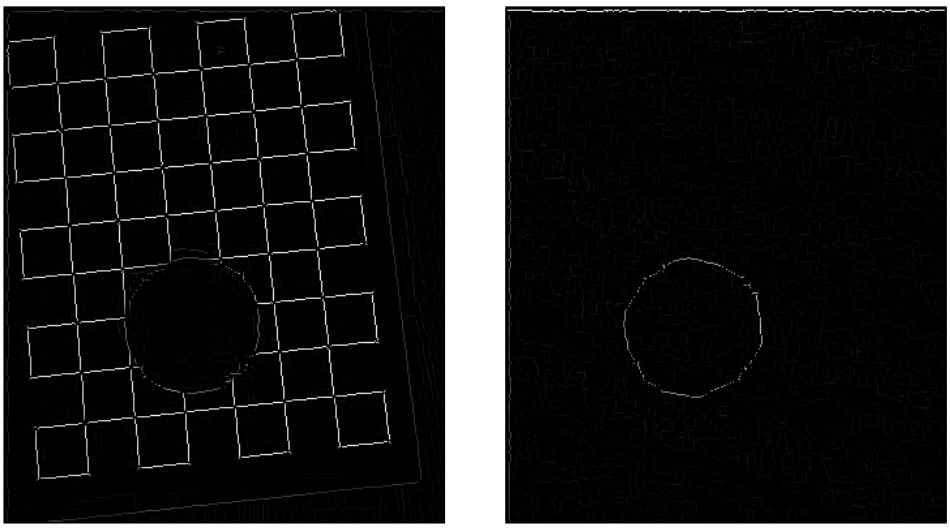
(b) : Edge gradient of grayscale version of depth image

Figure 3.7: Example of edge gradients of grayscale version of RGB and depth images

As such, edge thinning is required. The Canny edge detector achieves this by using non-maximum suppression. By determining the orientation of the edges using the image's first derivative as seen in Equation 3.4,

$$\theta = \arctan2(I_y, I_x) \tag{3.4}$$

pixels are able to be compared locally only by the direction specified by θ. Non-maximum suppression then suppresses a pixel (i.e. sets it to zero) if there are pixels in the specified direction which have greater intensity than it. Otherwise, the original value is kept. This yields the thinner edges as seen in Figure 3.8.



(a) : Non-maximum suppression applied to edge gradient of grayscale version of RGB image

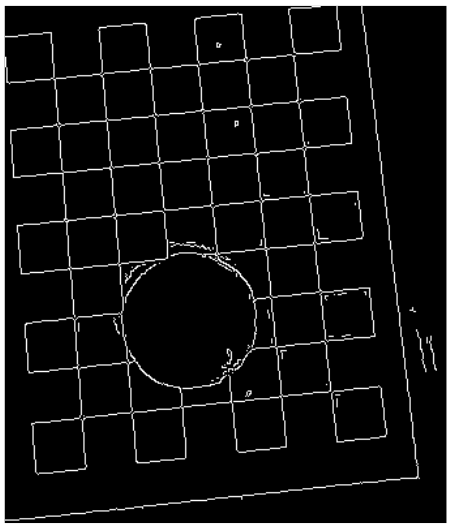
(b) : Non-maximum suppression applied to edge gradient of grayscale version of depth image

Figure 3.8: Example of non-maximum suppression applied to edge gradient images

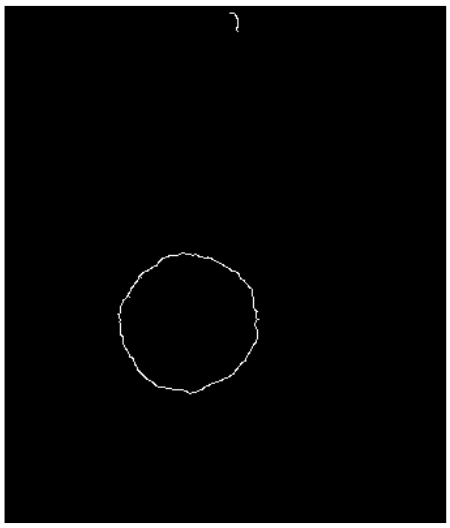
4. Although the remaining edge pixels are very representative of the edges in the actual image, they are not all of the same intensity values. This means that some of them may be still due to noise. The pixels are then classified as strong or weak in a process known as double thresholding. After choosing a high and low threshold value (usually decided automatically using Otsu's method[NA12]), pixels with intensities greater than the higher threshold are labeled as strong pixels. Those that fall between the high and low threshold values are considered as weak ones. Anything under the low threshold is not considered an edge and the strong-weak classification is used in the next step.

5. Finally, the Canny edge detection algorithm performs edge tracking by hysteresis. This means that if any of the 8 pixels surrounding a weak pixel is classified as strong, the pixel's intensity is changed to that of a strong pixel. Otherwise, it is changed to zero i.e. considered not part of an edge.

This final procedure then allows the edges in the image to be identified as seen in Figure 3.9.



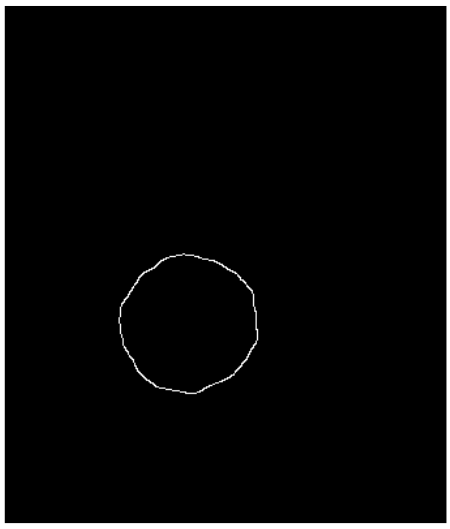
(a) : Canny edge detection results for grayscale version of RGB image



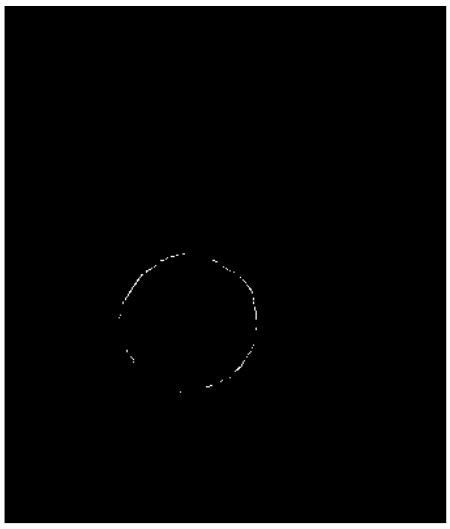
(b) : Canny edge detection results for grayscale version of depth image

Figure 3.9: Example of Canny edge detection results

3.2.4 RGB and depth image fusion



(a) : Edge of depth image to be used as mask



(b) : Result after using depth image edge as mask

Figure 3.10: Example of mask made from Canny edge detection results of depth image and its effect when applied to Canny edge detection results of RGB image

Looking at Figure 3.9b, it can be seen that Canny edge detection picked up the rough shape of the target only. In Figure 3.9a, however, the edges detected form a more accurate representation even though the checkerboard pattern was also detected. Although it was possible to simply use Figure 3.9b for shape detection, a better approach was devised in this thesis. By using it as a mask to filter out the unnecessary information in Figure 3.9a, more accurate edge results were possible. As such, small artefacts in Figure 3.9b were removed using Gaussian filtering. Then it was applied as a mask to Figure 3.9a.

3.2.5 Mathematical morphology

As seen in Figure 3.10b, the results are discontinuous since Canny edge detection produces thin edges. This means that when used as a mask, many relevant edge pixels are lost. Thus, there is a need to thicken the edges. This can be achieved via morphological operations.

In mathematics, morphology is the study of spatial/geometrical structures using set theory. In the domain of digital image processing, morphological operators are limited to 2-dimensions thus, they filter images and/or analyze its geometry using structuring elements[WR00]. A structuring element is simply a boolean image kernel that determines which neighborhood pixels are affected by a morphological operator. The most basic morphological operators are dilation, which expands the shapes in an image, and erosion, which does the opposite.

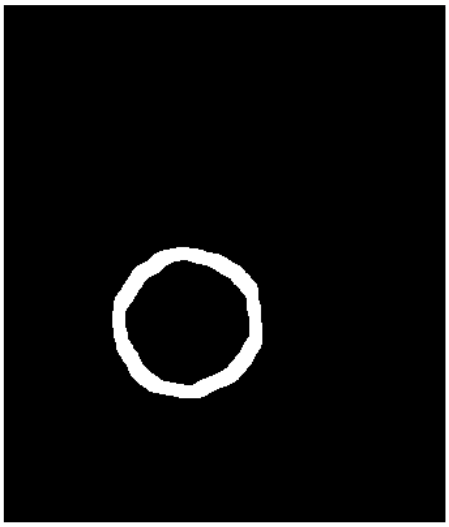
$$A \oplus B = \bigcup_{b \in B} A_b \tag{3.5}$$

Dilation, usually denoted as $\oplus$, is defined as seen in Equation 3.5. A is the set being analyzed, B is the structuring element and A_b is the translation of A by b. This means that the dilation of A by B, would result in all the points covered by B when its center iterates through every point in A. This assumes that structuring element B is centered at its origin. The operation yields a 'fatter' shape as seen in Figure 3.11a.

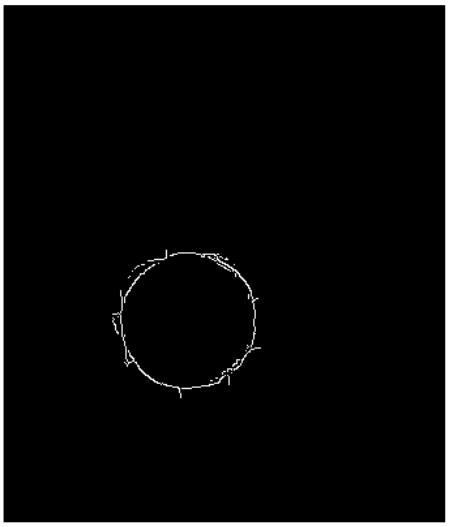
$$A \ominus B = \bigcap_{b \in B} A_{-b} \tag{3.6}$$

Erosion, on the other hand, is the 'opposite' of dilation. Usually denoted as $\ominus$, it is expressed as seen in Equation 3.6. The erosion of A by B yields all the points covered by the centroid of B while it iterates through points in A that would not result in any point in B leaving the area covered by A. This also assumes structural element B is centered at its origin. This operation's effects, when utilized with an elliptical structural element, can be seen in Figure 3.12b. The shape is shrunk after morphological closing operations to better represent the size of the target's shape.

Dilation and erosion are often used successively one after another. This method gives rise to the morphological opening, denoted as $\circ$, and closing, denoted as $\bullet$,

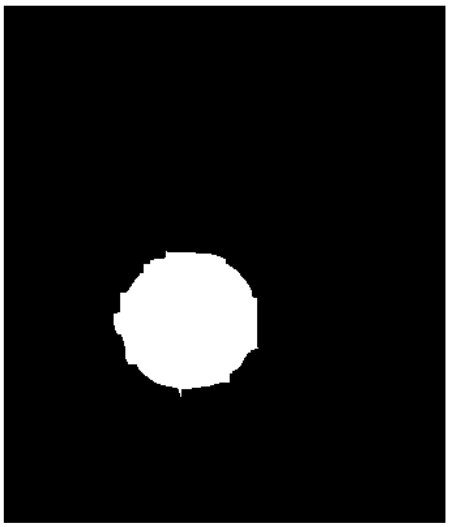


(a) : Dilated edge of depth image to be used as mask

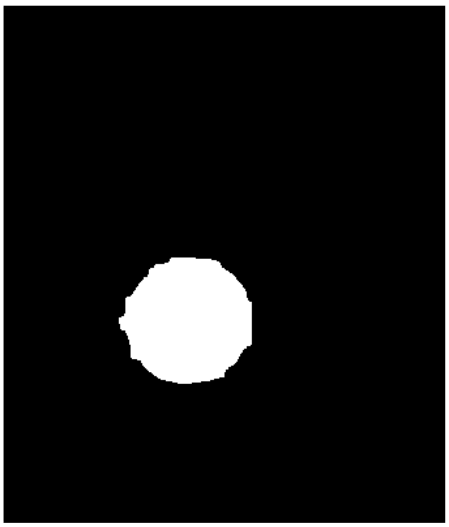


(b) : Result after using dilated depth image edge as mask

Figure 3.11: Example of dilating the mask made from Canny edge detection results of depth image and its effect when applied to Canny edge detection results of RGB image



(a) : Result of closing operation until one shape remaining



(b) : Eroded shape after closing operation

Figure 3.12: Example of morphological closing and erosion operations

operators as seen in Equations 3.7 and 3.8.

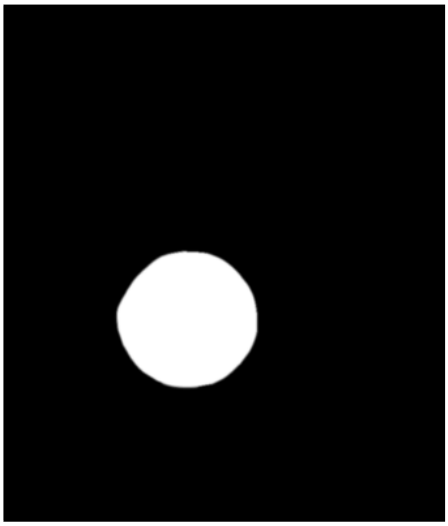
$$A \circ B = (A \ominus B) \oplus B \tag{3.7}$$

$$A \bullet B = (A \oplus B) \ominus B \tag{3.8}$$

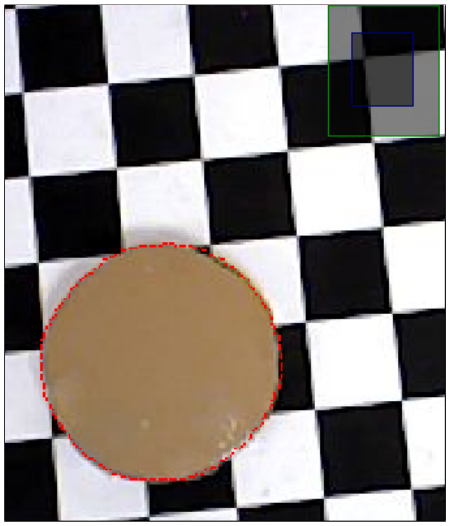
Both opening and closing are useful for what is known as 'salt and pepper' noise removal[WR00]. In such a scenario, opening 'disconnects' and 'breaks open'

blobs in an image while closing 'connects' and 'fills' them in. Given that the edges seen in Figure 3.11b are less than ideal, one approach to combat the issue is to 'fill' in the outlines until they correspond to the known number of targets (in this case, just one). This can be seen in Figure 3.12a where multiple closing operations would have filled in the gaps between the target's actual edges and any other edge due to noise. As such, erosion was used to shrink it down to account for the increase in size.

3.2.6 Median filtering



(a) : Result of median blurring eroded shape

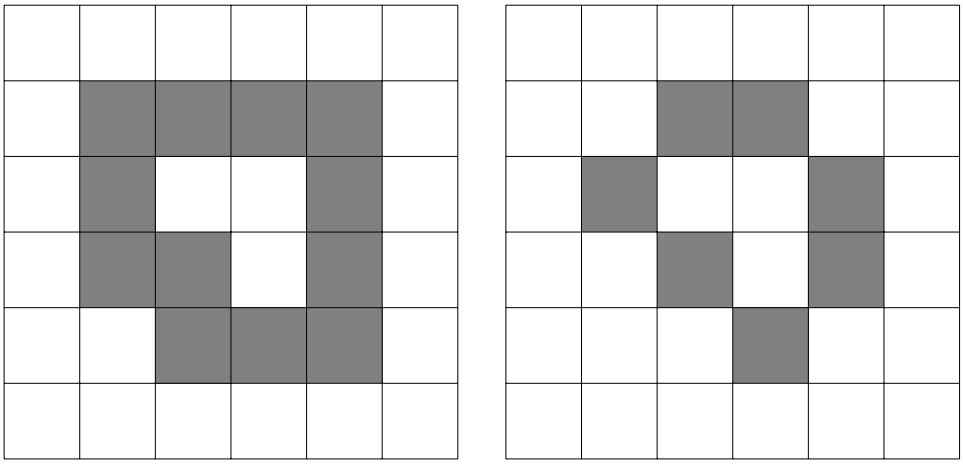


(b) : Contour detection results of 3.13a

Figure 3.13: Example of median blurring and contour detection

As it can be observed in Figure 3.12b, the edges of the final shape after multiple morphological closing and eroding operations were very jagged. This did not represent the target's actual shape very well. As such, some form of filtering/blurring is required to smooth them. One suitable approach is the use of median filtering. Unlike the linear Gaussian filter, it is capable of removing/smoothing noise while still preserving edges[WR00]. The concept behind this non-linear filter is straightforward. All that is required is the sorting of the pixel values as specified by a kernel before determining the median value. Implementation, however, can be difficult for large amounts of images as sorting algorithms have non-negligible computation times. Nevertheless, it is useful in this particular case as the median filter also has the tendency to round sharp corners[Sze10]. This is helpful for generating smooth trajectories and can be seen in Figure 3.13a.

3.2.7 Contour detection



(a) : 4-neighborhood connected contour

(b) : 8-neighborhood connected contour

Figure 3.14: Example of 4 and 8 neighborhood connectivity

Similar to edge detection, contour detection/tracing is the identification and drawing of the boundaries of an object or a segment in an image. This means that the definition of a contour is far more constrained than that of an edge which is any pixel where the difference in intensity is significant. Also, in most cases, contours are defined as closed boundary curves. This is due to the fact that the presence of an object or an obvious segment in an image should take up a specific area of it[Sze10]. As such, contour detection can also be used to count the number of objects or segments in an image.

Contours in a binary image can have two different types of border pixel connectivity as seen in Figure 3.14. They are traced using a number of different algorithms[AMFM11]. The most commonly implemented ones generally behave according to the following steps:

1. Iterate through the image until the first border pixel is detected.
2. Label that border pixel, $p_{count} = 1$, and backtrack to the previous pixel
3. Iterate clockwise through the 8-neighborhood of the first border pixel and label the next border pixel detected, $p_{count} = 2$, before backtracking again to the previous pixel.
4. Continue until the last border pixel detected is in the same position as the first border pixel in which case a complete contour has been detected, $p_{count} = n$
5. If there are no border pixels detected in 3., stop the process, $p_{count} = 0$, and move on to the next pixel after the one in 1. before repeating Steps 1. to 4. again.

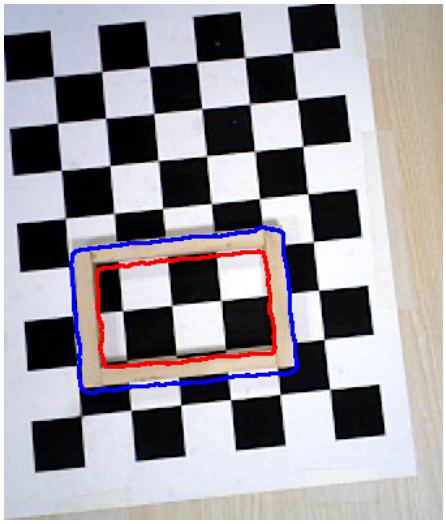


Figure 3.15: Example of contour hierarchy

With a number of conditions such as labelling any non-border pixel found in Steps 3 as part of a 'hole'[S^{+}85] within a contour, it then becomes possible for contours to be arranged in a hierarchical manner. This can be seen in Figure 3.15 where the red contour is a child of the blue contour. In the case of Figure 3.13b, the outermost contour was the only one of interest and hierarchy sorting made it easier to discard everything else. Hierarchy sorting and counting is expected to play a larger role in the case of multiple targets being present.

3.3 Non-photorealistc rendering (NPR)

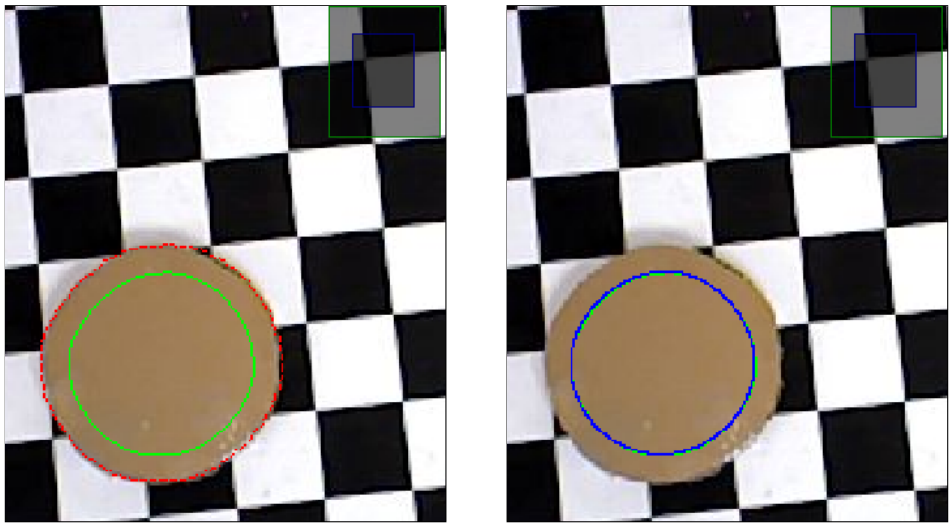
Even though the shape of the target object had been detected as seen in Figure 3.13b, there was still no guarantee that the contour was smooth i.e. had C^m continuity. Furthermore, for applications such as applying glue with a glue gun, the trajectory should be within the target rather than at its edges. These issues were tackled with the use of B-splines and scaling.

3.3.1 Scaling

Since the pixels that made up the contour of the target were all of equal importance, scaling it was not a complicated procedure. By determining the centroid of the contour, it was possible to then translate the entire contour so that it was centered around the origin, scale it by a user specified scaling factor and then translate the contour back to its original position. This yielded the change in size as seen in Figure 3.16a.

3.3.2 B-splines

The contour alone, however, was not enough to generate a trajectory as it only contained pixel positions. On top of that, the way that contours were generated



(a) : Original contour (red) scaled down (green)

(b) : 2D B-spline (blue) from scaled contour (green)

Figure 3.16: Example of contour scaling and generation of a 2D B-spline trajectory from it

meant that it was not necessarily C^m continuous. The contour's change in position, velocity and acceleration, jerk etc. was not necessarily smooth. As such, there was a need to approximate the contour using continuous curves. Two of the most popular are Bezier curves and B-splines.

Although Bezier curves are mathematically less complicated in comparison to B-splines, they have a fundamental downside. This downside is that for $n + 1$ control points, a Bezier curve of degree n must be used[AK03]. This makes it computationally expensive to fit a Bezier curve to a large number of control points as the number of coefficients (i.e. Bernstien polynomials) that need to be determined will also be large. One way to overcome this limitation is by connecting multiple lower degree Bezier curve segments together. This, however, has another issue where the continuity at the points of linkage are not naturally C^m (they can be constrained to have C^m continuity but this also requires computation time).

B-splines, on the other hand, have the advantage of their degrees being completely independent from their number of control points. This is because they are the interpolation of multiple Bezier curve segments rather than the interpolation of multiple points. Thus, B-splines can have all the benefits of Bezier curves without the continuity and coefficient computation issues. The major downside with B-splines is that it is not intuitive to use as its naive implementation does not pass through any of its control points. Nevertheless, this is not a big factor for the approximation of the contour via curve fitting. This is due to the fact that control points would be computed rather than provided

by a human user.

$$
\begin{aligned}
B(u) &= \sum_{i=0}^{n} P_i N_{i,p}(u) \quad \text{where } P \text{ are the control points and} \\
N_{i,0}(u) &= \begin{cases} 1 & \text{if } u_i \leq u \leq u_{i+1} \\ 0 & \text{otherwise} \end{cases} \\
N_{i,p}(u) &= \frac{u - u_i}{u_{i+p} - u_i} N_{i,p-1}(u) + \frac{u_{i+p+1} - u}{u_{i+p+1} - u_{i+1}} N_{i+1,p-1}(u)
\end{aligned}
\tag{3.9}
$$

The simplest way to determine the coefficients of a p degree B-spline with $n+1$ control points and $m+1$ knots is using Equation 3.9. Knots are the points where the Bezier curve segments that make up the B-spline are joined. They have the special property where if a knot is inserted at u in the curve $B(u)$ multiple times (i.e. increasing the knot multiplicity), the control point associated with the p-th knot becomes part of the B-spline. This can be seen in Figure 3.17 where the original control points (blue) are altered by knot insertion (green) without changing the B-spline (red). As it can be observed in Figure 3.17d, the yellow triangle highlights the interpolation of the control point by the B-spline after the 5th knot insertion.

It is important to note, however, that a B-spline is only C^{p-k} continuous at knot points with multiplicity k. This means that knot insertion alone can only guarantee C^0 continuity. In order to achieve knot points with greater continuity, it is then required to perform global B-spline interpolation[Die95]. This can be formulated by rewriting Equation 3.9 in matrix form as seen in Equation 3.10. Q is the vector of the known data points to be fitted and P is the vector of the control points that would allow the B-spline interpolate through all the points in Q.

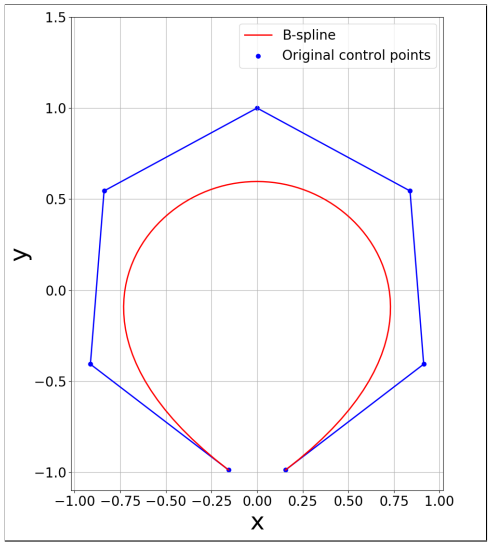
$$
\boldsymbol{Q} = B(u_j) = \sum_{i=0}^{n} N_{i,p}(u_j) P_i = \boldsymbol{N}\boldsymbol{P} \tag{3.10}
$$

$$
\text{where} \quad \boldsymbol{Q} = \begin{bmatrix} Q_0 \\ Q_1 \\ \vdots \\ Q_n \end{bmatrix}, \quad \boldsymbol{P} = \begin{bmatrix} P_0 \\ P_1 \\ \vdots \\ P_n \end{bmatrix}, \tag{3.11}
$$

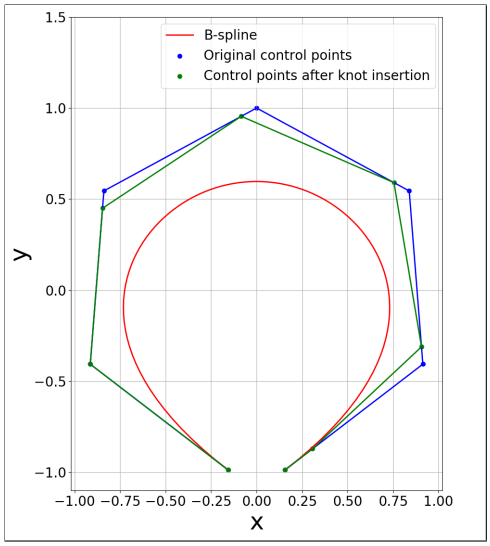
$$
\boldsymbol{N} = \begin{bmatrix} N_{0,p}(u_0) & N_{1,p}(u_0) & N_{2,p}(u_0) & \dots & N_{n,p}(u_0) \\ N_{0,p}(u_1) & N_{1,p}(u_1) & N_{2,p}(u_1) & \dots & N_{n,p}(u_1) \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ N_{0,p}(u_n) & N_{1,p}(u_n) & N_{2,p}(u_n) & \dots & N_{n,p}(u_n) \end{bmatrix} \tag{3.12}
$$

Note that N is a $n \times n$ matrix and in general, $u_0 = 0$ while $u_n = 1$. This means that the first and last rows of N will be filled with zeros with the exception of the first and last columns.

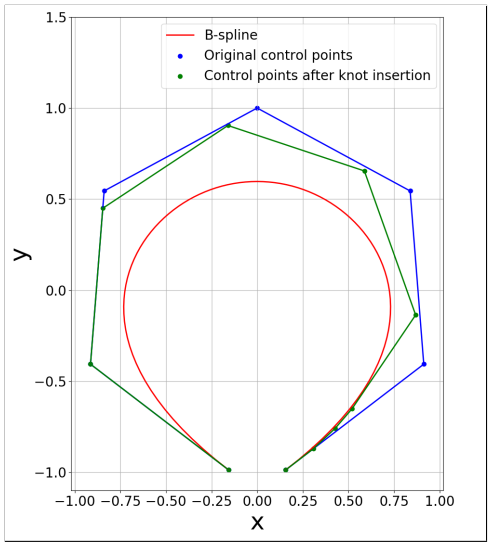
Equation 3.10 is a solvable linear system of equations where the size of matrix N is independent from the degree, p, of a B-spline. This is what allows B-splines to be more efficient in comparison to Bezier curves for curve fitting. Parameterizing the contour coordinate results in Figure 3.16a into equidistant segments of u and applying global interpolation then gives the result in Figure 3.16b. It contains a trajectory that is smooth and continuous compared to the contour detection results which only determine integer pixel positions. This can be seen in Figure 3.18 where the B-spline values (blue) are shifted slightly due to computation but essentially represent the same shape as the contour (orange)



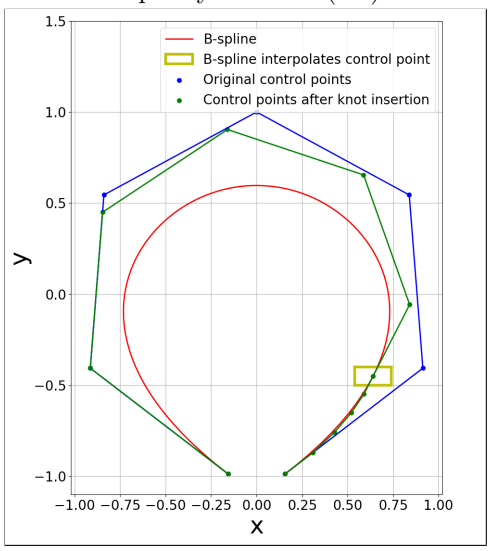
(a) : 5th degree B-spline with no knot multiplicity



(b) : 5th degree B-spline with knot multiplicity of 1 at $B(0.1)$

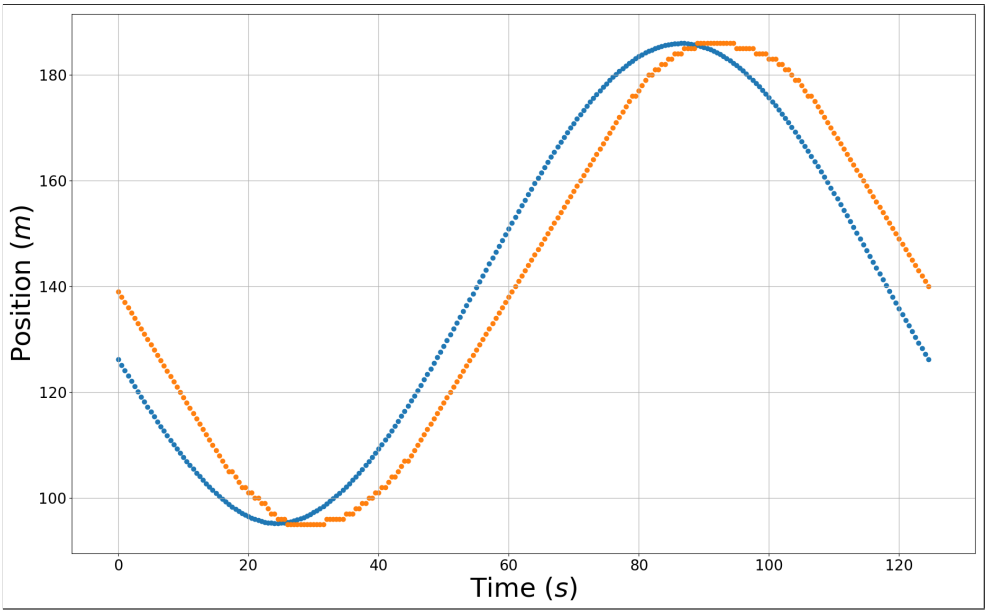


(c) : 5th degree B-spline with knot multiplicity of 3 at $B(0.1)$

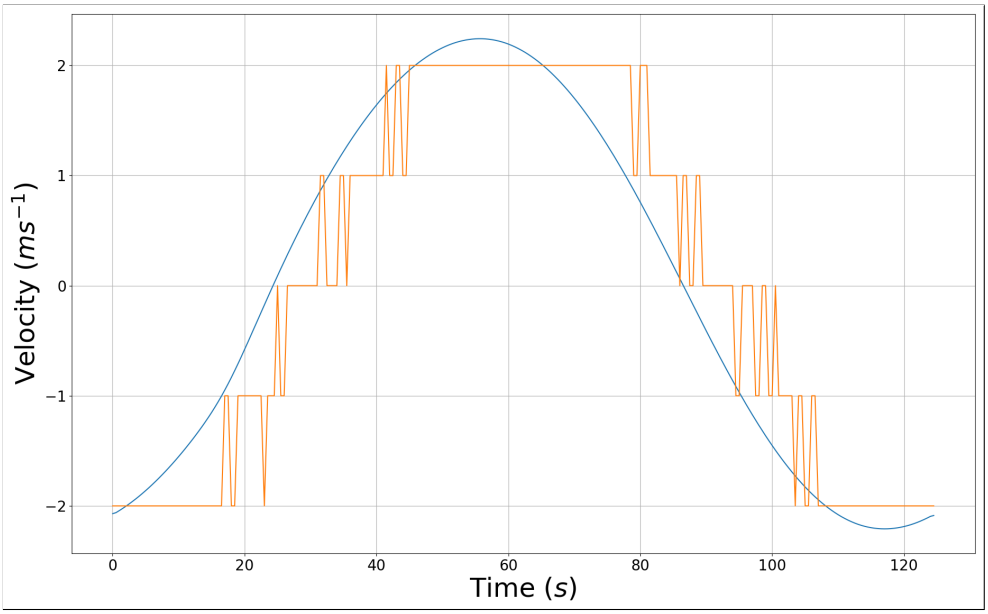


(d) : 5th degree B-spline with knot multiplicity of 5 at $B(0.1)$

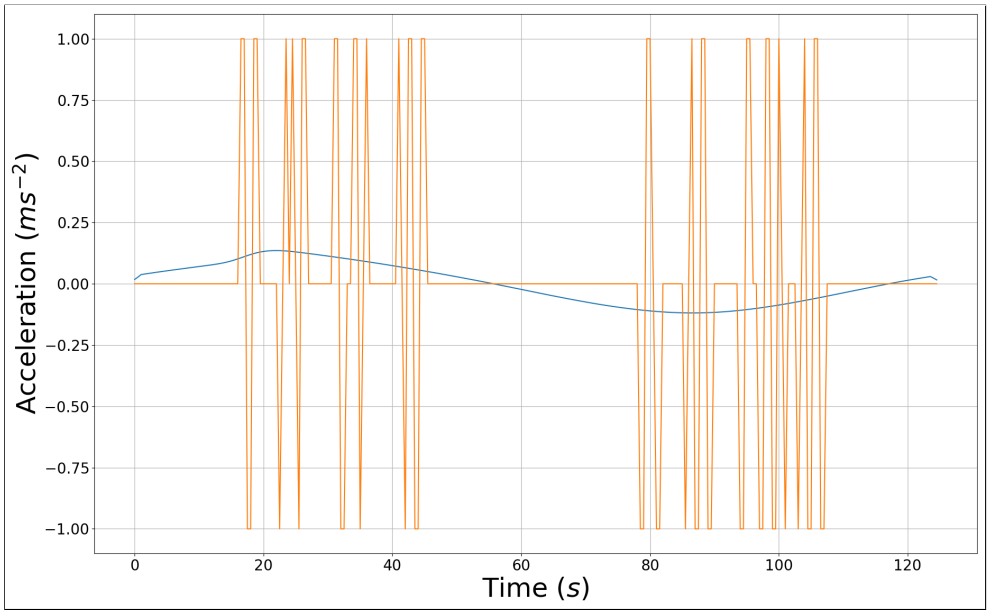
Figure 3.17: The effects of knot insertion on a $p = 5$ degree B-spline



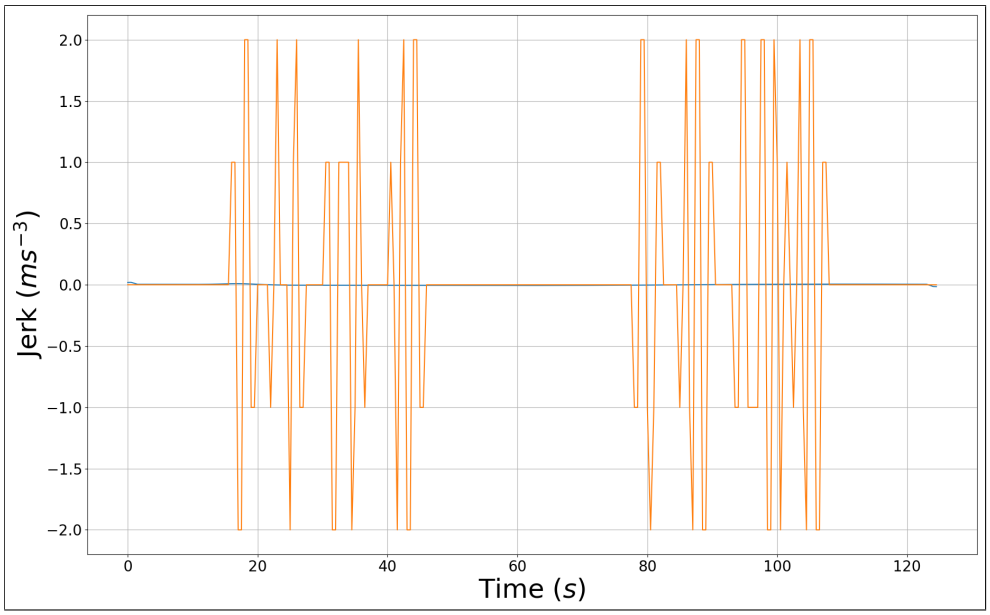
(a) : Profile of position (x component only) for contour (orange) and B-spline (blue) trajectories



(b) : Profile of first derivative (x component only) for contour (orange) and B-spline (blue) trajectories



(c) : Profile of second derivative (x component only) for contour (orange) and B-spline (blue) trajectories



(d) : Profile of third derivative (x component only) for contour (orange) and B-spline (blue) trajectories

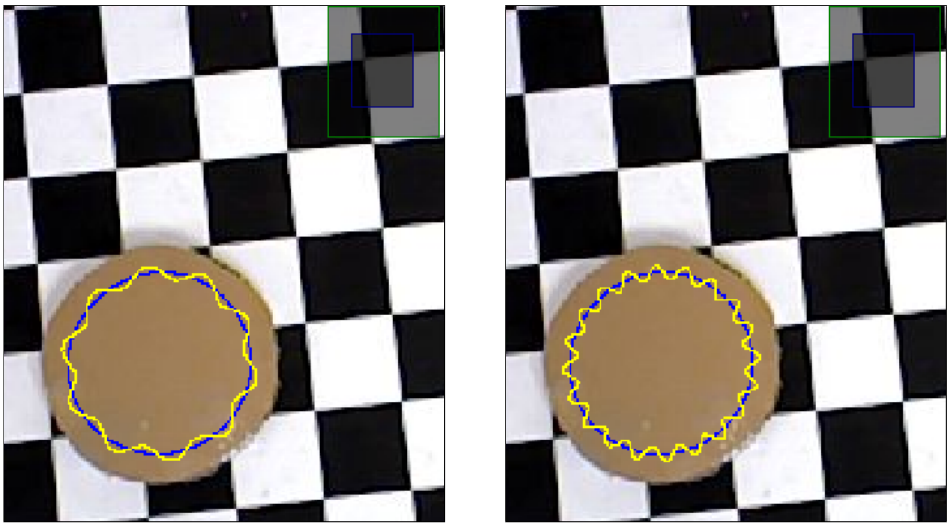
Figure 3.18: Example of difference in smoothness and continuity for the contour and B-spline trajectories in Figure 3.16b

results. Another interesting point to note is that the generated trajectory is closed i.e. it is a periodic B-spline. Periodic B-splines of degree p are a special case where their first and last p control points are exactly the same. This means the difference between their first and last $p+1$ knot values (their knot intervals) must also be the same.

3.3.3 Non-photorealistc trajectories (NPT)

Assuming the trajectory has a constant height, the method used in Section 3.3.2 was enough to generate a photorealistic (i.e. represents information in the image) trajectory in 3 dimensions. Nevertheless, the main goal of this thesis was to generate trajectories containing information that was not necessarily present in the image frame. One example is the generation of a sine wave pattern along the photorealistic B-spline (blue) as seen in Figure 3.16b. Since a sine wave is defined mathematically and is periodic, it was possible to generate one that followed a B-spline path created from a target's shape using Equation 3.13. A is a specified amplitude and n_p is a specified number of points per period. This was performed by calculating the relevant waypoints and then interpolating them with the global B-spline method mentioned in Section 3.3.2. This can be seen in Figure 3.19 where the NPTs are smooth and continuous.

$$\boldsymbol{v}_{\text{sine}} = \boldsymbol{v}_{\text{path}} + A \sin\left(\frac{2\pi}{n_p}\right)\hat{\boldsymbol{v}}_{\perp} \text{ where } \hat{\boldsymbol{v}}_{\perp} = \frac{\boldsymbol{v}_{\perp}}{\|\boldsymbol{v}_{\perp}\|} \text{ and } \boldsymbol{v} = \begin{bmatrix} x \\ y \end{bmatrix} \tag{3.13}$$



(a) : Low frequency sine trajectory example

(b) : High frequency sine trajectory example

Figure 3.19: Example of sine NPTs

Equation 3.13 demonstrates that it is relatively trivial to generate NPT using mathematical functions that are periodic in nature. Many functions, however, do not have this convenient property. One example is the conchoid

of de Sluze which is a family of mathematical curves studied in the 1600s. It is described parametrically using Equation 3.14. As seen in Figure 3.20, the conchoid is an anallagmatic (i.e. self-intersecting) curve when $a < -1$. If an NPT were to be generated from a region of the conchoid of de Sluze, that region would first need to be approximated in some manner. This is because it is not feasible to determine the correspondence between an anallagmatic curve and a non-anallagmatic B-spline. Once that approximation is determined, a transformation that minimizes the difference (the most common metric would be the sum of the squared error between points) between the approximation and the non-anallagmatic B-spline could then be computed. Applying that same transformation onto the conchoid's region of interest should result in a reasonable estimate of how an NPT containing multiple conchoids should look like. Thus, there was a need to look into methods to create an approximation of a complicated curve and to explore the type of transformation to use.

$$\begin{aligned} x_{\text{conchoid}} &= \cos\left(\sec(t) + a\cos t\right) \\ y_{\text{conchoid}} &= \sin\left(\sec(t) + a\cos t\right) \end{aligned} \tag{3.14}$$

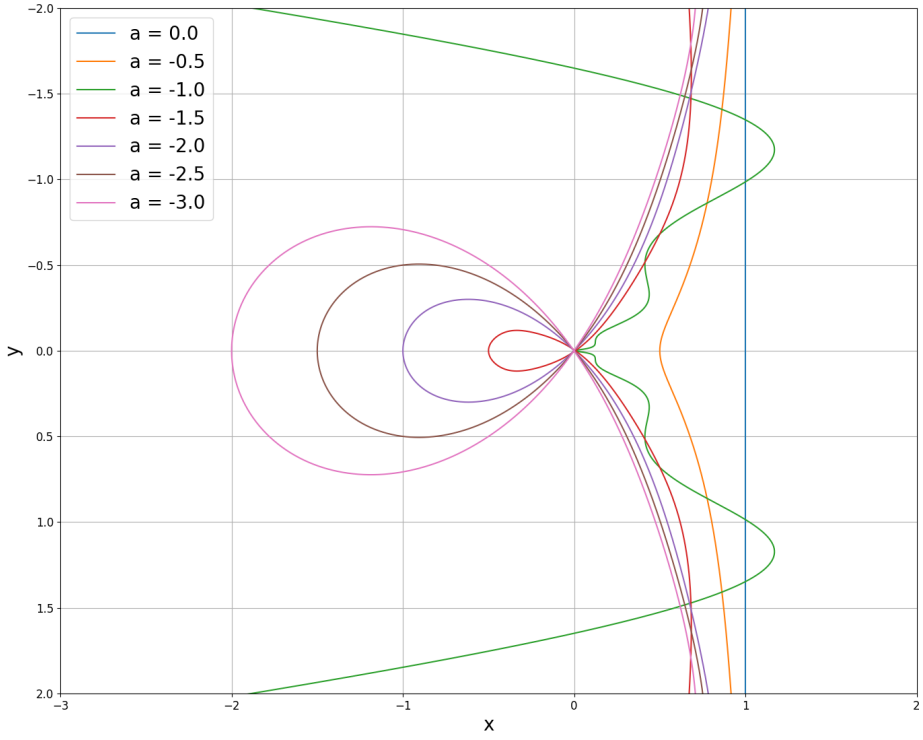


Figure 3.20: Example of the conchoid de Sluze curve with different parameters

A straightforward method to approximating a complex curve was formulated in this thesis. Working on the assumption that all its points are weighted equally, it was noticed that an approximation could be made by drawing another simpler curve which passed through the centroid of the complex curve. As seen in Figure 3.21, by:

- generating a simpler curve (green) using the endpoints of the complex curve (blue) and,
- translating (yellow) the simpler curve towards the centroid of the complex curve,

it became possible to create an approximation curve (pink). The red markers were computed using Equation 3.14 and used as interpolation points for Equation 3.10 which resulted in the blue points. It was now possible to determine the transformation $\boldsymbol{H}$ that would result in the least square error between the approximation curve (pink) and a second B-spline (cyan). This can be seen in Figure 3.21.

Based on [HZ03], 2D transformations can be classified into 4 types. Each type has an increasing number of degrees of freedom (dof). The first type is isometries or rigid transformations which include translation, rotation and reflection. They are described by the homogeneous Equation 3.15 where $\epsilon = \pm 1$ determines whether the image is reflected, θ determines how much the image is rotated by and Δt determines how much the centroid of the image is translated by. This means that in 2-dimensions, isometric transformations have 3 dof – 1 for rotation and 2 for translation. This makes them useful for rigid correspondence problems where scaling, shearing etc. are undesirable.

$$\boldsymbol{P}' = \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} \epsilon\cos\theta & -\sin\theta & \Delta x \\ \epsilon\sin\theta & \cos\theta & \Delta y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} \boldsymbol{R}(\theta) & \Delta\boldsymbol{t} \\ 0 & 1 \end{bmatrix} \boldsymbol{P} \tag{3.15}$$

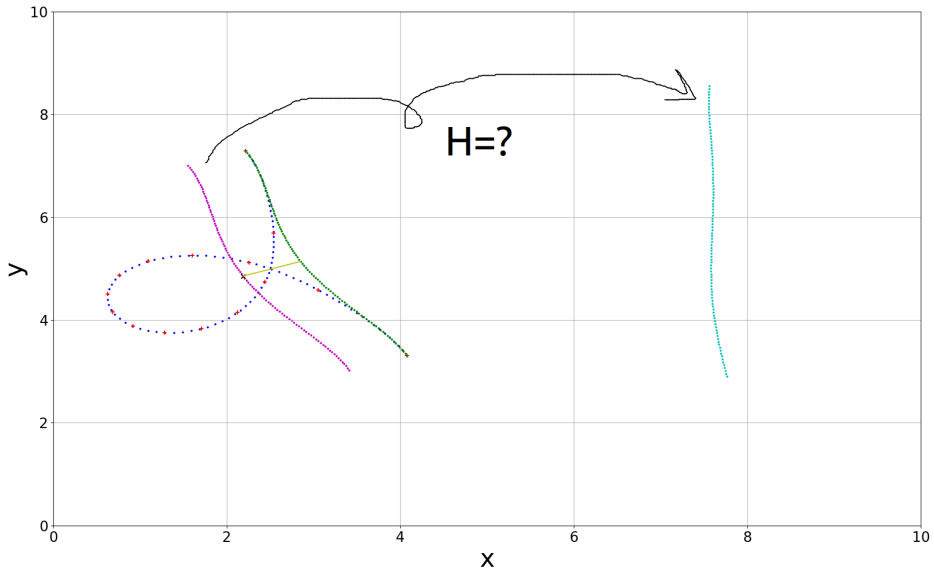


Figure 3.21: Example of an approximation curve for a region of the conchoid

The next type is known as similarity transforms which adds a uniform scaling factor, s, to the rigid transform as seen in Equation 3.16. The scaling adds an extra degree of freedom which makes the similarity transform effective at preserving the general 'shape' of an image/curve.

$$\boldsymbol{P}' = \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} s\cos\theta & -s\sin\theta & \Delta x \\ s\sin\theta & s\cos\theta & \Delta y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} s\boldsymbol{R}(\theta) & \Delta\boldsymbol{t} \\ 0 & 1 \end{bmatrix} \boldsymbol{P} \tag{3.16}$$

In comparison to isometric and similarity transforms, affine transformations require at least 3 coordinates/points instead of 2 in order to determine correspondence. This is due to the fact that affine transformations have 6 dof, as seen in Equation 3.17. These include shear and non-uniform scaling on top of translation, rotation and scaling. With the additional dof, affine transformations are suitable for cases where line parallelism and length ratios have to be maintained.

$$\boldsymbol{P}' = \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} a_{1,1} & a_{1,2} & \Delta x \\ a_{2,1} & a_{2,2} & \Delta y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} \boldsymbol{A} & \Delta\boldsymbol{t} \\ 0 & 1 \end{bmatrix} \boldsymbol{P} \tag{3.17}$$

Finally, the last type of 2D transformation is projective transformations which adds on another 2 dof to give it a total of 8 dof. The new dof enables it to rotate an image 'inwards' around both the x and y axes thus allowing for the creation of vanishing points in images. This is a useful property for perspective projection which is the projection of a 3D object onto a 2D surface. This means that 4 instead of 3 coordinates/points are required for projective transformations to determine correspondence. This is due to the fact that 2D projective transformation matrices can be any arbitrary 3×3 matrix as seen in Equation 3.18. Here, the only property of the original image that is preserved is that straight lines remain straight.

$$\boldsymbol{P}' = \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} a_{1,1} & a_{1,2} & \Delta x \\ a_{2,1} & a_{2,2} & \Delta y \\ v_1 & v_2 & v \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} \boldsymbol{A} & \Delta\boldsymbol{t} \\ \boldsymbol{v} & v \end{bmatrix} \boldsymbol{P} \tag{3.18}$$

Based on Table 3.2, it was inferred that isometric transformations would be too constrained to help generate NPTs. This is because the values of the approximation curve may be scaled differently compared to the non-anallagmatic B-spline path. Projective transformations, on the other hand, would not retain enough of the complex curve's properties for the transformed result to be 'faithful' to the original. Of the remaining two, the similarity transform was considered the more promising solution. Nevertheless, there was still the issue of whether it could handle determining the correspondence between 2 curves that had significantly different shapes. In order to determine the affine transformation that would result in the least square distance between the approximation curve (pink) and second B-spline (cyan), Equation 3.17 was rewritten into the system of equations seen in Equation 3.19. This was valid since the last row of matrix $\boldsymbol{H}$ should

Transformation type	DOF	Properties
Isometric	3	Translation, rotation and reflection Retains size and shape
Similarity	4	Adds scaling Retains shape
Affine	6	Adds shearing and non-uniform scaling Retains line parallelism and length ratios
Projective	8	Adds perspective projection Retains line straightness

Table 3.2: Summary of 2D transformation types and properties

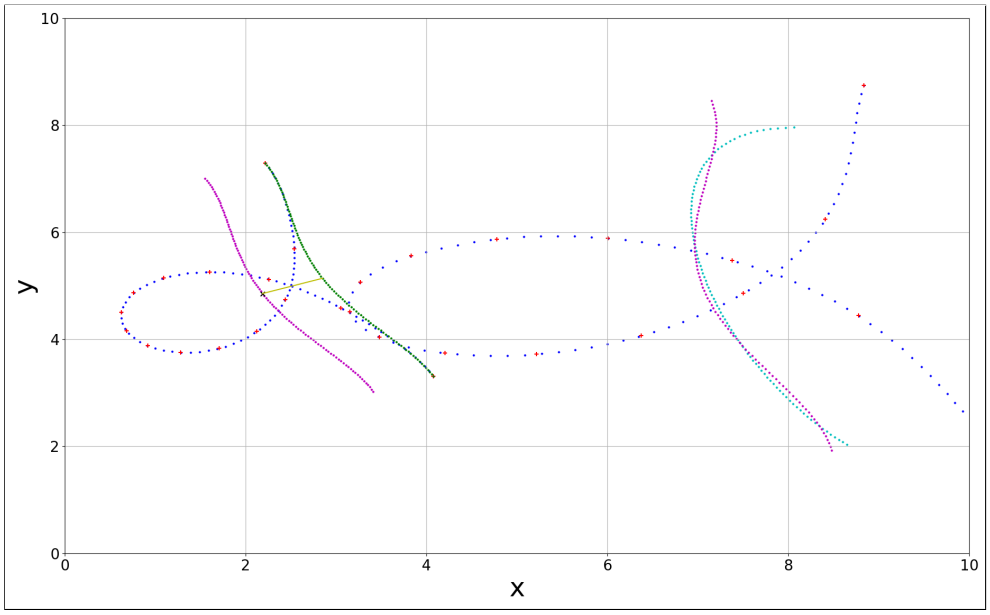
always be [0 0 1] for an affine transformation. From it, the system of equations for the least square error similarity transform could also be obtained. This is because removing the extra 2 dof controlling shear and non-uniform scaling is equivalent to constraining $\boldsymbol{A}$ so that both vectors are orthogonal to each other i.e. $a_{2,1} = -a_{1,2}$ and $a_{2,2} = a_{1,1}$. This then simplifies Equation 3.19 to Equation 3.20. The elements of $\boldsymbol{T}$ could then be computed by using the pseudo-inverse of rectangular matrix $\boldsymbol{M}$ as seen in Equation 3.21.

$$\boldsymbol{P} = \begin{bmatrix} x'_0 \\ y'_0 \\ x'_1 \\ y'_1 \\ \vdots \\ x'_n \\ y'_n \end{bmatrix} = \begin{bmatrix} x_0 & y_0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & x_0 & y_0 & 1 \\ x_1 & y_1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & x_1 & y_1 & 1 \\ & & & \vdots & & \\ x_n & y_n & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & x_n & y_n & 1 \end{bmatrix} \begin{bmatrix} a_{1,1} \\ a_{1,2} \\ \Delta x \\ a_{2,1} \\ a_{2,2} \\ \Delta y \end{bmatrix} = \boldsymbol{MT} \tag{3.19}$$

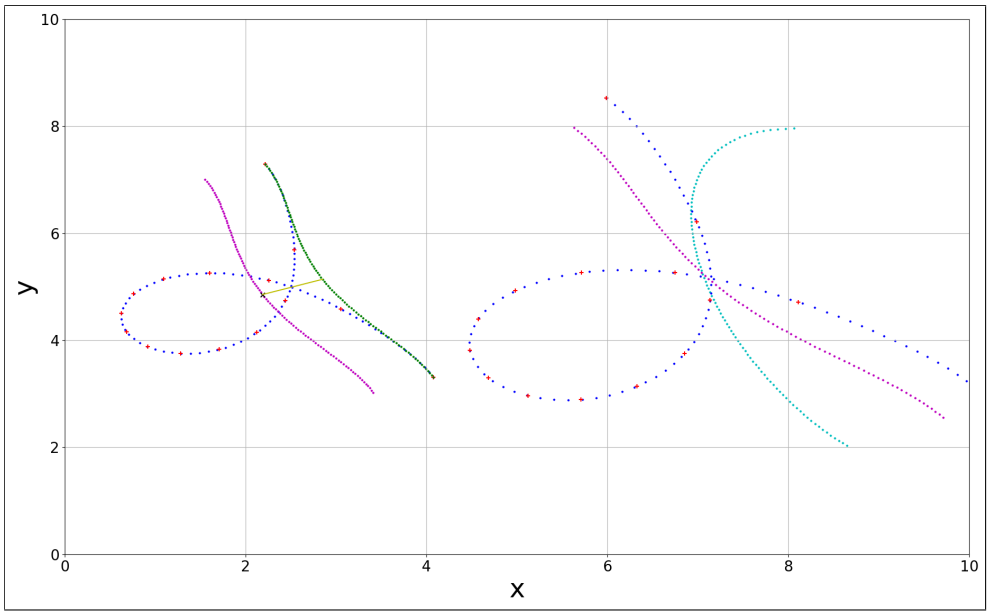
$$\boldsymbol{P} = \begin{bmatrix} x'_0 \\ y'_0 \\ x'_1 \\ y'_1 \\ \vdots \\ x'_n \\ y'_n \end{bmatrix} = \begin{bmatrix} x_0 & y_0 & 1 & 0 \\ y_0 & -x_0 & 0 & 1 \\ x_1 & y_1 & 1 & 0 \\ y_1 & -x_1 & 0 & 1 \\ & \vdots & & \\ x_n & y_n & 1 & 0 \\ y_n & -x_n & 0 & 1 \end{bmatrix} \begin{bmatrix} a_{1,1} \\ a_{1,2} \\ \Delta x \\ \Delta y \end{bmatrix} = \boldsymbol{MT} \tag{3.20}$$

$$\boldsymbol{T} = \boldsymbol{M}^{\dagger}\boldsymbol{P} = ((\boldsymbol{M}^T\boldsymbol{M})^{-1}\boldsymbol{M}^T)\boldsymbol{P} \tag{3.21}$$

Reorganizing the solved terms in $\boldsymbol{T}$ into transformation matrix $\boldsymbol{H}$ then resulted in a 2D transformation matrix. This could then be applied to the region of interest on the conchoid de Sluze curve as seen in Figure 3.22. The approximation (pink) and complex (blue) curves on the left are before transformation while those on the right are after transformation. As it can be observed in Figure 3.22a, the affine transformation results yielded an approximation curve (pink) that more

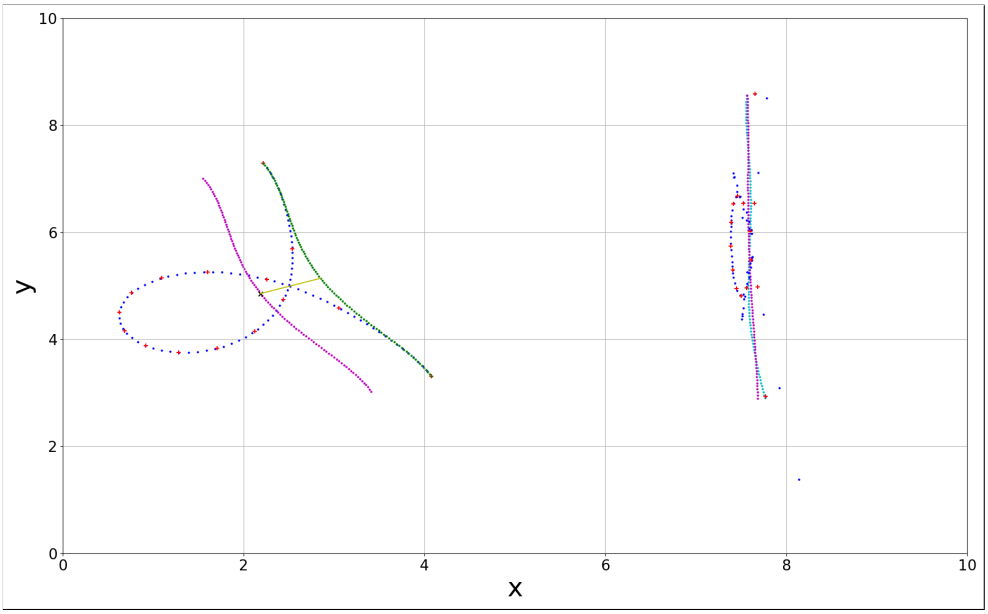


(a) : Least square error affine transformation between approximation curve (pink) and 2nd B-spline (cyan) applied to complex curve (blue)

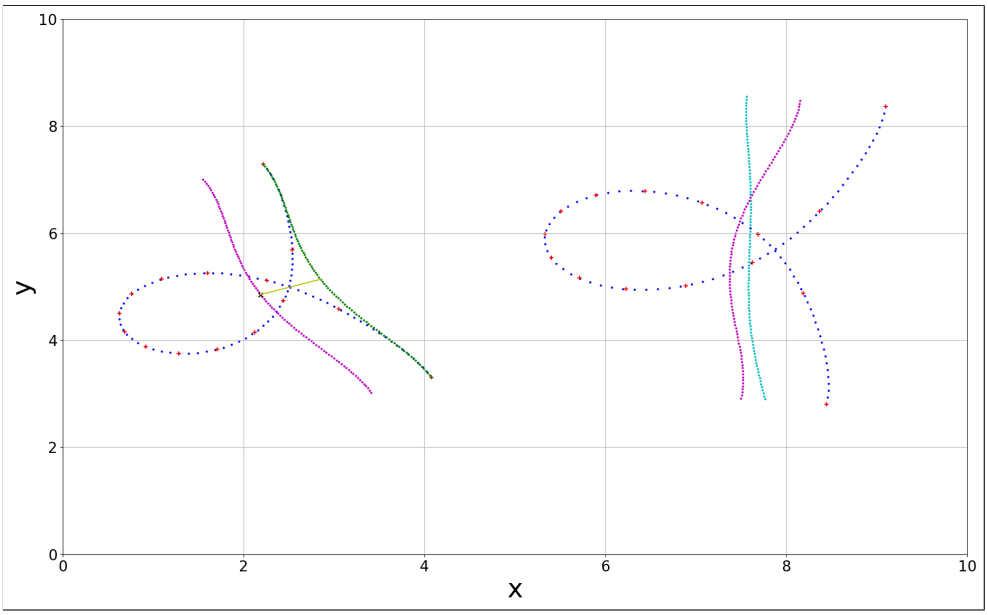


(b) : Least square error similarity transformation between approximation curve (pink) and 2nd B-spline (cyan) applied to complex curve (blue)

Figure 3.22: Example of applying affine and similarity transformations on a region of the conchoid (blue)



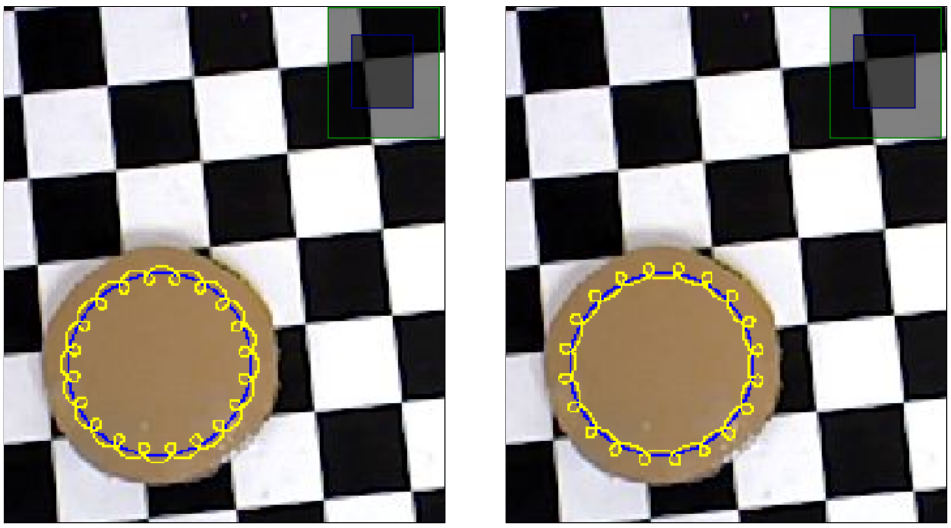
(a) : Least square error affine transformation between approximation curve (pink) and 2nd 'straighter' B-spline (cyan) applied to complex curve (blue)



(b) : Least square error similarity transformation between approximation curve (pink) and 2nd 'straighter' B-spline (cyan) applied to complex curve (blue)

Figure 3.23: Detrimental example of applying affine and similarity transformations on a region of the conchoid (blue)

closely 'fits' the second B-spline (cyan). This was less the case in Figure 3.22b. On the other hand, the affine transformed complex curve (blue) was 'stretched' significantly more compared to the similarity transformed case. This can be attributed to the affine transformation's ability to shear and scale non-uniformly. Although this effect was not necessarily a downside in the example shown in Figure 3.22, it became detrimental when the second B-spline was 'straighter' as seen in Figure 3.23. Here, the affine transform's extra 2 dof 'squashed' the complex curve until most of its shape was lost.



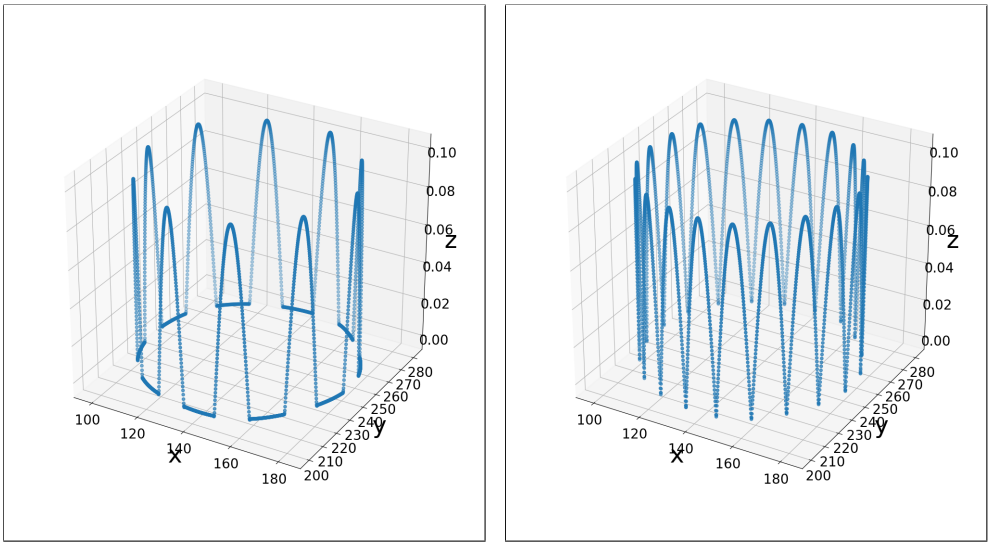
(a) : Inward facing trajectory generated from conchoid of de Sluze

(b) : Outward facing trajectory generated from conchoid of de Sluze

Figure 3.24: Example of NPTs created from the conchoid of de Sluze

Based on the results seen in Figure 3.23, the similarity transform was utilized to generate the NPT (yellow) seen in Figure 3.24. This transformation was repeated segment by segment across the entire photorealistic B-spline path (blue) in Figure 3.24. This resulted in what could be considered as the 'ideal'/non-distorted version of the trajectory since each self-intersecting loop is generated mathematically. Another further benefit of using this process is that the NPT's orientation could be flipped so that it would 'face' inwards or outwards.

Even though all results until now have assumed that the generated trajectory would have constant height, there are some simple cases where this would be erroneous such as dashed trajectories. For such trajectories, the height (i.e. z-values) should vary over time as the tip of the glue gun approaches and distances itself from the surface of the target. An efficient approach to generating such trajectories is with the use of Fourier series. The Fourier series is the estimation of a periodic function using the linear combination of sinusoids and is described as seen in Equation 3.22. a_0, a_n and b_n are Fourier coefficients while T is the length of one period. Based on [Han14], the Fourier coefficients for some



(a) : 3D dashed trajectory after applying half-rectified sine wave to 2D trajectory

(b) : 3D dotted trajectory after applying full-rectified sine wave to 2D trajectory

Figure 3.25: Example of dotted and dashed trajectories in 3D

Wave type	a_0	a_n	b_n
Half-rectified sine	$\frac{2A}{\pi}$	$\begin{cases} \frac{-2A}{\pi(1-n^2)} & \text{for } \mathrm{mod}(n,2) = 0 \\ 0 & \text{for } \mathrm{mod}(n,2) \neq 0 \end{cases}$	$\begin{cases} \frac{A}{2} & \text{for } n = 1 \\ 0 & \text{for } n > 1 \end{cases}$
Full-rectified sine	$\frac{4A}{\pi}$	$\begin{cases} \frac{-4A}{\pi(1-n^2)} & \text{for } \mathrm{mod}(n,2) = 0 \\ 0 & \text{for } \mathrm{mod}(n,2) \neq 0 \end{cases}$	0
Sawtooth/ramp	A	0	$-\frac{A}{n\pi}$

Table 3.3: Fourier coefficients for some common waves

common waves are as seen in Table 3.3. Of interest are the rectified sine waves were as they could be used to determine the behavior of a dashed trajectory. Applying the relevant coefficients to Equation 3.22 results in Equations 3.23 and 3.24. They could then be applied to the 2D B-spline trajectory generated in Figure 3.16b after paramterizing it by length. This yielded the 3D version as seen in Figure 3.25.

$$f(t) = \frac{a_0}{2} + \sum_{n=1}^{\infty} \Big[a_n \cos(\omega n t) + b_n \sin(\omega n t) \Big] \tag{3.22}$$

$$\text{where} \quad a_n = \frac{2}{T} \int_0^T f(t) \cos(\omega n t) dt,$$

$$b_n = \frac{2}{T} \int_0^T f(t) \sin(\omega n t) dt,$$

$$\omega = \frac{2\pi}{T}$$

$$z_{\text{half}} = \frac{A}{\pi} + \frac{A}{2} \sin(\omega t) - \frac{2A}{\pi} \sum_{n=1}^{\infty} \frac{\cos(2n\omega t)}{4n^2 - 1} \tag{3.23}$$

$$z_{\text{full}} = \frac{2A}{\pi} - \frac{4A}{\pi} \sum_{n=1}^{\infty} \frac{\cos(n\omega t)}{4n^2 - 1} \tag{3.24}$$

3.4 Trajectory classification

With a method to generate NPTs, there was now a need to associate user inputs (i.e. human drawn curves) with the NPTs that were considered the most similar. Although the HTC Vive system has a consistent sampling rate[NLL17], the inconsistency in the speed of human motion and the type of trajectory being drawn made the number of coordinate points for every trajectory segment of fixed length non-equal. This was an issue as correspondence between coordinates of different trajectories (which this thesis uses as part of the feature vector) would be unknown to the classifier being trained[LKP07]. In order to overcome this issue, the global interpolation method for B-splines in Section 3.3.2 was used to resample the trajectory segments. This was so that each would contain the same number of coordinate points which would allow correspondence between trajectory points to be determined naturally.

3.4.1 Supervised learning

Supervised learning is generally about the learning of the mapping between data and their respective labels. After being trained, the classifier is utilized to predict the labels i.e. the new data's class. Supervised learning is very common in cases like spam filtering and is usually very simple to implement. It, however, suffers from the downside that the number of classes must be known and that

all training data instances need to be labelled beforehand which can be very time consuming[MRT18]. In this thesis' case, data instances are never expected to number more than a few thousand and all class labels are known which makes the supervised learning approach suitable for the classification problem.

Classification algorithms can be split into two general types, parametric and non-parametric[Alp09]. In the parametric case, it is assumed that the dataset being used has a known distribution (usually Gaussian). This allows the parametric model to base itself on the mixture of probability distributions. This means that parametric learners have a fixed structure which means fixed complexity and number of parameters. Their optimal parameters (e.g. mean and variance) can usually be determined using approaches like maximum likelihood estimation. Some examples of parametric learners are linear regression and SVMs with linear kernels which could be used for multiclass problems.

Non-parametric models, on the other hand, can have potentially infinite parameters due to the fact that the structure and the complexity of the model grows as the amount of training data increases. This means that non-parametric models are appropriate for datasets where no assumption can be made about their structure which is the case for this thesis. The downside to using a non-parametric model is that it is memory-based, all training instances have to be stored before a final prediction model can be built from them. This is different from parametric models which only require the computation of the optimal parameters to build their prediction model. Examples of non-parametric learners include decision trees, k-nearest neighbor (KNN) and SVMs with the radial basis function (RBF) kernel. All of these were used in the classification of this thesis' trajectories.

Decision Tree

One of the earliest non-parametric models for supervised learning is the decision tree algorithm[Bel59]. It is hierarchical in nature and has branches that grow in number as more training data is provided. A decision tree consists of decision nodes, Q, that branch into two other decision nodes until the branches terminate at leaf nodes which contain the data classes. At every decision node, a test function $Q(\theta)$ is applied to the input. The results determine which branch to follow and this repeats until a leaf node. Every decision node recursively splits the data space into left and right subsets as seen in Equation 3.25. This is done until all data with the same labels are grouped together.

$$Q(\theta) = \begin{cases} Q_{\text{left}}(\theta) & \text{for } x_j \leq t_m \\ Q_{\text{right}}(\theta) & \text{otherwise} \end{cases} \tag{3.25}$$

where x is the training vector, $\theta = (j, t_m)$ is the decision split containing feature j and the decision threshold at the m-th node, t_m. The impurity (i.e. the frequency of a randomly chosen data from the training set being randomly labeled incorrectly) of node m can then be determined by Equation 3.26.

$$G(Q, \theta) = \frac{n_{\text{left}}}{N_m} I(Q_{\text{left}}(\theta)) + \frac{n_{\text{right}}}{N_m} I(Q_{\text{right}}(\theta)) \tag{3.26}$$

where n_{left} and n_{right} are the number of data samples in each subset, N_m is the total number of data samples at node m i.e. $n_{\text{left}} + n_{\text{right}}$ and I is the impurity metric which in this thesis' case is according to Equation 3.27.

$$I(X_m) = \sum_k p(k|m)(1 - p(k|m)) \tag{3.27}$$

Here, $p(k|m)$ is the probability of observing a data sample from class k in node m. With $G(Q, \theta)$ known, the decision splits that will minimize impurity metric I can then be obtained from Equation 3.28. In this manner, Equations 3.25 to 3.28 are performed recursively so that the optimal split, θ^*, at every decision node is determined.

$$\theta^* = \arg\min\left(G(Q, \theta)\right) \tag{3.28}$$

K-nearest neighbor (KNN)

Another early non-parametric learner is the KNN algorithm[CH+67]. In the KNN, the class of some input data is determined by a majority vote of the k-nearest neighbors to it. This makes the algorithm's efficiency dependent on the size of its training set. This is due to the fact that it has to store every single training data in order make predictions about new data by computing the distances between them. The distance used is usually the euclidean distance and the parameter k determines the amount of smoothing between class boundaries[Alp09]. This is due to the fact that having multiple neighbors reduces the effect of noise on class borders.

Support Vector Machine (SVM)

In contrast to the decision tree and KNN algorithms, SVMs are very recent[CV95]. They have been used to great effect on non-linear classification due to the SVMs' ability to map data into higher-dimensional space. The purpose of this mapping is so that data, that is not linearly separable in its original dimensions, may be transformed and separated by a hyperplane in higher dimensions. Even if the data is linearly separable, SVMs can improve upon the separation by determining what is known as the maximum-margin hyperplane[MRT18]. This means that the distance from the nearest point to the class boundary is maximized. Assuming binary classes, the maximum-margin hyperplane consists of the points $\boldsymbol{x}$, known as support vectors, that satisfy Equation 3.29. $\boldsymbol{w}$ is the normal vector to the hyperplane and b is a scalar that is part of the parameter $\frac{b}{\|\boldsymbol{w}\|}$ which describes the hyperplane's offset from the origin in the direction of $\boldsymbol{w}$.

$$\boldsymbol{w} \cdot \boldsymbol{x} - b = 0 \tag{3.29}$$

If linearly separable, two parallel hyperplanes should be found at Equation 3.29 equal to 1 and -1 (assuming binary classes) instead of 0. These parallel hyperplanes should contain the data points closest to the class border. Hence,

the convex optimization problem in Equation 3.30 has to be solved in order to find the parallel hyperplanes that have maximum distance between them. This is because the distance between both is equivalent to $\frac{2}{\|\boldsymbol{w}\|}$.

$$\begin{aligned} \min_{\boldsymbol{w},b} \quad & \|\boldsymbol{w}\|^2 \\ \text{subject to:} \quad & y_i(\boldsymbol{w}\cdot\boldsymbol{x}_i - b) \geq 1, \quad \forall i \in [1,n] \end{aligned} \tag{3.30}$$

where y_i is one of the binary classes (1 or -1) and n is the number of training data. On the other hand, data that is not linearly separable requires a more relaxed version of the constraint seen in Equation 3.30. This is introduced by using slack variables, ξ_i. Their values are usually determined by the hinge loss function[MRT18] as seen in Equation 3.32.

$$y_i(\boldsymbol{w}\cdot\boldsymbol{x}_i - b) \geq 1 - \xi_i, \quad \forall i \in [0, n-1] \tag{3.31}$$

$$\text{where } \xi_i = \max(0, 1 - y_i(\boldsymbol{w}\cdot\boldsymbol{x}_i - b)) \tag{3.32}$$

The new constraints in Equation 3.31 then transforms Equation 3.30 into the convex optimization problem as seen in Equation 3.33. $\lambda \geq 0$ acts as a weight to control the effect of the slack variables, $\boldsymbol{\xi}$.

$$\begin{aligned} \min_{\boldsymbol{w},b,\boldsymbol{\xi}} \quad & \|\boldsymbol{w}\|^2 + \lambda \sum_{i=0}^{n-1} \xi_i \\ \text{subject to:} \quad & y_i(\boldsymbol{w}\cdot\boldsymbol{x}_i - b) \geq 1 - \xi_i, \quad \forall i \in [0, n-1] \\ \text{where} \quad & \xi_i \geq 0 \text{ and } \boldsymbol{\xi} = \begin{bmatrix} \xi_i \\ \vdots \\ \xi_n \end{bmatrix} \end{aligned} \tag{3.33}$$

Although Equation 3.33 allows for the determination of a linear classifier, a non-linear classifier is required in many other cases. Linear SVMs can be adapted for non-linear classification through the use of the kernel trick[BGV92]. First, data is mapped into a higher dimensional space as usual. Then, it is transformed non-linearly via a kernel so that the resulting support vectors form a linear hyperplane in that non-linear space. This is useful as the hyperplane may turn out to be non-linear in the original space. $\boldsymbol{w}$ is mapped by $\boldsymbol{x}$ as seen in the left of Equation 3.34 where the support vectors consist of some weighted (indicated by a_i) linear combination of kernel transformed $\boldsymbol{x}$. The kernel trick can be applied by adding an extra dot product operation to $\boldsymbol{w}$ giving the right side of Equation 3.34.

$$\boldsymbol{w} = \sum_{i=0}^{n-1} \alpha_i k(\boldsymbol{x}_i, \boldsymbol{x}) \quad \Rightarrow \quad \boldsymbol{w}\cdot\varphi(\boldsymbol{x}) = \sum_{i=0}^{n-1} \alpha_i y_i k(\boldsymbol{x}_i, \boldsymbol{x}) \tag{3.34}$$

$$\text{where } \varphi(\boldsymbol{x}) = \boldsymbol{y}, \quad k(\boldsymbol{x}_i, \boldsymbol{x}_j) = \varphi(\boldsymbol{x}_i)\cdot\varphi(\boldsymbol{x}_j), \quad \forall (i,j) \in [0, n-1]$$

One of the most popular kernels used in SVMs is the Gaussian or radial basis function (RBF) kernel[MRT18]. This is defined in Equation 3.35 and allows for class boundaries that form closed curves or surfaces in the original feature space.

$$k(\boldsymbol{x}_i, \boldsymbol{x}_j) = \exp\left(-\frac{\|\boldsymbol{x}_i - \boldsymbol{x}_j\|^2}{2\sigma^2}\right) \tag{3.35}$$

Even with non-linear kernels, SVM is still a binary classifier. Despite it being possible to formulate multiclass SVM classification as a single optimization problem[MRT18], most implementations overcome this limitation by reducing the multiclass problem to a binary one. By treating all classes except one as a single class, a binary SVM classifier is then trained to discriminate between data in that one class and data from the rest of the other classes hence the name of this method, "one-versus-rest". This results in as many classifiers as there are classes being trained and have an associated confidence score. They are then aggregated into the final model where the individual classifier with the greatest confidence score determines the class of the input data for the multiclass problem.

3.4.2 Feature selection

Before the classifiers in Section 3.4.1 can be utilized, there is a need to select features in the dataset. This is because prediction accuracy can be improved by removing noisy/redundant features. This also reduces the dimensions of the problem and speeds up the training process. Some of the methods to determine which subset of features to use include statistical evaluations such as the chi-squared, χ^2, test and the analysis of variance (ANOVA) F-test. The χ^2 test is among one of the most common statistical metrics[For03]. In a machine learning context, it is used to determine which features are least likely to contribute to classification. It establishes this by evaluating the likelihood of them being class independent. It is defined in Equation 3.36 where N is the number of data samples, K is the number of classes, $O_{i,j}$ is the number of observations of i from class j while $E_{i,j}$ is the expected number of observations of i from class j. A greater χ^2 value means more statistically significant data that would provide more information for a classifier to make a prediction.

$$\chi^2 = \sum_{i=0}^{N-1} \sum_{j=0}^{K-1} \frac{(O_{i,j} - E_{i,j})^2}{E_{i,j}} \tag{3.36}$$

The χ^2 test is, however, meant for frequency and boolean based datasets which makes it incompatible with this thesis' dataset since it contained trajectory coordinate values. The ANOVA F-test[EIO14], on the other hand, was suitable as it would determine the ratio between:

- the variance of the means between classes and
- the variance of the sample means within a class

as seen in Equation 3.37. K is the number of classes, n_i is the number of data points in the i-th class, $\bar{Y}_i$ is the mean of the data in class i, $\bar{Y}$ is the mean of all data in all classes, $Y_{i,j}$ is the j-th data in class i and N is the total number of data points in all classes. This allowed the ANOVA F-test to identify features that were very similar to each other and eliminate them from the dataset. This was helpful as high F value features would ensure each class had very little overlap

in distribution and hence, clearer class boundaries.

$$F = \frac{\sum_{i=0}^{K-1} n_i \frac{(\bar{Y}_i - \bar{Y})^2}{K-1}}{\sum_{i=0}^{K-1} \sum_{j=0}^{n_i} \frac{(Y_{i,j} - \bar{Y}_i)^2}{N-K}} \tag{3.37}$$

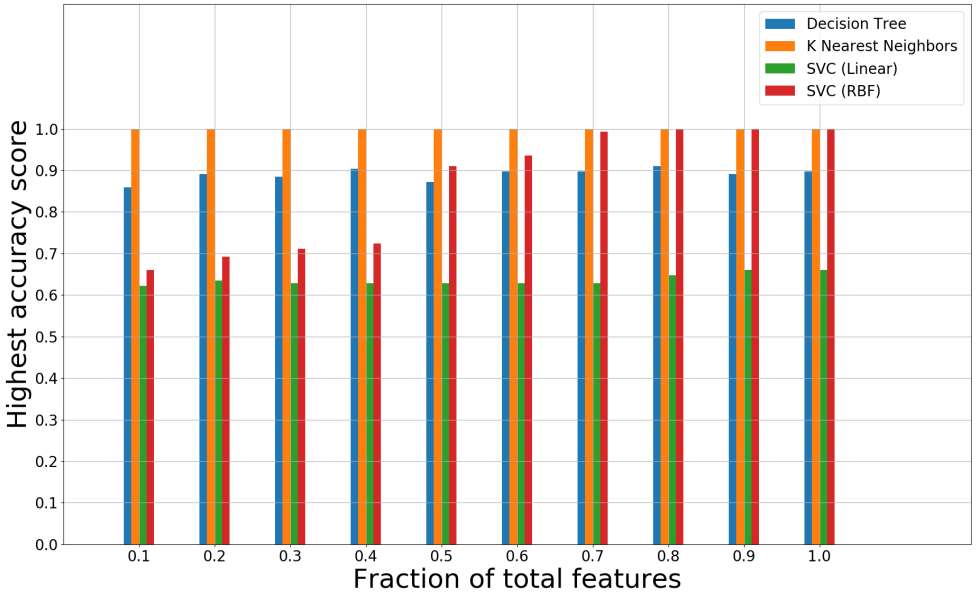


Figure 3.26: Comparison of supervised classifiers without the use of cross-validation on xyz features

3.4.3 Cross-validation

Using the feature selection metric in Equation 3.37, the training half of the trajectory dataset containing xyz coordinates was trimmed. Then, it was utilized to train the classifiers mentioned in Section 3.4.1 which yielded the accuracy scores seen in Figure 3.26. Different fractions (e.g. if the total number of features was 100, then 0.1 would mean only 10 features were being used in training and classifying) as well as a grid (i.e. brute-force) search for hyper-parameter selection[BBBK11] were used. This was because it was uncertain how much the training set should be trimmed by and which hyper-parameters to choose for the classifiers in order to obtain the best results. On first glance, the scores were very worrisome especially for the KNN algorithm as a 100% accuracy score on a dataset as small as this thesis' usually indicated that overfitting had occurred[Alp09]. One way to overcome this was with the use of k-fold cross-validation[Alp09]. Cross-validation is where a training set is split into k equal subsets and $k-1$ subsets are used for training while the subset left out is utilized for testing. This process is repeated k times with a different subset being used for testing each time and the mean accuracy for all folds is then used as the final score. Another

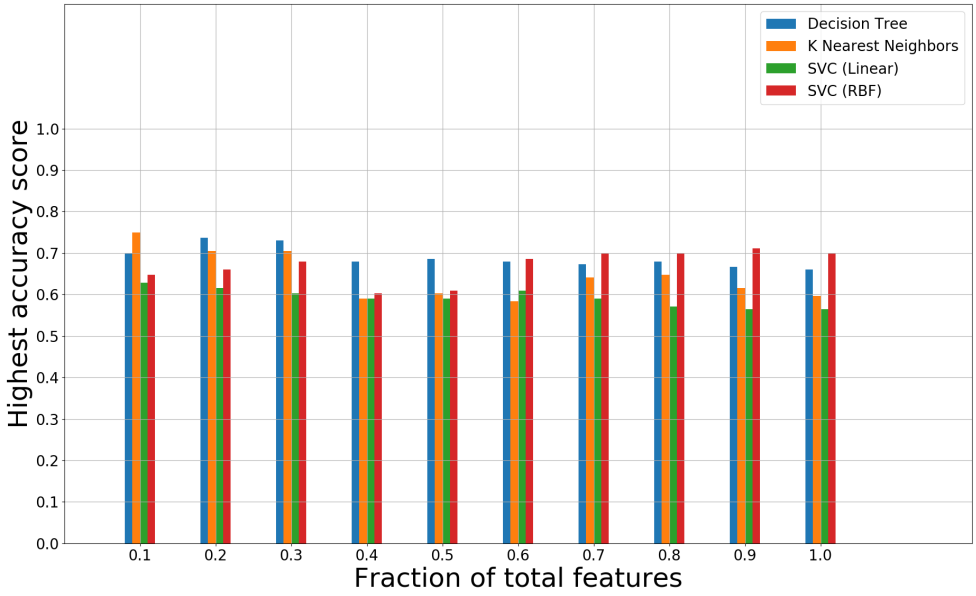


Figure 3.27: Comparison of supervised classifiers with the use of 5-fold stratified cross-validation on xyz features

point to note is that the class distributions are unbalanced as seen in Table 3.1 so stratified splits[Alp09] had to be used. This ensured that the ratio between classes in all k subsets were the same. This resulted in a more general model and realistic scores as seen in Figure 3.27. The decision tree and KNN algorithms performed better on a smaller fraction of ANOVA F-tested features while the RBF SVM did better on a larger fraction.

3.4.4 Feature extraction

From Figure 3.27, it can be seen that the highest score for any one of the algorithms was slightly above 70%. This meant that at least 40 of the 156 training samples were being misclassified. One simple way to boost the accuracy was by the extraction of features as seen in [BKS06]. They determine the view invariant features, CDF and CSS, of their trajectories. The 3D equivalent that was used in this thesis was formulated as seen in Equations 3.38 and 3.39. CDF, d, was determined by computing the distance between all points and the centroid of the trajectory. Curvature, κ, and torsion, τ,[ASG17] was determined using the derivatives of the trajectory.

$$\boldsymbol{d} = \sqrt{(\boldsymbol{x} - x_c)^2 + (\boldsymbol{y} - y_c)^2 + (\boldsymbol{z} - z_c)^2} \tag{3.38}$$

$$\text{where } x_c = \frac{1}{N}\sum_{i=0}^{N-1} x_i, \quad y_c = \frac{1}{N}\sum_{i=0}^{N-1} y_i, \quad z_c = \frac{1}{N}\sum_{i=0}^{N-1} z_i$$

$$\kappa = \frac{\|\dot{\boldsymbol{v}} \times \ddot{\boldsymbol{v}}\|}{\|\dot{\boldsymbol{v}}\|^3}, \quad \tau = \frac{(\dot{\boldsymbol{v}} \times \ddot{\boldsymbol{v}}) \cdot \dddot{\boldsymbol{v}}}{\|\dot{\boldsymbol{v}} \times \ddot{\boldsymbol{v}}\|^2} \quad \text{where } \boldsymbol{v} = \begin{bmatrix} x \\ y \\ z \end{bmatrix} \tag{3.39}$$

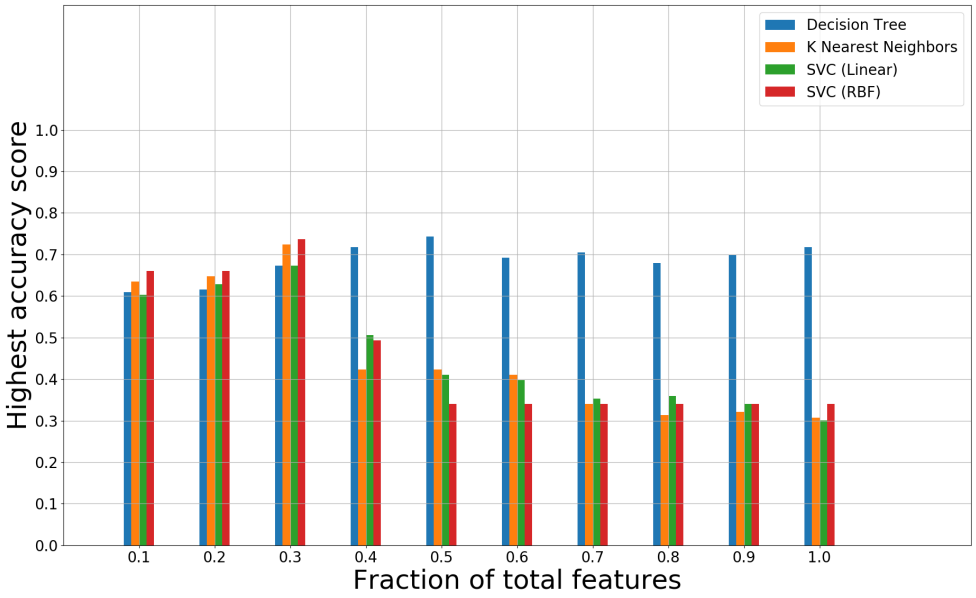


Figure 3.28: Comparison of supervised classifiers with the use of 5-fold stratified cross-validation on CDF, κ and τ features

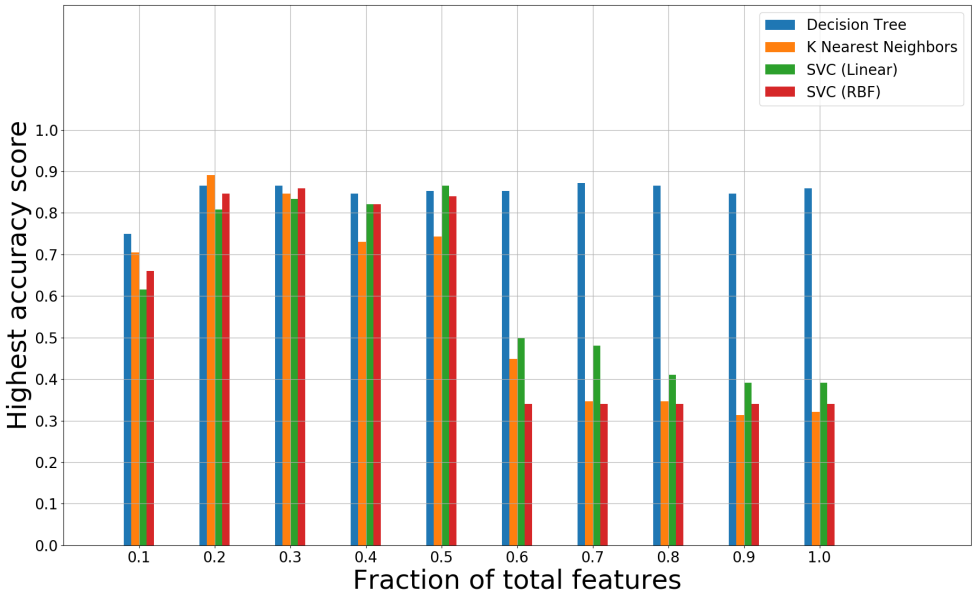


Figure 3.29: Comparison of supervised classifiers with the use of 5-fold stratified cross-validation on xyz, CDF, κ and τ features

With the extracted features, the same training process that yielded Figure 3.27 was applied and this gave Figure 3.28. As it can be seen, there was an obvious decrease in classifier accuracy when a larger fraction of the features were used. This meant that a majority of the information provided by CDF, κ and τ was actually redundant. So, removing them using the ANOVA F-test actually helped classification. Nevertheless, accuracy results were not as high as when only the trajectory xyz coordinates were used for training as seen in Figure 3.28. When CDF, κ and τ were used on top of the xyz features, however, the same training method resulted in Figure 3.29. This time, there was an improvement in accuracy when using the half or less of the total features for all classifiers. The only exception was the decision tree classifier which had almost the same accuracy regardless of the fraction of features used. Although the algorithms all had similar accuracy scores, this did not mean that all of them were misclassifying the same cross-validated samples. As such, a linear combination of each classifier was made. This is known as a voter[Alp09] where each classifier's predictions are taken into account before the final prediction is made. This was similar to the SVM "one-versus-all" approach for multiclass problems. There are two types of voters:

- hard voters which take into account only the number of predictions by multiple classifiers and assigns the final prediction as the prediction with the greatest frequency,
- soft voters which take into account not only the prediction frequency but also the probability of the predictions made by each classifier.

Both were utilized on the final system with the testing half of the trajectory dataset as seen in Chapter 4.

Chapter 4

Experimental Results

4.1 Final system architecture

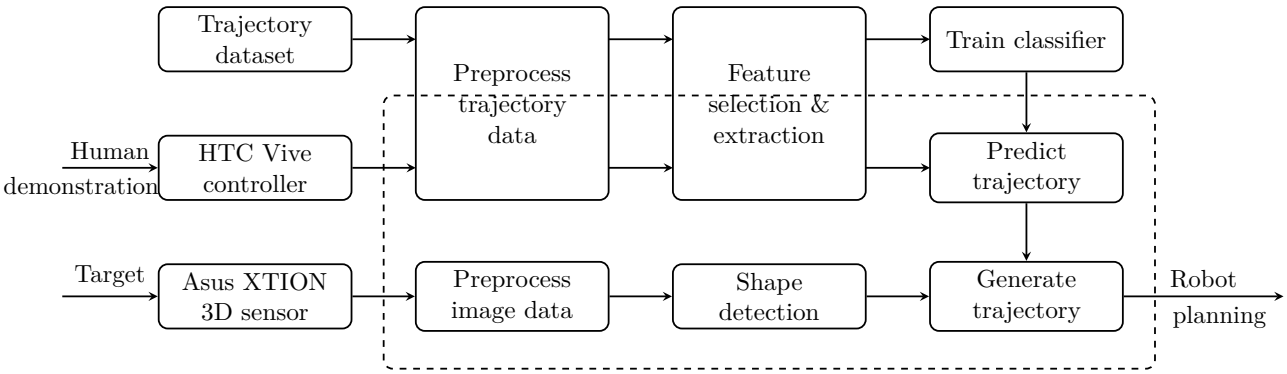


Figure 4.1: Diagram of final system developed by this thesis

With all the methods detailed in Chapter 3, the final system developed by this thesis is structured as seen in Figure 4.1. The dotted box represents the parts of the system where quick computation would be crucial as an industrial level implementation would ideally not require the user to wait for long periods of time. Classifier training, on the other hand, could take much longer as models could be pre-trained before being deployed.

4.2 Classifier results

The classifiers in Section 3.4.1 were bundled into two voters as each voter should ideally have an odd number of classifiers to prevent a tie. The weakest of the 4 algorithms was determined from Figure 3.29 for specific fractions of features and eliminated.

1. (Decision tree + KNN + RBF SVM) for 0.2 and 0.3 of total features
2. (Decision tree + linear SVM + RBF SVM) for 0.3 and 0.5 of total features

Both voters were then used on the testing half of the trajectory dataset which yielded Figures 4.2 and 4.3. It can be seen that 1. has consistently greater

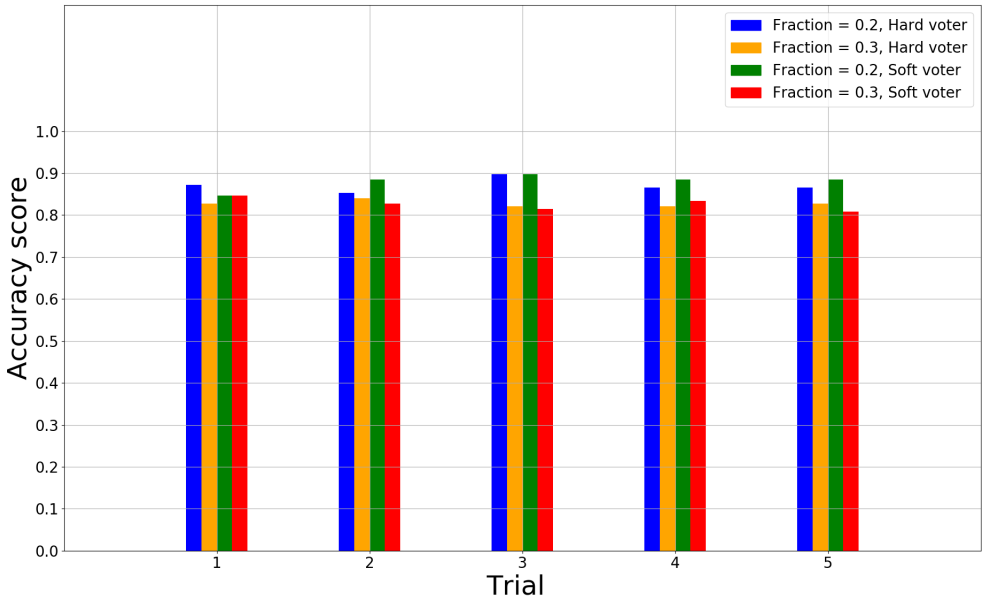


Figure 4.2: Accuracy scores for voter 1.

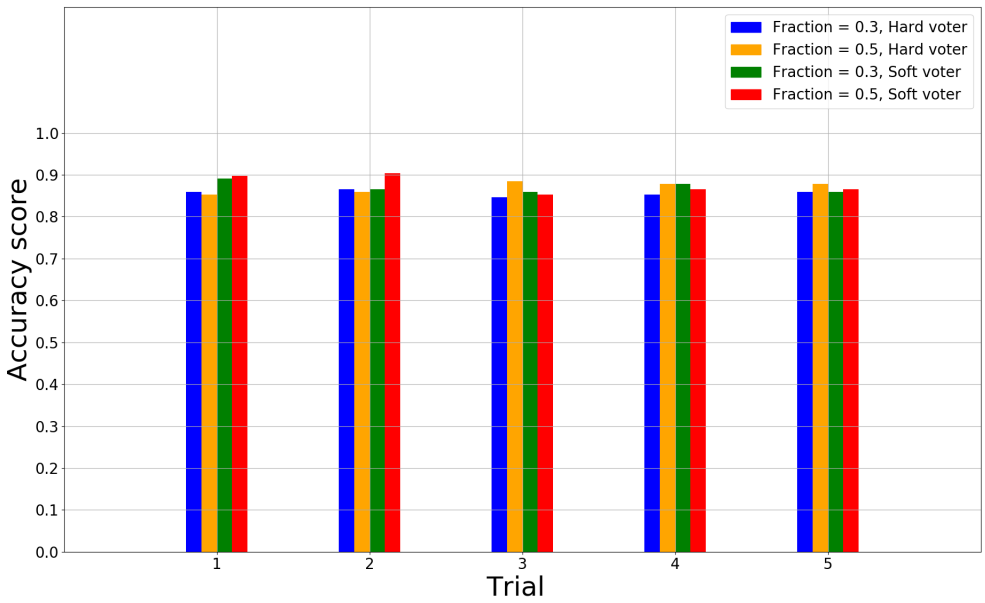
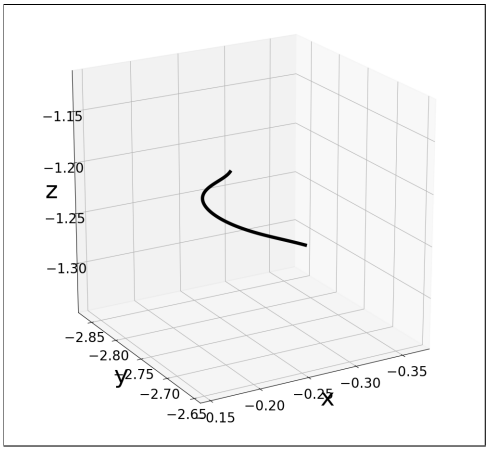


Figure 4.3: Accuracy scores for voter 2.

accuracy when the xyz, CDF, κ and τ features are trimmed down to 20% of their original numbers by the ANOVA F-test whereas 2. has similar accuracy regardless of whether feature selection reduced the features to 30% and 50% of their original number. Another point to note is that the choice of either voter did not significantly impact the accuracy scores and as such, 1. with hard voting and a fraction of 0.2 was used to generate the results in Section 4.3.

4.3 NPT prediction and generation



(a) : Line trajectory made by HTC Vive controller used as input to system

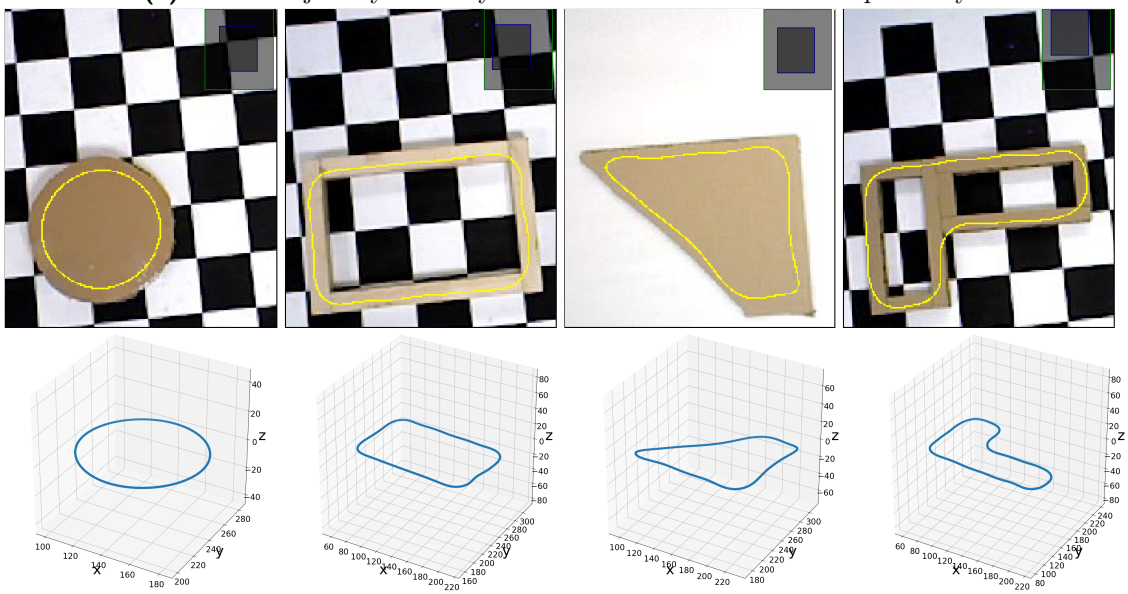
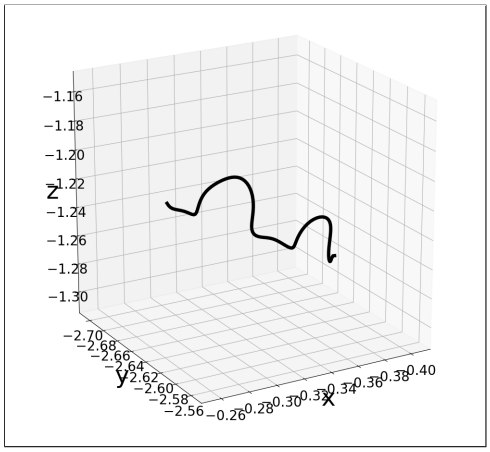


Figure 4.4: Trajectory outputs of system for line input 4.4a in 2D and 3D for blocks with different shape



(a) : Dashed trajectory made by HTC Vive controller used as input to system

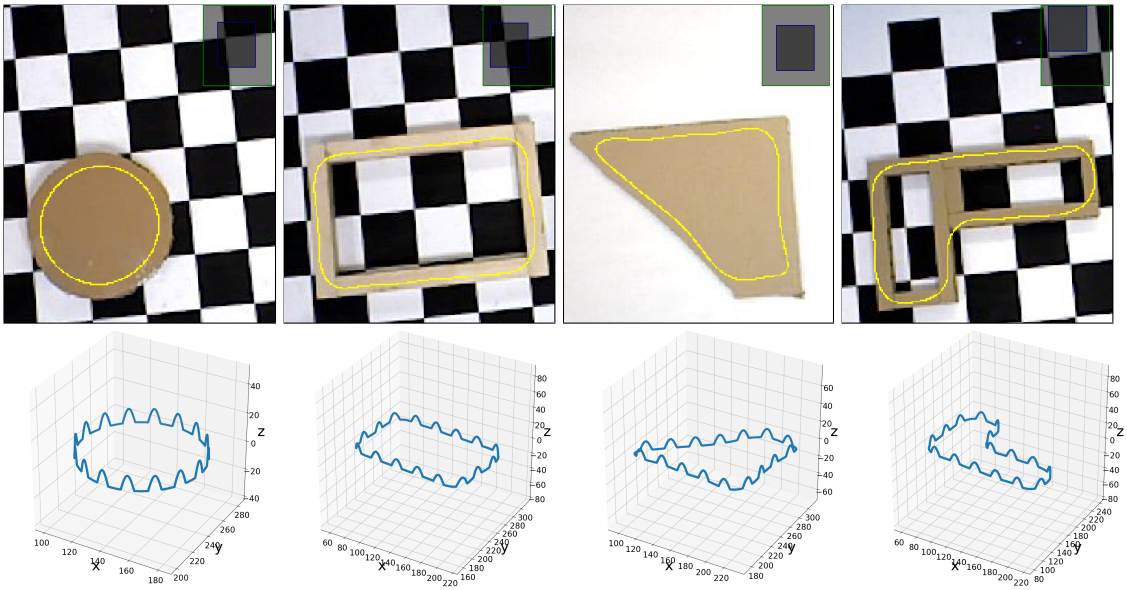


Figure 4.5: Trajectory outputs of system for dashed input 4.5a in 2D and 3D for blocks with different shapes

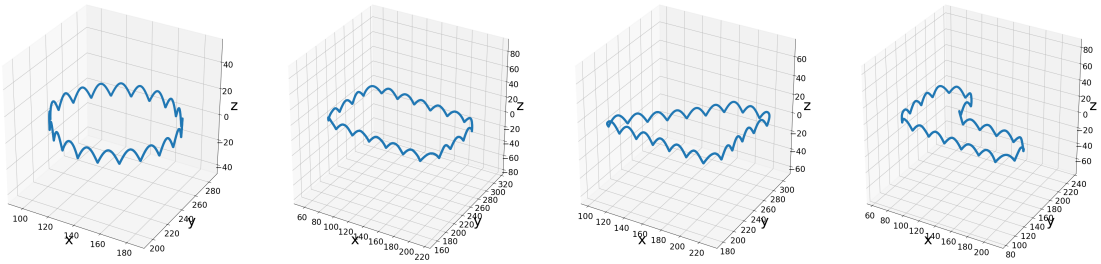
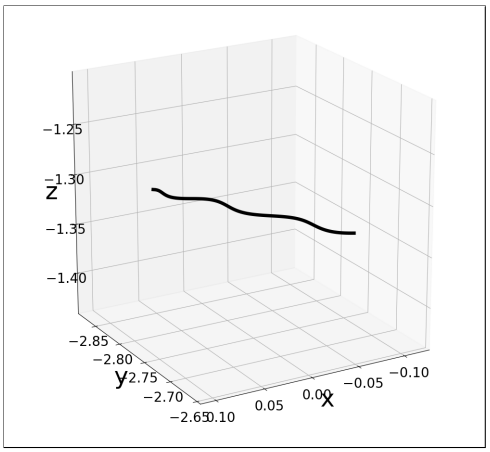


Figure 4.6: Example of dotted NPTs for blocks with different shapes



(a) : Low frequency sine trajectory made by HTC Vive controller used as input to system

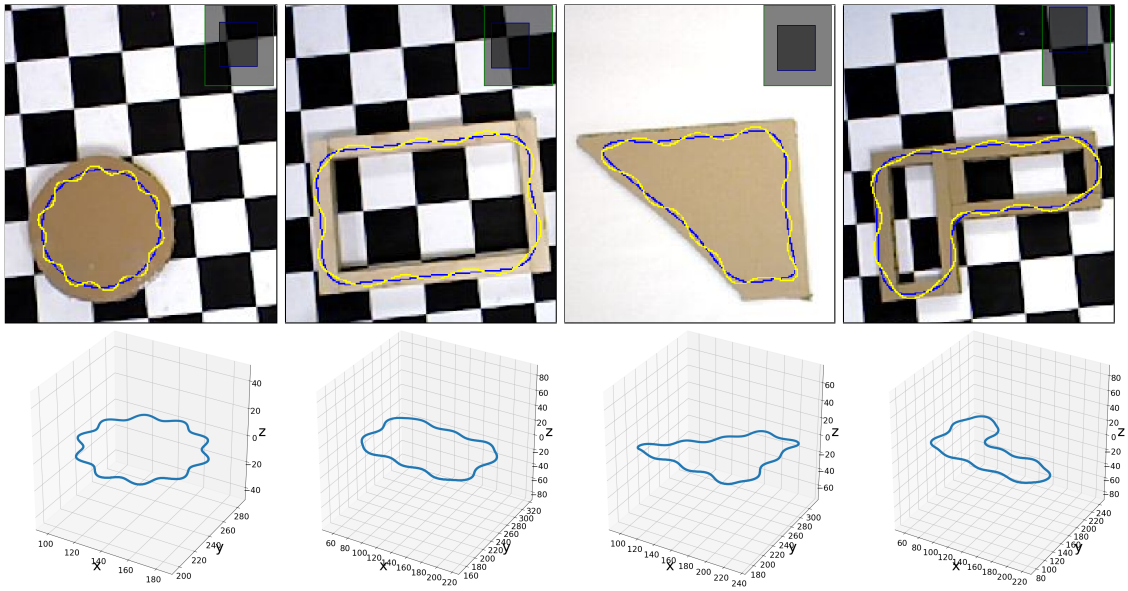


Figure 4.7: Trajectory outputs of system for low frequency wave input 4.7a in 2D and 3D for blocks with different shapes

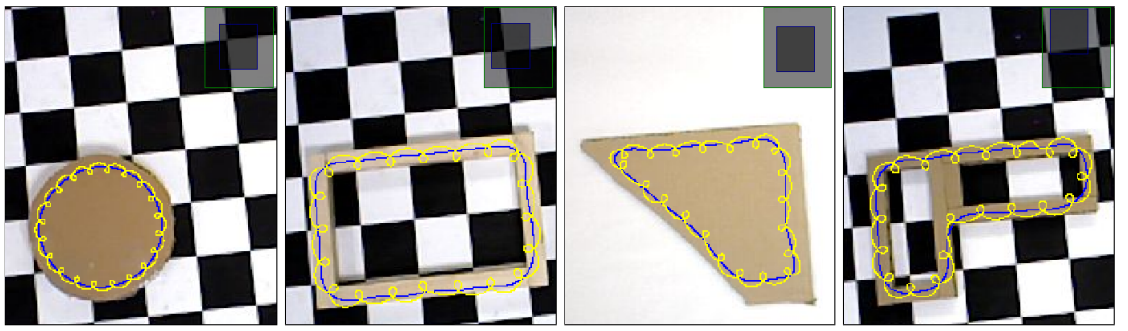
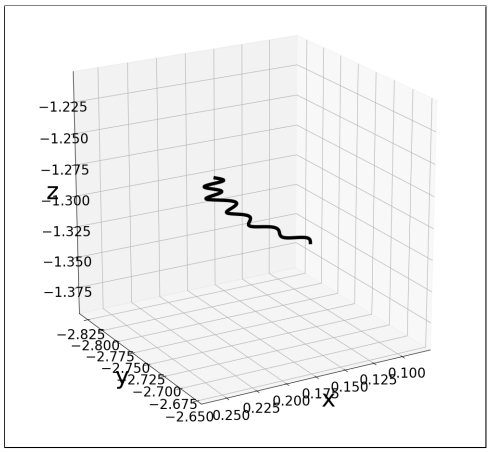


Figure 4.8: Example of inward conchoid NPTs for blocks with different shapes



(a) : High frequency sine trajectory made by HTC Vive controller used as input to system

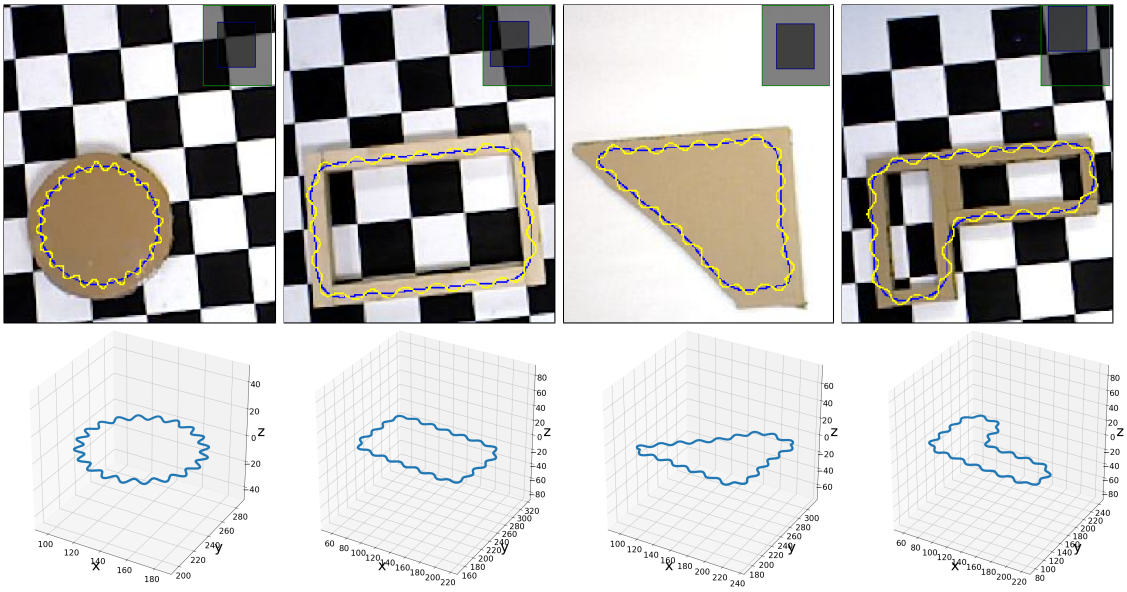


Figure 4.9: Trajectory outputs of system for high frequency wave input 4.9a in 2D and 3D for blocks with different shapes

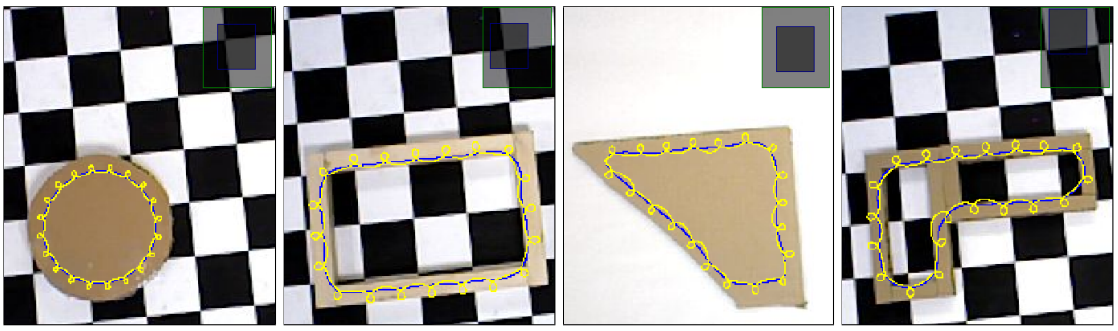


Figure 4.10: Example of outward conchoid NPTs for blocks with different shapes

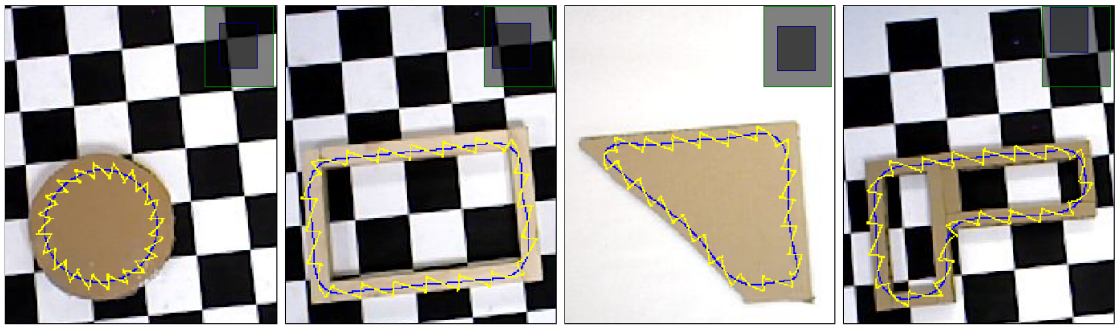


Figure 4.11: Example of sawtooth NPTs for blocks with different shapes

A selection of individual trajectory segments from the testing half of the dataset was visualized in Figures 4.4a, 4.5a, 4.7a and 4.9a. They were used as inputs to the system in 4.1 and their corresponding outputs were determined by voter 1. . The voter identified the input trajectory's class and chose the most similar NPT to superimpose over the shape detection results. This yielded Figures 4.4, 4.5, 4.7 and 4.9. Although the system should be capable of classifying more different types of NPTs, there was not enough time to gather more data for training the classifiers. Thus, only their trajectories were generated as shown in Figures 4.6, 4.8, 4.10 and 4.11. The triangular shape for all experiments had a blank background as it was too thin for the Asus XTION PRO's depth sensor to detect. This meant that the technique detailed in Section 3.2.4 was not applicable and only its RGB image was utilized. This limitation will be discussed further in Chapter 5.

Chapter 5

Discussion and Conclusion

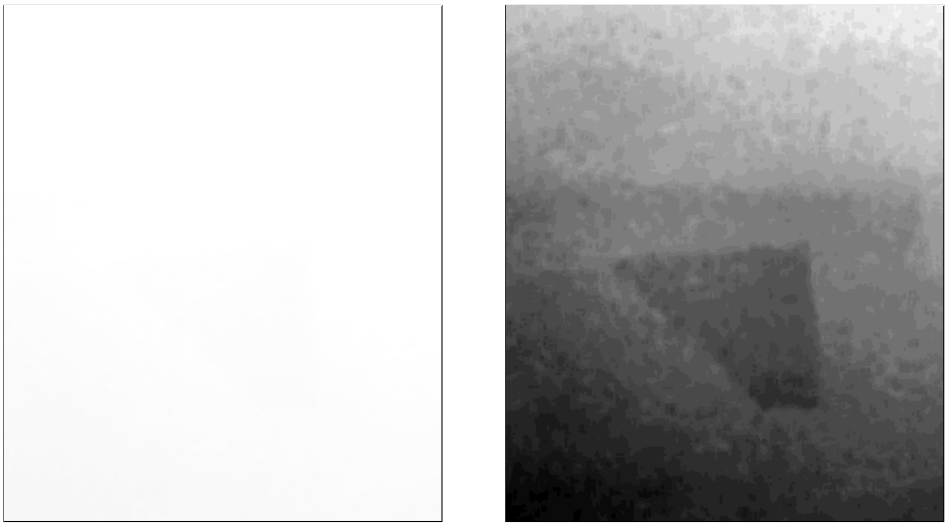
As seen in the cases of dashed and dotted trajectories, there is no difference in the shape of the trajectory from the top-down perspective as only the height values change over time. The sawtooth trajectory seen in 4.11 was generated using the concept behind Equation 3.13 with the appropriate Fourier coefficients found in Table 3.3. This reinforces the point that generating trajectories of naturally periodic functions is trivial unlike those seen in Figures 4.8 and 4.10. Those required the approach discussed in Section 3.3.3.

Based on Figure 4.1, shape recognition and NPT generation within the system can be observed to be decoupled. This has the advantage of the fact that the trajectory generated will not be drastically different from the shape of the target even if misclassification occurred. Thus, this makes the system more predictable/safer. This decoupling was also necessary otherwise 'autocompletion' of the final trajectory would be far more difficult. This is due to the fact that the shape of the target would have to be deduced from the input trajectory segments in Section 3.1.3. Another benefit to this decoupling is that user made trajectories would not be constrained to being performed on the surface of the target (which is the case for this thesis). They could have been done further away from the robot which would increase safety.

According to Section 4.2, classification accuracy was on average greater than 85%. This meant that at least 20 of the 156 testing samples were being misclassified i.e. half the rate associated with the training seen in Figure 3.27. Also, utilization of a hard or soft voter structure did not have any significant effect as mentioned in Section 4.2. Although one could make the argument that the linear combination of multiple classifiers would allow for the check and balance of overfitted models, more experimentation needs to be done before this can be ascertained.

Some of the issues with the system developed by this thesis includes the bias in the trajectory dataset used to train the supervised classifiers. As mentioned earlier, most of the trajectory data was performed on the surface of the targets. Thus, there was most likely a very high correlation between the xyz trajectory coordinates of each data sample. This would explain why initial training with those features resulted in such high accuracy values as seen in Figure 3.27.

Without further experimentation, it is uncertain whether the features and training methods used in Section 3.4 would yield results as high as Section 4.2 for more general cases i.e. when input trajectories are performed in different areas of the coordinate space.



(a) : Depth image for triangular target

(b) : Grayscale version of depth image for triangular target

Figure 5.1: Example of unresolvable depth image for a thin block

Another issue is the limitation of the RGB and depth image fusion to remove the checkerboard pattern for shape recognition. The checkerboard pattern had to be replaced as mentioned in Section 4.3 where all the triangular targets have a plain background. This was due to the target being too thin (less than 1cm) which resulted in a grayscale depth image that was too noisy for Canny edge detection to pick up its outline. As seen in Figure 5.1b, the boundaries of the target were less distinct compared to that in Figure 3.5b. One way to overcome this constraint is by elevating the target by placing something else smaller underneath it. This, however, introduces complications such as the target tipping or sliding off the object elevating it. There is also the option using a sensor with greater depth resolution but this may not be a cost-effective approach.

From an operator's standpoint, the scaling process utilized by this thesis' system is also another issue as it is controlled by a single value (e.g. 0.1 would scale the contour down to 10% of its original size). This means that fine control of the trajectory's perpendicular distance from the target's edges (e.g. 5mm from edge etc.) is not possible. As a result, trial and error is usually required to determine the user's desired scaling factor. This is further complicated by the fact that it is difficult to visually determine the exact distance in images since they are defined by discrete pixels not continuous values.

There is also the issue of whether the 5th order B-splines used to generate the trajectories seen in Section 4.3 map to suitable joint position, velocity, acceleration

and jerk profiles for an industrial robot. Although the use of B-splines in the NPTs allow for continuity in higher order dimensions, this is only for the space where the trajectory resides. This means that within the joint space of the robot, there is no guarantee that there will be no sudden changes in joint position, velocity, acceleration and/or jerk. This is because the discrete points on the B-splines are spaced equidistantly regardless of the physical limits of an industrial robot. Thus, more study is required to determine how to best incorporate the real-world limits of industrial robotic systems into the trajectories generated by the system developed by this thesis.

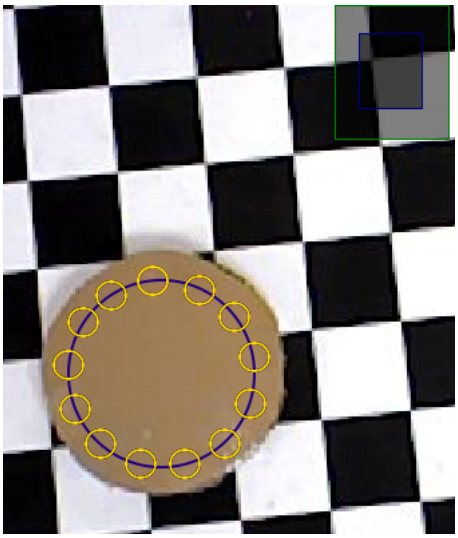


Figure 5.2: Example of NPT that system 4.1 cannot generate

Although the approach in Section 3.3.3 yielded many appropriate looking 'ideal' versions of NPTs as seen in Section 4.3, the method was limited to open curves. This is due to the fact that the approximation line (pink) depended on the endpoints of the complex curves as seen in Figure 3.21. As such, for trajectories similar to that seen in Figure 5.2, this approach would not be applicable as a circle starts and ends at the same position. Another technique would be required to take into account the entire curve regardless of where its endpoints are. On top of that, there would also be the issue of determining the profile of the height values as the trajectory moves between the yellow circles. It would definitely not be as straightforward as the case of dashed and dotted trajectories.

In addition, the assumption of a constant height for most of the trajectories generated in this thesis has its downsides. One of the reasons this assumption is valid is that the glue from a glue gun is viscous which means it will drop downwards. Also, the best way to avoid object collision is for the user to set a trajectory height just slightly above the target's surface. Nevertheless, relying on a human operator means that the chance of human error is increased[KC].

Finally, the prediction method used for this thesis' system does not take into account the case where a user is demonstrating a completely new pattern/complex

curve unrelated to all the data used for training the classifier. One possible way of dealing with such a scenario is with the use of prediction probabilities which quantify the confidence level of the classifier. If the confidence level is below a certain threshold, some 3D to 2D transform could be applied to the new input pattern. Then, the approach in Section 3.3.3 could be utilized to create an entirely new NPT. This idea, however, was not explored in this thesis as there was not enough time to collect the relevant input trajectory data.

Although the system developed by this thesis was not perfect, it has set the groundwork for the further development of 'autocompleting' human demonstrated NPTs that can be used for motion planning in an industrial setting. A list containing the many improvements that can be made to adapt the system for a larger variety of scenarios is detailed in Chapter 6.

Chapter 6

Future work

There are a large number of improvements and additions on top of those mentioned in Chapter 5 that can be made to this thesis' system in order to allow it to encompass a greater assortment of conditions associated with automatic glue application by an industrial robot. With some reiteration, the following list contains descriptions and approaches to some of the work that can further what this thesis has done.

- Decrease the amount of bias in the trajectory dataset used to train the supervised classifiers by collecting trajectory data performed in different areas of the coordinate space (i.e. not on the surface of the target block) in order to allow for a more general classification of NPT types.
- Improve upon this thesis' RGB and depth image fusion to detect even thinner targets on a checkerboard pattern. As the checkerboard is necessary only for calibrating the HTC Vive system, it does not have to be part of the image being used for shape recognition. Nevertheless, it would be more convenient to not have to change backgrounds. One possible approach to enhancing the outline of a thin target is by subtracting the values in the depth image when nothing is present from the depth image containing the target. This idea was not pursued by this thesis due to the experimental area being modified after the dataset in Section 3.1.1 was created making it difficult to recreate the original experimental setup.
- Improve on the precision of the scaling approach used by this thesis' system so that specific perpendicular distances between target edges and generated trajectory can be specified.
- Incorporate the limits of an industrial robot when generating NPTs to ensure smooth and appropriate position, velocity, acceleration and jerk profiles in joint space.
- Improve the methodology used to generate the approximate line (pink) for complex curves as seen in Figure 3.21 so that it encompasses a larger variety of open and closed complex curves.
- Automate the generation of trajectory height to reduce the chance of human error. This could be done with the use of a 2nd sensor placed to the side

of the target so that a profile of its height can be determined via edge detection.

- Improve the classification algorithm to take into account prediction probabilities. This will allow newly introduced user made trajectories which are classified with low probabilities to leverage on the method in Section 3.3.3 for generating new NPTs.
- Add an on-off profile that synchronizes the dispensing of glue from the glue gun with the NPT so that glue is not applied on undesired locations such as between dashes or dots.

Finally, work on intergrating the system developed in this thesis with an actual industrial robot system should be explored. Due to the lack of time, it was not possible to test out the performance of path/trajectory planning by an actual industrial robot using the trajectories generated by this thesis' system. Nevertheless, it is acknowledged that the space in which the NPTs were generated does not necessarily correspond to the right area in the robot's workspace so there would be a need to determine the correct transformation matrix either from camera or HTC Vive calibration matrices. This and the generation of an appropriate joint position, velocity, acceleration and jerk profile would be most key to allowing for the demonstration of NPTs in an industrial setting to be a success.

Appendix A

Algorithms

Algorithm 1 Image preprocessing and shape recognition

Load images i_{rgb} and i_{depth} from rosbag files and store in dictionary
for (i_{rgb}, i_{depth}) in dictionary **do**
    Crop and remove NaNs from (i_{rgb}, i_{depth}) to give (n_{rgb}, n_{depth})
    Convert (n_{rgb}, n_{depth}) into grayscale, (g_{rgb}, g_{depth})
    Apply Canny edge detection to g_{rgb} to give c_{rgb}
    Apply morphological dilation to c_{rgb} to give d_{rgb}
    Determine contours in d_{rgb} as t_{rgb}
    if No. of objects derived from t_{rgb} ! = user given value **then**
        Apply Gaussian filter to i_{depth} to give s_{depth}
        Apply Canny edge detection to s_{depth} to give c_{depth}
        Apply morphological dilation to c_{depth} to give d_{depth}
        Apply d_{depth} as mask onto c_{rgb} to give c_{combi}
        Determine contours in c_{combi} as t_{combi}
        $j = 0$
        while No. of objects from t_{combi} ! = user given value **do**
            $j++$
            if $j >$ user specified max **then**
                break
            end if
            for k from $0 \rightarrow j$ **do**
                Apply morphological closing to c_{combi} to give l_{combi}
            end for
            Determine contours in l_{combi} as t_{combi}
        end while
        Apply morphological erosion to l_{combi} to give e_{combi}
        Apply median blurring to e_{combi} to give m_{combi}
        Determine contours in m_{combi} as t_{combi}
    end if
    Store t_{combi} in dictionary with associated (i_{rgb}, i_{depth})
end for
return dictionary

Algorithm 2 Trajectory preprocessing and classification

Load trajectory coordinate data t_{xyz} from rosbag
for t_i in t_{xyz} **do**
    Segment by length to give $t_i = [s_0, s_1, \ldots, s_n]$
    for s_j in t_i **do**
        Convert s_j to a B-spline b_j with constant number of points
        Compute CDF, κ and τ of b_j
        Assign label/class of segment as i
        Store (b_j, CDF, κ, τ, label) in dictionary d_t
    end for
end for
Initialize classifiers, $clf = [c_0, c_1, \ldots, c_n]$
Initialize relevant hyper-parameters, $p = [p_0, p_1, \ldots, p_n]$
Select best features in d_t using ANOVA F-test to give d_{slctd}
for c_i in clf **do**
    Set c_i to use 5-fold CV and p_i for grid search
    Train c_i using d_{slctd} and determine best parameters $p_{i,b}$
    Set c_i parameters to $p_{i,b}$ to give $c_{i,b}$
    Assign $c_{i,b}$ as an estimator in voter
end for
return voter

Algorithm 3 NPT generation

Load contour t_{img}
Load voter v
Load user demonstrated trajectory u
Convert t_{img} into B-spline b with n points
Predict trajectory type y using v on u
if y can be modelled using a mathematically periodic function **then**
    Compute NPT waypoints, w, using appropriate function
else
    Calculate a single period, r, of NPT using appropriate function
    Convert r into B-spline b_r with m points
    Determine approximation B-spline a, also with m points, of b_r
    Initialize empty w
    for j from $0 \rightarrow \frac{n}{m} - 1$ **do**
        $s = b_i[(j * m) : ((j + 1) * m) - 1]$
        Find the similarity transformation matrix M between a and s
        Apply M to r and append result to w
    end for
end if
Apply low-pass filter to w to give w_{low}
Convert w_{low} into a NPT B-spline with $2 * n$ points
return NPT B-spline

Appendix B

Glossary

Symbol/Abbreviation	Meaning
ANOVA	ANalysis Of VAriance
ASL	Australian Sign Language
CDF	Centroid Distance Function
CSS	Curvature State Space
dof	degrees of freedom
HMM	Hidden Markov Models
KNN	K-Nearest Neighbors
MRF	Markov Random Field
NaN	Not a Number
NPR	Non-Photorealistic Rendering
NPT	Non-Photorealistic Trajectory
RBF	Radial Basis Function
RGB	Red, Green, Blue
ROS	Robot Operating System
SVM	Support Vector Machine
C^m	Curve continuity to the m-th order
κ	Curvature
τ	Torsion
χ^2	Chi-squared
$\oplus$	Morphological dilation
$\ominus$	Morphological erosion
$\circ$	Morphological opening
$\bullet$	Morphological closing

Appendix C

Bibliography

[AK03] Fredrik Andersson and Berit Kvernes, *Bezier and b-spline technology*, Umea university Sweden (2003).

[Alp09] Ethem Alpaydin, *Introduction to machine learning*, MIT press, 2009.

[AMFM11] Pablo Arbelaez, Michael Maire, Charless Fowlkes, and Jitendra Malik, *Contour detection and hierarchical image segmentation*, IEEE transactions on pattern analysis and machine intelligence **33** (2011), no. 5, 898–916.

[ANSL03] S Amin-Nejad, JS Smith, and J Lucas, *A visual servoing system for edge trimming of fabric embroideries by laser*, Mechatronics **13** (2003), no. 6, 533–551.

[ASG17] Elsa Abbena, Simon Salamon, and Alfred Gray, *Modern differential geometry of curves and surfaces with mathematica*, Chapman and Hall/CRC, 2017.

[BBBK11] James S Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl, *Algorithms for hyper-parameter optimization*, Advances in neural information processing systems, 2011, pp. 2546–2554.

[BCDS08] Aude Billard, Sylvain Calinon, Ruediger Dillmann, and Stefan Schaal, *Robot programming by demonstration*, Springer handbook of robotics (2008), 1371–1394.

[Bel59] William A Belson, *Matching and prediction on the principle of biological classification*, Journal of the Royal Statistical Society: Series C (Applied Statistics) **8** (1959), no. 2, 65–75.

[BGV92] Bernhard E Boser, Isabelle M Guyon, and Vladimir N Vapnik, *A training algorithm for optimal margin classifiers*, Proceedings of the fifth annual workshop on Computational learning theory, ACM, 1992, pp. 144–152.

[BKS06] Faisal I Bashir, Ashfaq A Khokhar, and Dan Schonfeld, *View-invariant motion trajectory-based activity classification and recognition*, Multimedia Systems **12** (2006), no. 1, 45–54.

[BSS07] Meru Brunn, Mario Costa Sousa, and Faramarz F Samavati, *Capturing and re-using artistic styles with reverse subdivision-based multiresolution methods*, International Journal of Image and Graphics **7** (2007), no. 04, 593–615.

[Can87] John Canny, *A computational approach to edge detection*, Readings in computer vision, Elsevier, 1987, pp. 184–203.

[CH+67] Thomas M Cover, Peter E Hart, et al., *Nearest neighbor pattern classification*, IEEE transactions on information theory **13** (1967), no. 1, 21–27.

[CV95] Corinna Cortes and Vladimir Vapnik, *Support-vector networks*, Machine learning **20** (1995), no. 3, 273–297.

[Die95] Paul Dierckx, *Curve and surface fitting with splines*, Oxford University Press, 1995.

[EIO14] Nadir Omer Fadl Elssied, Othman Ibrahim, and Ahmed Hamza Osman, *Research article a novel feature selection based on one-way anova f-test for e-mail spam classification*, Research Journal of Applied Sciences, Engineering and Technology **7** (2014), no. 3, 625–638.

[FD09] Diego R Faria and Jorge Dias, *3d hand trajectory segmentation by curvatures and hand orientation for classification through a probabilistic approach*, 2009 IEEE/RSJ International Conference on Intelligent Robots and Systems, IEEE, 2009, pp. 1284–1289.

[For03] George Forman, *An extensive empirical study of feature selection metrics for text classification*, Journal of machine learning research **3** (2003), no. Mar, 1289–1305.

[FR99] Eckhard Freund and Juergen Rossmann, *Projective virtual reality: Bridging the gap between virtual reality and robotics*, IEEE Transactions on Robotics and Automation **15** (1999), no. 3, 411–422.

[FS94] Adam Finkelstein and David H Salesin, *Multiresolution curves*, Proceedings of the 21st annual conference on Computer graphics and interactive techniques, ACM, 1994, pp. 261–268.

[GMP05] Peter D Grünwald, In Jae Myung, and Mark A Pitt, *Advances in minimum description length: Theory and applications*, MIT press, 2005.

[GVPS12] Alessandro Gasparetto, Renato Vidoni, Daniele Pillan, and Ennio Saccavini, *Automatic path and trajectory planning for robotic spray painting*, ROBOTIK 2012; 7th German Conference on Robotics, VDE, 2012, pp. 1–6.

[Han14] Eric W Hansen, *Fourier transforms: principles and applications*, John Wiley & Sons, 2014.

[HOCS02] Aaron Hertzmann, Nuria Oliver, Brian Curless, and Steven M Seitz, *Curve analogies.*, Rendering Techniques, 2002, pp. 233–246.

[HZ03] Richard Hartley and Andrew Zisserman, *Multiple view geometry in computer vision*, Cambridge university press, 2003.

[KC] Joonyoung Kim and Elizabeth A Croft, *Trajectory planning for robots: the challenges of industrial considerations.*

[KC05] Dana Kulić and Elizabeth A Croft, *Safe planning for human-robot interaction*, Journal of Robotic Systems **22** (2005), no. 7, 383–396.

[KMM+02] Robert D Kalnins, Lee Markosian, Barbara J Meier, Michael A Kowalski, Joseph C Lee, Philip L Davidson, Matthew Webb, John F Hughes, and Adam Finkelstein, *Wysiwyg npr: Drawing strokes directly on 3d models*, ACM Transactions on Graphics (TOG), vol. 21, ACM, 2002, pp. 755–762.

[LaV06] Steven M LaValle, *Planning algorithms*, Cambridge university press, 2006.

[LHLG08] Jae-Gil Lee, Jiawei Han, Xiaolei Li, and Hector Gonzalez, *Traclass: trajectory classification using hierarchical region-based and trajectory-based clustering*, Proceedings of the VLDB Endowment **1** (2008), no. 1, 1081–1094.

[LKP07] Chuanjun Li, Latifur Khan, and Balakrishnan Prabhakaran, *Feature selection for classification of variable length multiattribute motions*, Multimedia data mining and knowledge discovery, Springer, 2007, pp. 116–137.

[LMPD15] Thomas Lindemeier, Jens Metzner, Lena Pollak, and Oliver Deussen, *Hardware-based non-photorealistic rendering using a painting robot*, Computer graphics forum, vol. 34, Wiley Online Library, 2015, pp. 311–323.

[LP17] Kevin M Lynch and Frank C Park, *Modern robotics*, Cambridge University Press, 2017.

[MPW14] Eka Samsul Maarif, Endra Pitowarno, and Rusminto Tjatur Widodo, *A trajectory generation method based on edge detection for auto-sealant cartesian robot*, Journal of Mechatronics, Electrical Power, and Vehicular Technology **5** (2014), no. 1, 27–36.

[MRT18] Mehryar Mohri, Afshin Rostamizadeh, and Ameet Talwalkar, *Foundations of machine learning*, MIT press, 2018.

[NA12] Mark Nixon and Alberto S Aguado, *Feature extraction and image processing for computer vision*, Academic Press, 2012.

[NLL17] Diederick C Niehorster, Li Li, and Markus Lappe, *The accuracy and precision of position and orientation tracking in the htc vive virtual reality system for scientific research*, i-Perception **8** (2017), no. 3, 2041669517708205.

[OGL08] Alex Olwal, Jonny Gustafsson, and Christoffer Lindfors, *Spatial augmented reality on industrial cnc-machines*, The Engineering Reality of Virtual Reality 2008, vol. 6804, International Society for Optics and Photonics, 2008, p. 680409.

[PMF08] Claudio Piciarelli, Christian Micheloni, and Gian Luca Foresti, *Trajectory-based anomalous event detection*, IEEE Transactions on Circuits and Systems for video Technology **18** (2008), no. 11, 1544–1554.

[S+85] Satoshi Suzuki et al., *Topological structural analysis of digitized binary images by border following*, Computer vision, graphics, and image processing **30** (1985), no. 1, 32–46.

[SD04] Saul Simhon and Gregory Dudek, *Sketch interpretation and refinement using statistical models.*, Rendering Techniques, 2004, pp. 23–32.

[SL17] Ching-Long Shih and Li-Chen Lin, *Trajectory planning and tracking control of a differential-drive mobile robot in a picture drawing application*, Robotics **6** (2017), no. 3, 17.

[SS02] Thomas Strothotte and Stefan Schlechtweg, *Non-photorealistic computer graphics: modeling, rendering, and animation*, Morgan Kaufmann, 2002.

[SSGG19] Lorenzo Scalera, Stefano Seriani, Alessandro Gasparetto, and Paolo Gallina, *Non-photorealistic rendering techniques for artistic robotic painting*, Robotics **8** (2019), no. 1, 10.

[Sze10] Richard Szeliski, *Computer vision: Algorithms and applications*, 1st ed., Springer-Verlag, Berlin, Heidelberg, 2010.

[WR00] Joseph N Wilson and Gerhard X Ritter, *Handbook of computer vision algorithms in image algebra*, CRC press, 2000.