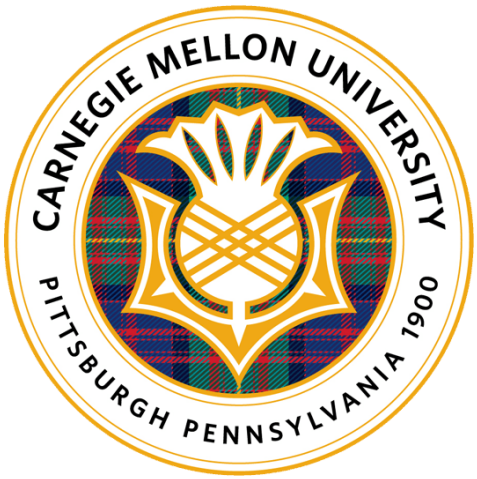
# Leveraging Multimodal Sensory Data for Robust Cutting

Kevin Zhang

CMU-RI-TR-19-64
August 2019




The Robotics Institute
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Thesis Committee:
Manuela M. Veloso, Co-Chair
Oliver Kroemer, Co-Chair
George Kantor
Anahita Mohseni-Kabir

*Submitted in partial fulfillment of the requirements for the degree of Master of Science in Robotics.*

# Abstract

Cutting food is a challenging task due to the variety of material properties across food items. In addition, different events may occur while executing cutting actions which need to be detected for proper skill execution and termination. Due to occlusions, it is often difficult for solely vision to solve both these problems. However, by utilizing vibration feedback from contact microphones and robot force data, we can classify the toughness of an object as well as monitor the status of the robot while slicing. We show that by leveraging this information, the robot is able to adapt its cutting technique according to the material, which results in more robust cutting.

In this paper, we will cover our entire slicing robot pipeline from the ground up. We will introduce the entire experimental setup, a new robot interface that allows us to control the robot and teach it new skills such as the slicing skill, the vision system for picking up and placing objects onto the cutting board, and finally the multimodal system for monitoring the robot and classifying the toughness of objects while slicing, which enables the robot to adapt it slicing motion to the material.

# Acknowledgements

I would like to first thank both of my co-advisors, Professor Manuela Veloso and Professor Oliver Kroemer, for continuously providing me with guidance for my research as well as the opportunity to work on an exciting project that I am passionate about. I first met Manuela as a cheeky Junior in March of 2017 wanting to put some arms on CoBot for RoboCup at Home. Later that day, I ended up joining the RoboCup at Home team and had the privilege to work with Pepper for 2 years. Then one day, I was chatting with Manuela about my steak cooking robot for my robotics capstone project when she told me that there would be an opportunity to work on cooking robots with Sony in the future. I was completely surprised and ecstatic of the fortunate timing of the research project. Then I met Oliver a few times in the elevator over the summer and at the Sony cooking project kickoff meeting. I was glad that he was willing to let me join his lab and co-advise me with Manuela. Oliver has provided me with extensive insights for robot manipulation and access to the robots. Without Manuela's or Oliver's input and support, my thesis would not have been possible.

I would also like to thank Professor George Kantor for agreeing to be on my committee on such short notice.

Next, I would like to thank my friends and family for supporting me through the Masters program. In particular, I would like to thank my labmates in both the CORAL Lab and IAM Lab for the lively discussions about research that helped me formulate ideas for my research. In particular, I would like to thank Mohit Sharma for creating the robot interface with me and coaching me with ideas, Samuel Clarke for helping me set up the microphones on the robot, and Jacky Liang for helping me register the cameras to the robot. I would also like to thank Travers Rhodes for always being positive and bringing a smile to the lab, Tanya Marwah for helping Mohit and I improve the writing style in our papers, and Anahita Mohseni Kabir for agreeing to be on my committee and giving me advice.

Finally, I would like to thank Sony Corporation for funding this research.

# Contents

# List of Figures

# List of Tables

# 1. Introduction

Robots are becoming increasingly prevalent in our everyday lives, ranging from autonomous cars to smart home assistants. However, we are still a few ways off from having fully autonomous robots in our homes that can assist us with all of our daily chores because of the difficulties of enabling robots to robustly handle different environments. Instead, companies like iRobot have created robots like the Roomba to do simple cleaning tasks well. In turn, they were able to spawn a new market for robotic vacuum cleaners.

Cooking is another daily household task that could be automated using robots. Cooking takes a significant amount of time that busy families or elderly people may be unable to do, which leads them to eat less healthy alternatives such as frozen food or fast food. Robot chefs could instead cook 24/7 to prepare healthy breakfast, lunch, and dinners for families by following recipes from top chefs.

Figure 1.1: Spyce Rotating Wok Robot

Figure 1.2: Creator Hamburger Robot

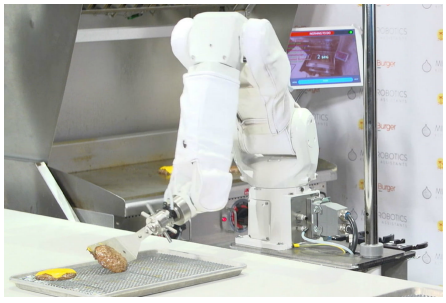


Figure 1.3: Flippy the Robot

Figure 1.4: Moley Robotics

Companies such as Spyce and Creator, shown in Figures 1.1 and 1.2 above, have created highly constrained conveyor belt styled robots that are only able to cook a select number of dishes, namely wok-based foods for Spyce and just hamburgers for Creator. The creators of Flippy, shown in Figure 1.3, instead focused on the simple task of flipping burgers on the grill using some computer vision for feedback; however, it still requires humans to accurately

place the hamburger patties on the grill. Finally, Moley Robotics, shown in Figure 1.4 above, have tried to solve robotic cooking by using motion capture to record and mimic the exact movements of a top chef in order to recreate a dish; however, no ingredients are ever the same. They can differ in size, shape, color, etc., which makes it incredibly difficult for motion captured trajectories to repeatedly manipulate ingredients successfully.

We will detail our work in applying multimodal techniques to the cooking domain in order to make robots more robust to differences in the material properties of food items. More specifically, we will be focusing on the task of slicing food items. Cutting and preparing ingredients is a fundamental part of cooking, but it is also a monotonous and time-consuming task for chefs. Food processing factories have specialized machines for processing specific ingredients in massive quantities, but those are not economically or practically feasible for small restaurants or homes.

Our solution is to endow off-the-shelf general-purpose robotic arms with the ability to cut and prepare a variety of ingredients autonomously. However, this is a very challenging problem because food items differ greatly in both visual properties (e.g, shape, color, and texture) and mechanical properties (e.g., hardness, density, and friction) that cannot be readily inferred from just their exteriors [26]. In addition, cutting deals with deformable and soft objects, which are difficult to simulate. Finally, different events can occur while the robot is executing a standard sequence of cutting skills that it must monitor and react appropriately to for proper execution.

To address all of the above issues, we combine cameras, the robot's force-torque sensors, and contact microphones, which relay tactile feedback, to provide continuous feedback to the bi-manual robot setup so that it can successfully pick up and slice a desired object. First, we will detail our entire system pipeline to control the robot. Then, we will illustrate how we taught the robot the cutting skill, which we can leverage in the future to teach the robot a more diverse set of cooking skills such as stirring, grating, sauteing, etc. Finally, we will present the results of using the different modalities to train neural networks that provide the robot with feedback while it is executing skills and enable it to adapt its slicing action to each class of materials.

# 2. Related Work

Previous work on robotic cutting has focused on model-predictive control (MPC) to generalize between different types of food items. Lenz et al. [17] proposed learning a deep network to model the cutting interaction's dynamics for their DeepMPC approach. This network continually estimated the latent material properties throughout the cutting process using interactive perception and a recurrent network structure. Their setup is shown in Figure 2.1. Mitsioni et al. [23] extended this approach by incorporating measurements from a force-torque sensor into the dynamics model. They again used a recurrent neural network to model the dynamics of the cutting task. Their setup is shown in Figure 2.2.

Figure 2.1: DeepMPC

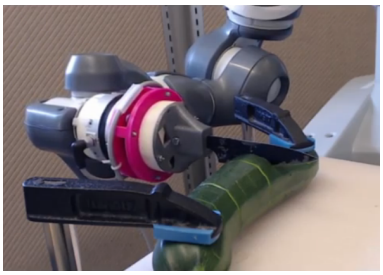


Figure 2.2: Data-Driven MPC

In contrast, rather than attempting to model the complex dynamics of cutting interactions in detail, we instead directly learn a dynamic movement primitive (DMP) [27] policy for the cutting skill. The robot learns the DMP from human demonstrations, which allows the human to implicitly transfer prior knowledge to the robot in an intuitive manner (e.g., using long smooth lateral motions for cutting). The DMP adapts to different food items based on two input parameters which we estimate based on the initial interactions with the food item. Learning the policy is generally easier than learning the low-level dynamics for the cutting task. Unlike the MPC approaches, we do not continuously update our parameters to material variations during individual cuts. However, we were still able to cut 23 different food items using our DMP-based approach.

A large part of robotics research has focused on using vision as the primary sensory modality for performing manipulation tasks [18], [19], [24]. However, for complex and contact rich tasks such as food cutting, using visual sensory data alone is insufficient. As a robot is cutting a food item, there can be major occlusions in the visual scene. In addition, in many cases, key events may occur that are not reflected in the visual sensory data. For instance, the knife may have cut the food item completely, but the new slice may not fall down, or it may be stuck to the knife. Thus, using solely visual sensory data to execute and monitor cutting events is extremely challenging.

Many previous works have used haptic feedback to accomplish a range of robotics tasks. For instance, Drimus et al. [5] use haptic feedback to classify both deformable and rigid objects while Chu et al. [3] use haptic feedback to classify haptic adjectives. Furthermore, haptic feedback has also been used to infer the object properties of deformable objects as

well as properties such as hardness [7], [29], [14]. Also, Gemici and Saxena [9] use tactile feedback along with other robot data such as poses to determine the physical properties of different food items. However, they use specific tools and actions to infer carefully designed features which are then used to predict properties such as the elasticity and hardness of the food items by training a network using supervised learning. In contrast, instead of determining the exact material properties of food items, we detect the food material type, which is then mapped to parameters that correspond to the adapted slicing skill for that food item.

The use of vibration signals for robotic tasks has been explored in the past. Clarke et al. [4] estimate the weight of materials that was scooped by a Sawyer arm using a contact microphone and a shaking motion. They test their algorithm on granular materials such as coffee beans, rice etc.. Additionally, Liang et al. [20] use a microphone to estimate the height of the liquid in a container that a robot poured into. Kroemer et al. [16] use a contact microphone at the end of a robot's end effector to learn a material's properties by stroking and visually inspecting it. Previous works have also focused on using tactile feedback for object classification [2], for both rigid and deformable objects. Similar to the above works, we use the rich auditory signals generated during robot cutting to robustly cut food items. However, in contrast to the above approaches, we combine the continuous auditory data to detect and infer the occurrence of both discrete and continuous events during deformable object manipulations. We use these inferred events to chain the different skills that are required to cut different food items.

# 3. Experimental Setup

We will first discuss our experimental setup in order to ground our methods.

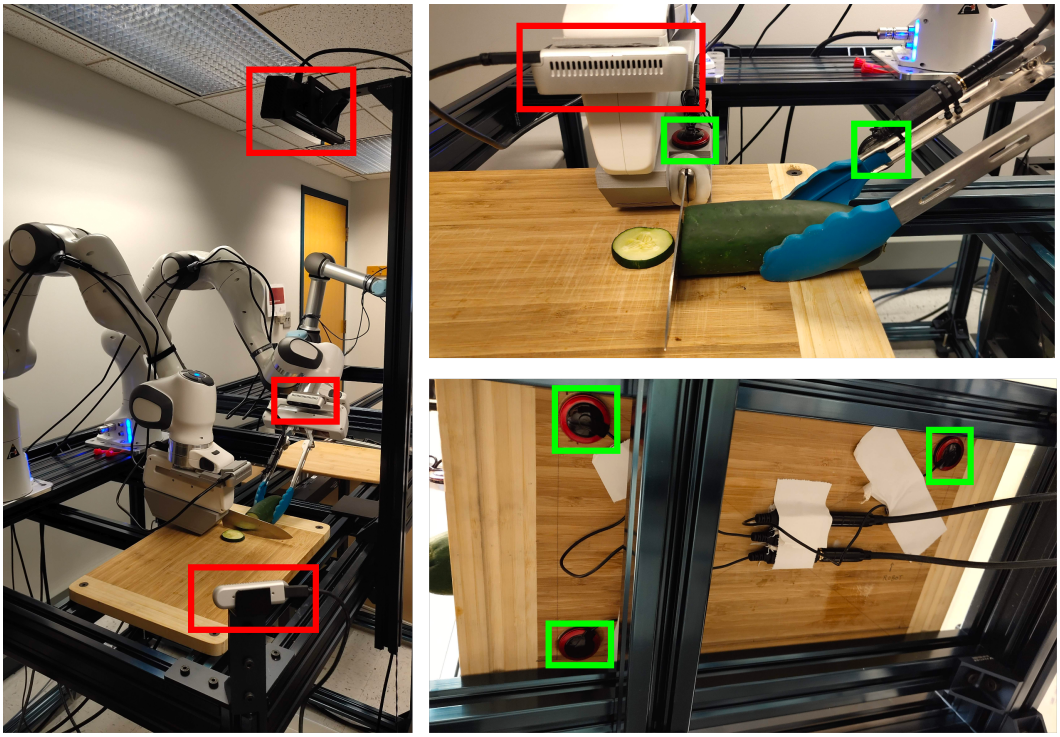


Figure 3.1: Experimental setup with cameras shown in red boxes and contact microphones shown in green boxes

Our experimental setup is shown above in Figure 3.1. This setup is used to both collect the training data of different cutting events as well as evaluate the learned networks. Two Franka Panda Research Arms are mounted next to each other on a custom Vention metal frame. The left Franka Arm is grasping a 3D printed knife attachment, which we call the Knife Arm, while the right Franka Arm has 9 inch tongs bolted onto its finger tips, which we call the Tong Arm. We have two cutting boards, one of which is bolted to the metal frame.

Our setup includes 4 cameras mounted to various locations around the setup, as shown in red boxes in Fig. 3.1. One Microsoft Kinect V2 is located overhead while three Intel RealSense D435 cameras are mounted to the side of the frame, on the knife arm looking at the knife, and on the right arm looking at the tongs. The images from each camera are shown below in Fig. 3.2. We only use the overhead Microsoft Kinect v2 in this work.

There are five contact microphones attached to various objects in the setup shown in green boxes in Fig. 3.1. These contact microphones can interpret vibrations that will give

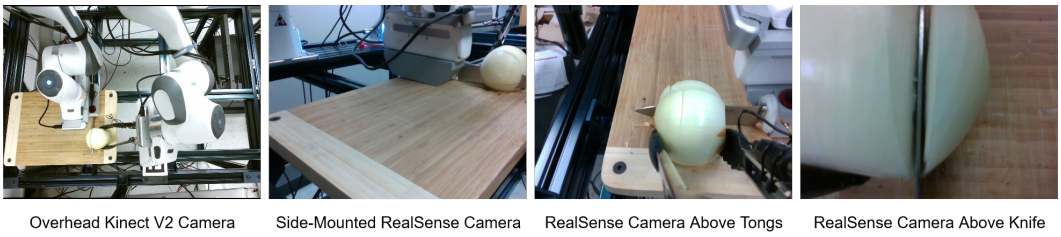



Figure 3.2: Images from each of the cameras on our setup

the robot tactile sensations in the form of audio signals. This approach was inspired by the fast afferents located in human skin, which detect vibrations during manipulation tasks and during tool usage [13]. One is located on the knife attachment and is able to sense vibrations from the blade of the knife. Another is located on the right tong of the right arm, and it is able to sense vibrations from both tongs. Finally, three microphones are located underneath the cutting board. The vibration signals from the microphones are captured using a Behringer UMC404HD audio interface system. The Behringer UMC404HD can only capture audio from 4 microphones at once, so we only utilize two of the microphones in opposing corners underneath the cutting board along with the ones on the knife attachment and tongs.

We synchronize the robot state, cameras, and audio using ROS. We capture rosbags of cutting on a variety of vegetables, which we process offline in order to train neural networks that can run online.

# 4. Robot Interface

Next, we will cover our robot interface, which we built from the ground up to control the robot at its maximum control frequency (1 kHz) as well as teach it new skills.

## 4.1 System Overview

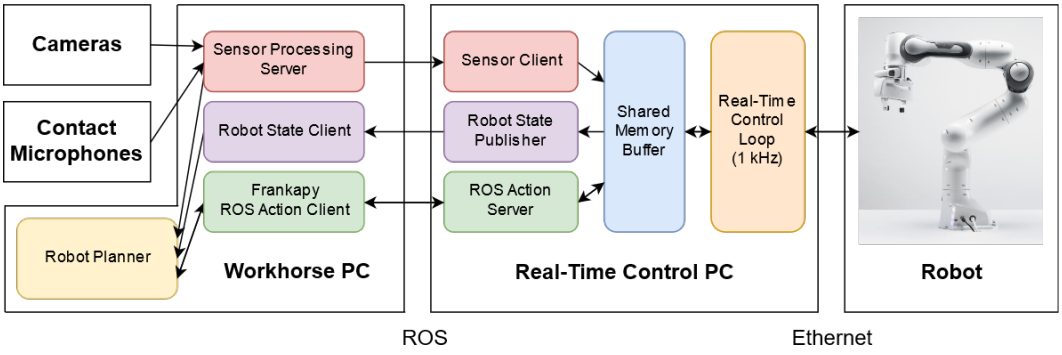



Figure 4.1: Robot Interface System Diagram

Our system shown in Figure 4.1 consists of three main parts: the Franka robot, a Control PC that is running a real-time Ubuntu Kernel, and a Workhorse PC with a powerful CPU and GPU that handles the sensor processing as well as the high-level planning. We will walk through the entire system starting from the Workhorse PC on the left and slowly moving to the robot on the right.

On the Workhorse PC, we have the sensor processing server, robot state client, Frankapy ROS action client and the high-level robot planner. In addition, the Workhorse PC is connected to the various cameras and contact microphones scattered around the robot setup. First of all, the high-level planner is typically a python script that is a finite state machine, and it takes processed sensor information as well as the current robot state as inputs to command the robot. Next, the sensor processing server typically consists of a neural network that outputs a classification result that it then publishes. This is sent to both the robot and the high-level planner and is what we mainly use as feedback for robust cutting. We will go into further depth on the sensor processing networks we implemented in later chapters.

The robot state client on the Workhorse PC is standard and reads information about the robot's current joint positions, end-effector pose, experienced torques, current skill executing, etc. Finally, the Frankapy ROS Action Client is a python wrapper API that contains all of the skills that we can have the robot execute. Some example functions that the Frankapy wrapper contains are 'go_to_pose', 'go_to_joints', 'open_gripper', and 'execute_dmp'. These functions wrap around the standard ROS Action server goals, which contain the parameters for each skill. They are sent over the ROS networking protocol to the ROS Action server on the Real-Time Control PC that receives and parses the parameters.

The ROS action server on the Real-Time Control PC has access to a shared memory buffer that is shared with the 1 kHz Real-Time Control Loop. Once the ROS action server parses the parameters, it stores them into the shared memory buffer. The Real-Time Control Loop continually queries the shared memory to see if a new skill's parameters were input, and when it sees that a new skill is available, it starts executing the skill. The skill parameters contain 5 main fields: the type of the skill, trajectory generator, feedback controller, termination handler, and the sensor topics to subscribe to. Examples of skill types are: using impedance control to go to a robot pose, using the robot's internal pose controller to go to a robot pose, using the robot's internal joint controller to go to joint positions, etc.

Next, we implemented several different trajectory generators for each skill type. For example, we implemented minimum jerk joint and robot pose trajectory generators that reduce the wear and tear on the robot. In addition, we implemented a special Dynamic Movement Primitives (DMP) trajectory generator that we used to teach the robot the slicing motion. We will go into further detail about learning the slicing DMP in the next section. For feedback controllers, we wrote our own impedance feedback controller that uses the next desired position for the robot and the current robot's location to determine the forces needed to command the robot to move to the next position. Otherwise, most of the other feedback controllers simply set the internal robot controller's cartesian and joint impedances.

Finally, we have the termination handlers and the sensor topic parameters. The termination handlers range from time-based termination handlers that stop the skill after the designated amount of time has elapsed to termination handlers that tell the skill to stop when the robot is within a threshold of the final desired goal position of the skill or has made contact with an object. The sensor topic parameters tell the sensor ROS client which topics to subscribe to and store message data in the shared memory for the real-time control loop to query.

The last and arguably most important portion of the entire system is the real-time control loop, which communicates directly with the robot at a 1 kHz frequency. This loop handles commanding the robot over the libfranka C++ API as well as storing the robot state into logs and shared memory that can be accessed and published by the Robot State Publisher. A lot of care was taken to ensure that the real-time control loop can indeed run at 1 kHz by using multiple mutexes and shared memory partitions.

## 4.2 Slicing DMP

One of the key components of our system is the robot's slicing action. We used a dynamic movement primitive (DMP) to learn and reproduce a slicing action at any starting location.

### 4.2.1 DMP Formulation

Dynamic movement primitives [27] are a general framework for smooth trajectory generation for complex actions. They consist of linear dynamical systems and must be learned for each skill component, aka axis of movement in our case. We utilized a particular form of the DMP formulation from [15] which we illustrate below:

$$\ddot{y} = \alpha_z(\beta_z\tau^{-2}(y_0 - y) - \tau^{-1}\dot{y}) + \tau^{-2}\sum_{j=1}^{M}\phi_j f(x; \boldsymbol{w}_j)$$

where $y$ is the state, $y_0$ is the initial state, $\alpha_z$ and $\beta_z$ are constants that define the spring and damper coefficients, $\tau$ is a time coefficient, $x$ is the state of the canonical system, and $f$ is a forcing function. The canonical state $x$ acts as a timer for synchonizing multiple linear systems, and it starts at $x = 1$ and decays according to $\dot{x} = -\tau x$. $\phi_j$ are object features that allow the skill to adapt to different scenarios. $\boldsymbol{w}_j$ is a vector of weights $\in \mathbb{R}^K$ of the forcing function $f(x; \boldsymbol{w}_j)$, which shapes the DMP's trajectory. The forcing function $f$ is of the form:

$$f(x; \boldsymbol{w}_j) = \alpha_z\beta_z(\frac{\sum_{k=1}^{K}\psi_k(x)w_{jk}x}{\sum_{k=1}^{K}\psi_k(x)} + w_{i0}\psi_0(x))$$

where $w_{jk}$ is the $k$th element of the vector $\boldsymbol{w}_j$ and $\psi_k(x)\ \forall k \in 1, ..., K$ are Gaussian basis functions and $\psi_0$ is a basis function that follows a minimum jerk trajectory from 0 to 1. This DMP formulation in particular allows us to easily start the slicing motion from any arbitrary position. In addition, it allows us to modify the generated trajectory using $\phi_j$ parameters that control the amplitude and height of the slicing motion. This allows us to adapt the DMP slicing trajectory to the material.

### 4.2.2 Learning DMP Weights

We taught the robot the slicing motion using ten kinesthetic demonstrations. We collected these ten demonstrations by saving the robot's end-effector poses and joint positions at a 1 kHz frequency while a human moved the robot arm back and forth in a slicing motion as shown in Figure 4.2.

After we had collected the demonstrations, we temporally aligned them based on when the robot arm first started being moved. Then we learned the weights of the DMP trajectory in each axis using kernel ridge regression in order to create the smooth DMP trajectory shown in Figure 4.3.

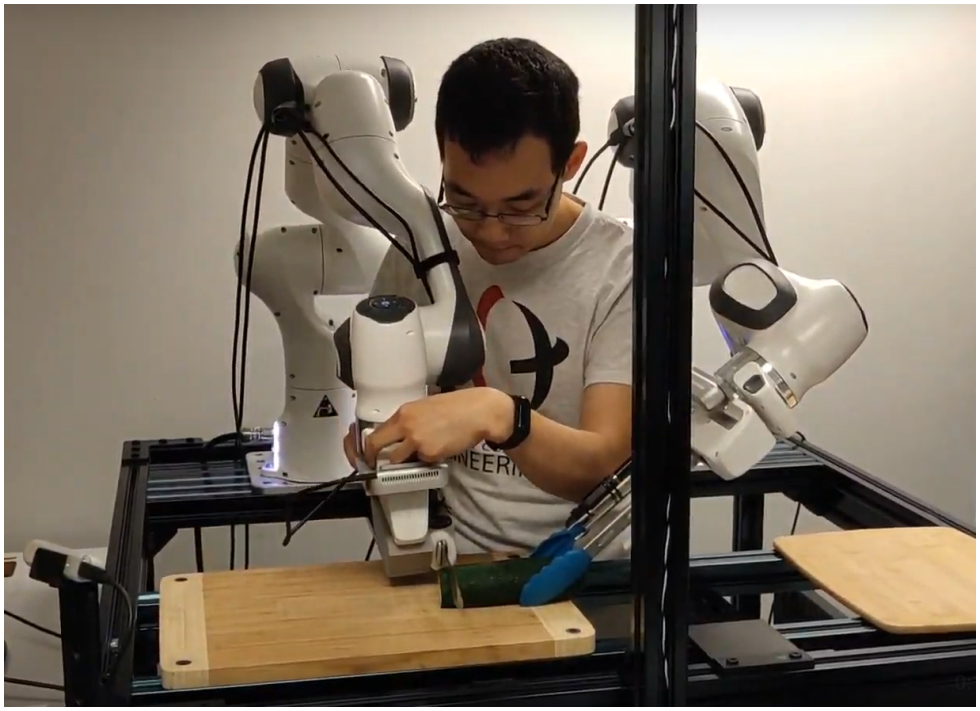


Figure 4.2: Teaching the robot through Kinesthetic Demonstrations

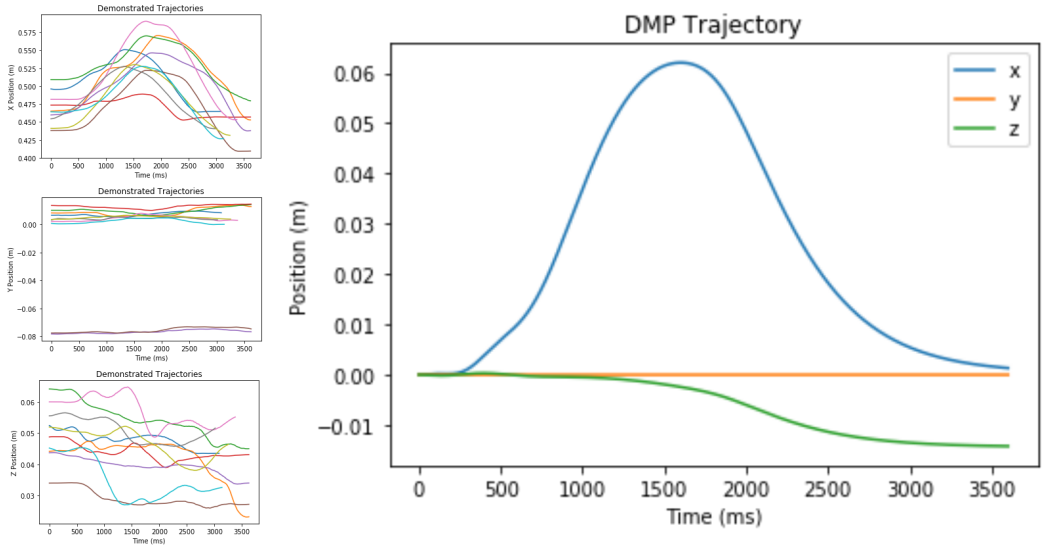



Figure 4.3: 10 Demonstrated trajectories we collected on the left and learned DMP trajectory on the right

# 5. System Overview

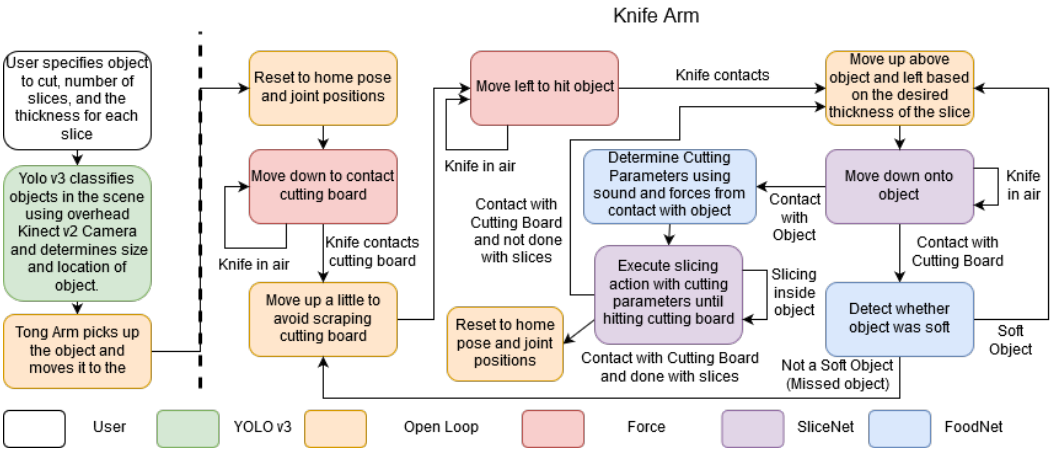



Figure 5.1: Overall Robot Cutting Flow Diagram

The overall flow of our system is shown in Figure 5.1. The main input into our system is the desired object to cut, the number of slices to create, and the thickness for each slice. In addition, we require that the desired object is placed somewhere that is within the Tong Arm's workspace in order for the system to function properly. There are 2 main phases separated by a vertical dashed line: on the left is the system for picking and placing the object using the Tong Arm, while on the right is the system for cutting the object with the Knife Arm. We will go into more depth of each phase in the following sections.

## 5.1 Object Classification using YOLOv3

In order to determine the location of a desired object, we first need to be able to recognize the object. There are numerous object recognition neural networks that perform extremely well such as Mask-RCNN [11]. However, very few object recognition systems are able to perform well in real-time, which we will consider as above 30 fps. We decided to use YOLO v3 [25] for our object recognition system because it is both fairly accurate and it runs in real-time on a single Nvidia GTX 1080Ti GPU.

We used an open sourced labeling tool [30] and labeled 43 images in the YOLO format with 14 classes of objects including their resulting slices using images from the overhead Kinect. We started training with the pretrained YOLOv3 network and fine-tuned it on our dataset for 300 epochs. An output from YOLO v3 is shown below in Figure 5.2.

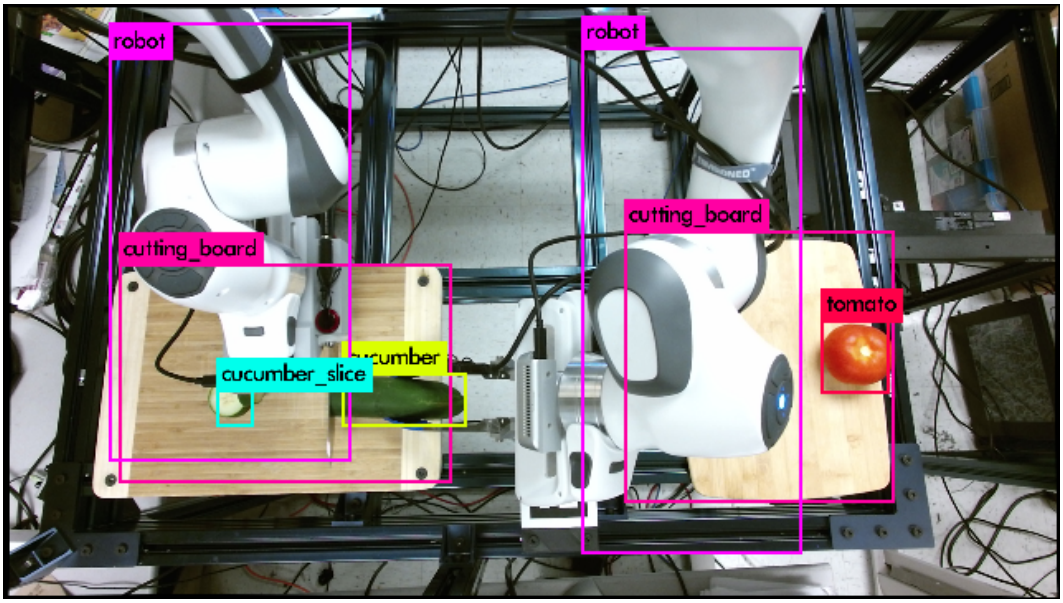



Figure 5.2: YOLO v3 results from overhead Kinect camera

## 5.2 Picking and Placing the Object

Now that we are able to recognize the objects in the scene with the help of YOLO, we need to determine the location of the object with respect to the robot in order for the robot to successfully pick it up. To do so, we attached a checkerboard to a piece of laser-cut wood that was positioned a fixed distance away from the center of the robot as shown in Figure 5.3. To register the depth camera to the robot coordinate frame, we took a few images of the checkerboard using the IR depth camera of the Kinect, found the center of the checkerboard in the IR Depth image, and then used the known distance of the checkerboard from the robot center to calculate the transformation matrix.

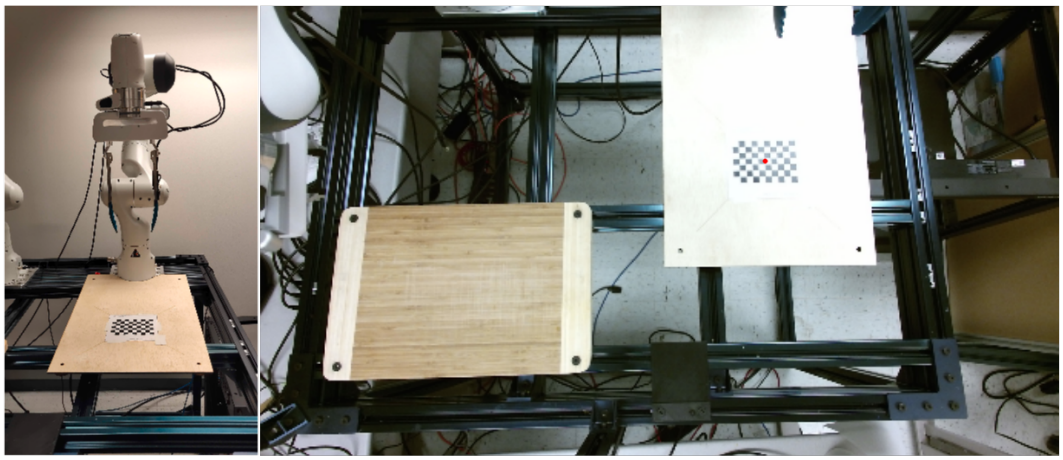



Figure 5.3: Checkerboard used for Kinect Robot Calibration with the red dot at the center

We also needed to register the RGB camera with the Depth camera of the Kinect by taking multiple pictures of a checkerboard at different positions. We used the inbuilt Kinect2_calibration package inside the iai_kinect2 [31] ROS package to determine the intrinsic camera parameters for us. Once the camera registration was completed, each pixel in the rectified RGB Image had a one-to-one correspondence with a pixel in the rectified Depth Image.

Now, with all the pieces in place, the robot was able to determine the location of the object by first using the center of the bounding box detected by YOLO v3 in the rectified RGB image. Then with the one-to-one pixel correspondence between the rectified RGB and depth images from the Kinect, the depth of the object from the Kinect was able to be queried. Afterwards, we simply used the transformation matrix to transform the center of the object in the depth image to a location in the robot's coordinate frame.

Once armed with the location of the object, we command the Tong Arm to first reset to its initial pose. Next, we instruct the arm to go to a pose above the center of the desired object to avoid colliding with other objects on the cutting board. Afterwards, the robot simply goes down to the center of the object, closes its gripper and lifts the object up. It then moves the object to a predetermined location on the cutting board, opens its gripper, and regrasps the object in a more favorable position so that the Knife arm will have a lower chance of colliding with the Tong Arm while slicing. The entire pick and placing process is

illustrated below in Figure 5.4.

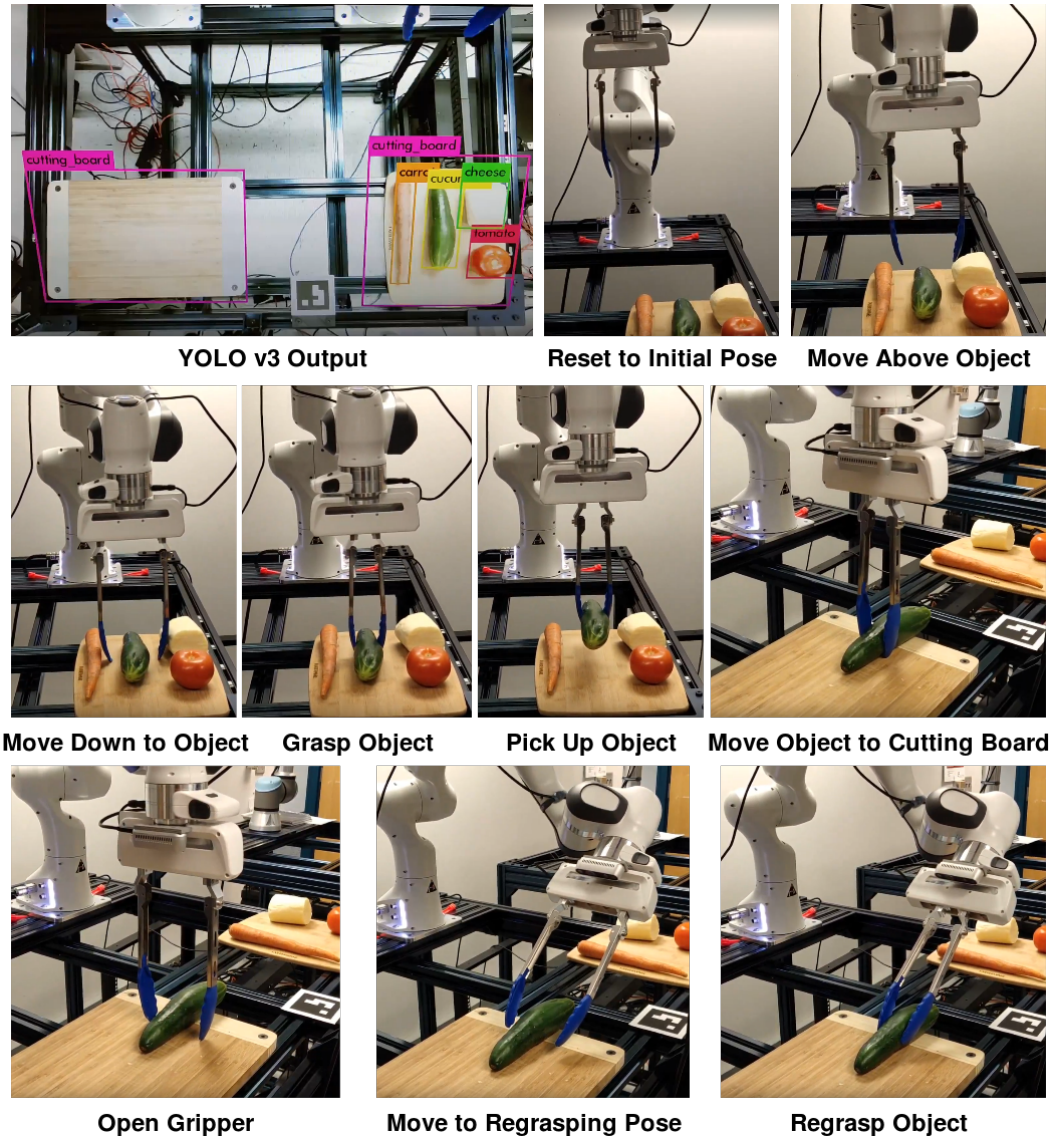



Figure 5.4: Time-lapse of Picking and Placing a Cucumber

## 5.3 Knife Arm Slicing

At the heart of the robotic cutting problem is the problem of controlling the Knife Arm to execute the slicing skill appropriately and cut the designated number of slices, each according to a desired thickness. This is a challenging problem because there are numerous events that can occur as the Knife Arm is executing that we will need to constantly monitor and provide as feedback to the robot. In addition, as every material has different properties, we need to be able to adapt our cutting action to the material. We will first discuss our overall robot slicing pipeline and then move onto the separate issues.

### 5.3.1 Slicing Process

Figure 5.5 below illustrates the main events of the Knife Arm as it slices an object.

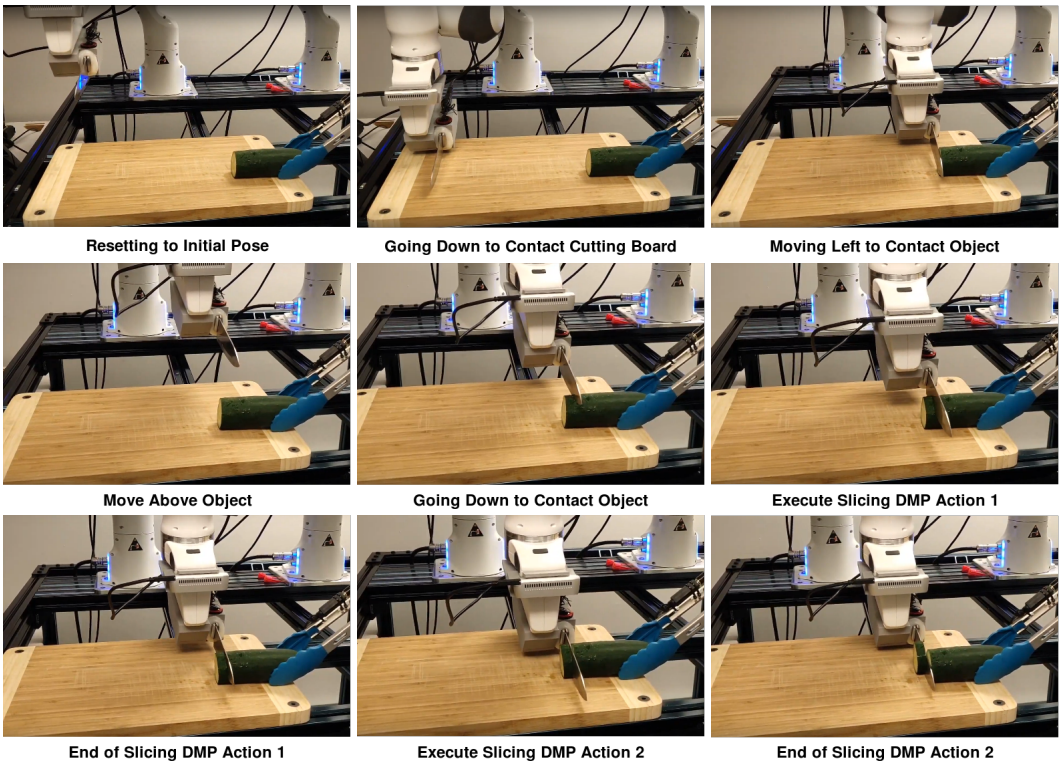



Figure 5.5: Time-lapse of Slicing a Cucumber

First of all, the Knife Arm resets to its initial pose. Then, it goes down until it contacts the cutting board in order to localize the cutting board. Afterwards, it lifts up a little bit and moves left until it hits the object in order to localize the object. This is because YOLO v3 cannot localize the edges of the object well as the bounding boxes it outputs are not perfect. Next, the robot moves above the object, moves left the desired thickness of the first cut, and goes down until it contacts the object. Subsequently, the Knife Arm determines the optimal cutting parameters for the object and executes slicing DMP actions until the

robot has cut through the object completely. Then it moves onto the next slice until the desired number of slices has been created.

The entire finite state machine for the Knife Arm is illustrated in Figure 5.6 below.

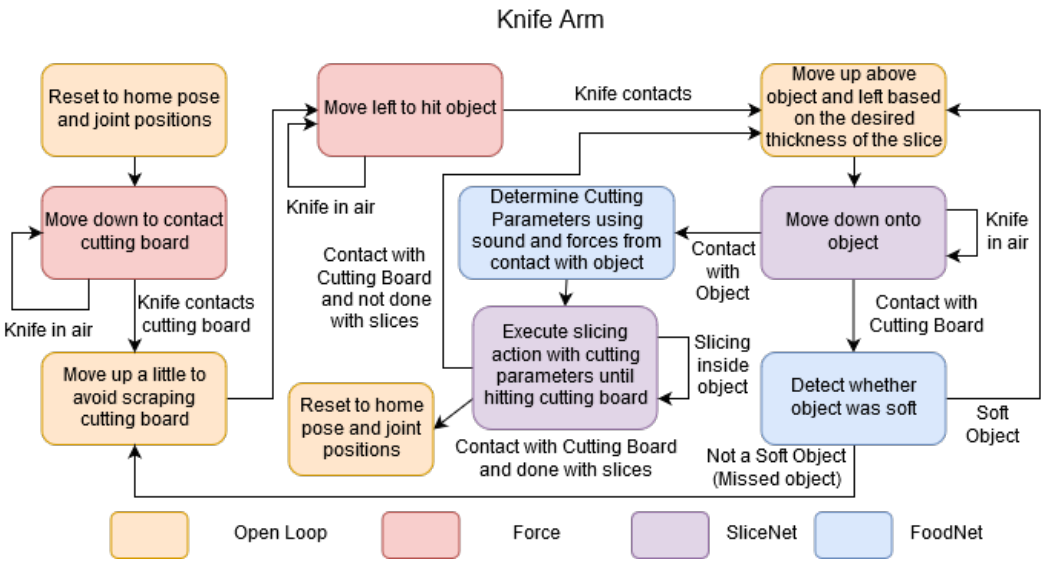



Figure 5.6: Knife Arm Cutting Flow Diagram

As shown in Figure 5.6 above, the Knife Arm's Finite State Machine relies on two networks in order to cut food items robustly. **SliceNet** continuously monitors the Knife Arm's status and detects events. Meanwhile, **FoodNet** predicts the type of material of the object we are slicing and outputs the appropriate parameters for the slicing action.

### 5.3.2 Sensory Events During Cutting Tasks

SliceNet came out of necessity because we have experienced numerous abnormal events while executing cutting actions open loop. For example, the robot may hit the cutting board immediately when trying to go down on the object because either the object was too soft, like tofu, or the knife slipped on the object because the desired slice thickness was too thin. We want to be able to detect these abnormalities quickly so that we can stop the robot from unnecessarily executing the slicing actions on the cutting board which may make indentations in the cutting board or just waste time.

Another event that can occur is the knife slipping on the surface of the object when it is executing the cutting action. This event sometimes occurs when the skin of the object is sloped and tough like a watermelon's. If we do not catch this slipping / scraping quickly, the resulting slice will have an inaccurate thickness. We can recover from this error by reorienting the knife to prevent slipping or going down harder on the object in order to embed the knife within the object's surface.

Next, we want to also be able to monitor correct robot states so that if any detected events deviate from normality, the robot can better recover from it in the future. For instance, if the Knife Arm can correctly classify when it is slicing an object, it will also be able to notice when it is stuck because it might be attempting to execute the slicing action,

but no movement and only high forces result. In addition, the robot needs to know when it has correctly made contact with an object instead of the cutting board so that it doesn't falsely raise errors. Finally, when the robot has finished cutting through the object, it should detect when it has started scraping the cutting board in order to terminate properly and quickly.

The events we talked about are illustrated in Figure 5.7 below.

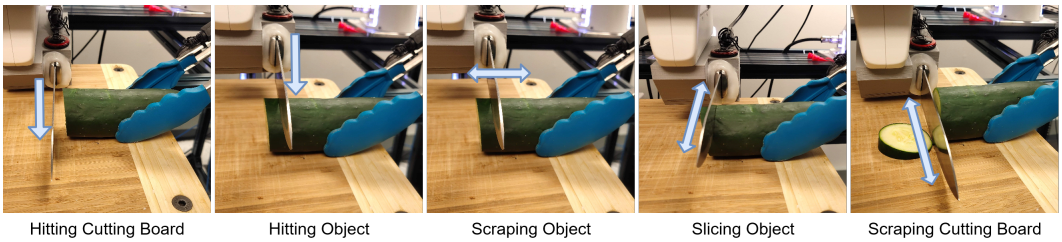



Figure 5.7: Different events that may occur when cutting

### 5.3.3 Adapting to Materials

As different objects have different material properties, there is no way for one slicing motion to work for all of them. So, our goal with the FoodNet was to allow the robot to classify the toughness of the food material being cut and consequently adapt its slicing technique accordingly. We are able to do this by modifying the $\phi_j$ parameters of the slicing DMP skill. Since there is a DMP for each axis, we can modify both the amplitude of the x-axis forward and backward action as well as the overall downward height displacement in the z-axis.

The default parameters that the original slicing DMP learned was approximately a 6.3cm x-axis amplitude and a -1.5cm z-axis displacement, which is shown in Figure 4.3. We collected the amplitude and height parameters for each class of object manually from human trial and error, and when we predict the material type of an object, we substitute the parameters of the predicted material type into the slicing DMP action in order to cut the objects more efficiently. We will go into more details in the later chapters.

# 6. Method

## 6.1 Overview

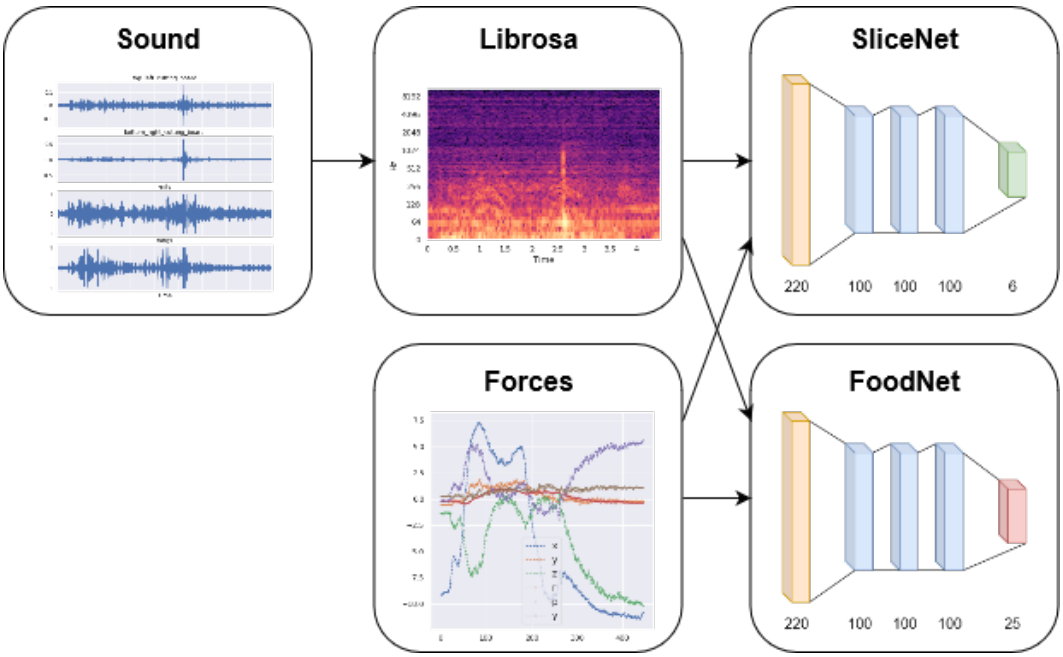



Figure 6.1: SliceNet and FoodNet System Diagram

The high-level **SliceNet** and **FoodNet** system architectures are depicted in Figure 6.1 above. To reiterate, **SliceNet** monitors the status of the robot in real-time using the features and acts as a feedback signal for the robot's slicing finite state machine. On the other hand, **FoodNet** uses the features from one downward motion onto the object to classify the hardness of the food item and output the appropriate parameters that the robot can then use to cut the food object quickly and robustly.

We extract features from four contact microphones using the Librosa python audio processing library [22], which we then ensemble with the robot's force data. These combined features are then sent into the separate **SliceNet** and **FoodNet** neural networks. We do not use visual features for either network. First of all, visual data will be noisy for **SliceNet** because it is difficult for vision to determine when the robot has completely cut through an object unless the cut slice falls down. Secondly, if there are multiple slices, vision may incorrectly report that the slice was already cut if several slices are stuck together. Next, visual features are not very useful for **FoodNet** because YOLO v3 can only see the surface of objects and will be unable to determine the interior properties of the object without first cutting it open. For both networks, vibrations and forces are enough information to robustly

evaluate the robot's state and the current object's inherent material properties.

Now, we will go in depth with each component of the systems.

## 6.2 Audio Processing

The first critical component of our system is the audio processing. We can capture the sound from the 4 microphones by reading in the raw sound data using the python-sounddevice package [8], which directly interfaces with our Behringer UMC404HD using the PortAudio and ALSA framework for Linux. The sounddevice package provides the data in a easy to use NumPy 2d array format where one dimension is the number of channels (4 in our case) and the other dimension is a varying sound byte length. We sample from the microphones with a 44.1kHz frequency, and we use an audio buffer to process 0.1 seconds of sound at a time, which gives us an array of size (4410, 4). I wrote a simple sounddevice_ros package that publishes the sound bytes continuously in ROS, which allows the robot to do inference in real-time as well as sync and save the audio in rosbags. The waveplots from all 4 microphones are shown in Figure 6.2.

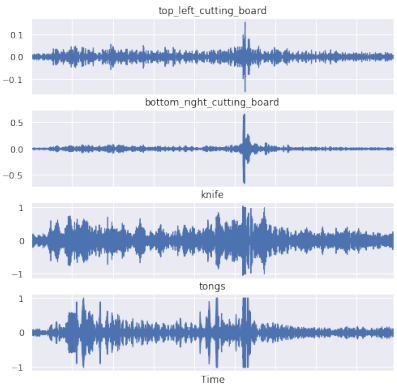



Figure 6.2: Audio waveplots from all 4 microphones

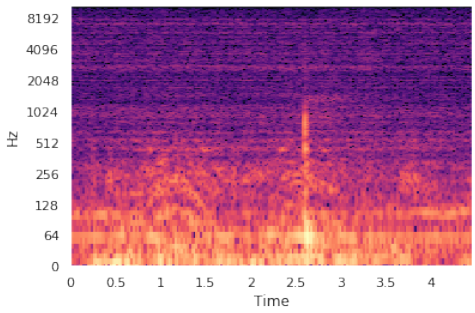



Figure 6.3: Spectrogram using Librosa

We then use Librosa to do audio processing as it is a well established library with many great audio processing functions easily accessible. In particular, we use the following audio features built in to Librosa: Mel-frequency cepstral coefficients (MFCCs), chromagrams, mel-scaled spectrograms, spectral contrast features, and tonal centroid features (tonnetz). The Mel frequency scale provides a rough model of human frequency perception [28]. An example of a mel-spectrogram is shown above in Figure 6.3. Mel-frequency cepstral coefficients are often used for speech recognition systems and are basically timbre features, but we thought that they might also be useful in our application [21]. Chromagrams project audio from the entire spectrum onto 12 bins representing the 12 distinct semitones of the musical octave because humans perceive notes exactly one octave apart as similar [6]. Spectral Contrast features have been shown to perform well when discriminating between different music types [12]. Finally, tonal centroid features can detect changes in the harmonic content of musical audio signals using chroma features. [10]

We process each audio channel separately and extract the mean of each feature over the 0.1 second sound bite. There are 40 Mel-frequency cepstral coefficients, 12 chromagram

features, 128 mel-scaled spectrogram features, 7 spectral contrast features, and 6 tonal centroid features. In total, that gives us 193 features per channel, which when multiplied by the 4 channels results in 772 audio features for every 0.1 seconds. These are the vibration features that we concatenate with the forces and input into our **SliceNet** and **FoodNet**.

## 6.3 Forces

The robot provides us with force feedback at a 1kHz frequency; however, we slow it down to a 100 Hz frequency that is communicated over ROS. Thus, in order to match up the forces with the 0.1 seconds of sound, we use a buffer of the last 10 robot forces which are with respect to the x, y, z, and roll, pitch, yaw axes. In total there are 60 force features that are concatenated with the 772 audio features. Example forces are shown in Figure 6.4 below.

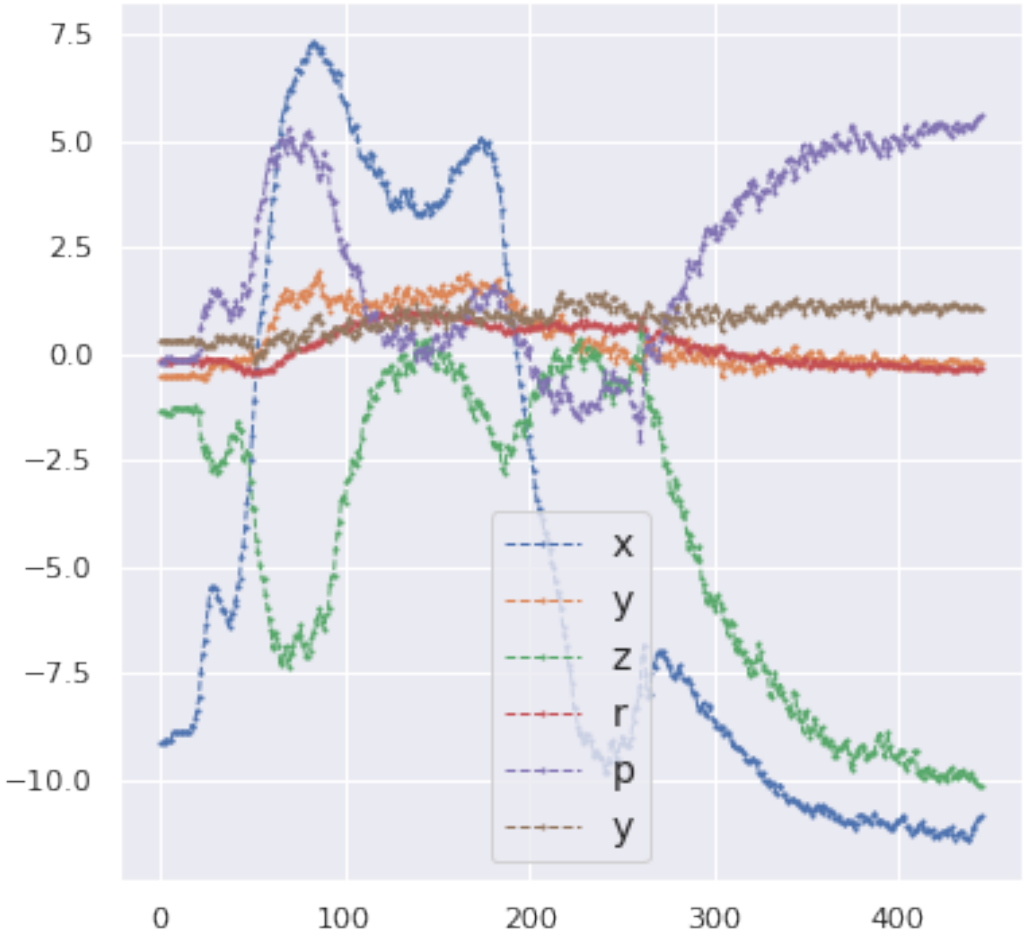



Figure 6.4: Forces in each axis

## 6.4 Data Collection

We collected a comprehensive dataset of a variety of different materials to train FoodNet. In addition, we also collected a dataset of events that may occur during slicing in order to train SliceNet. Both datasets consist of rosbags that can be processed to produce audio wav files, images, and robot state data.

### 6.4.1 SliceNet Dataset

For the SliceNet dataset, we collected separate rosbags for each of the following event classes for monitoring slicing:

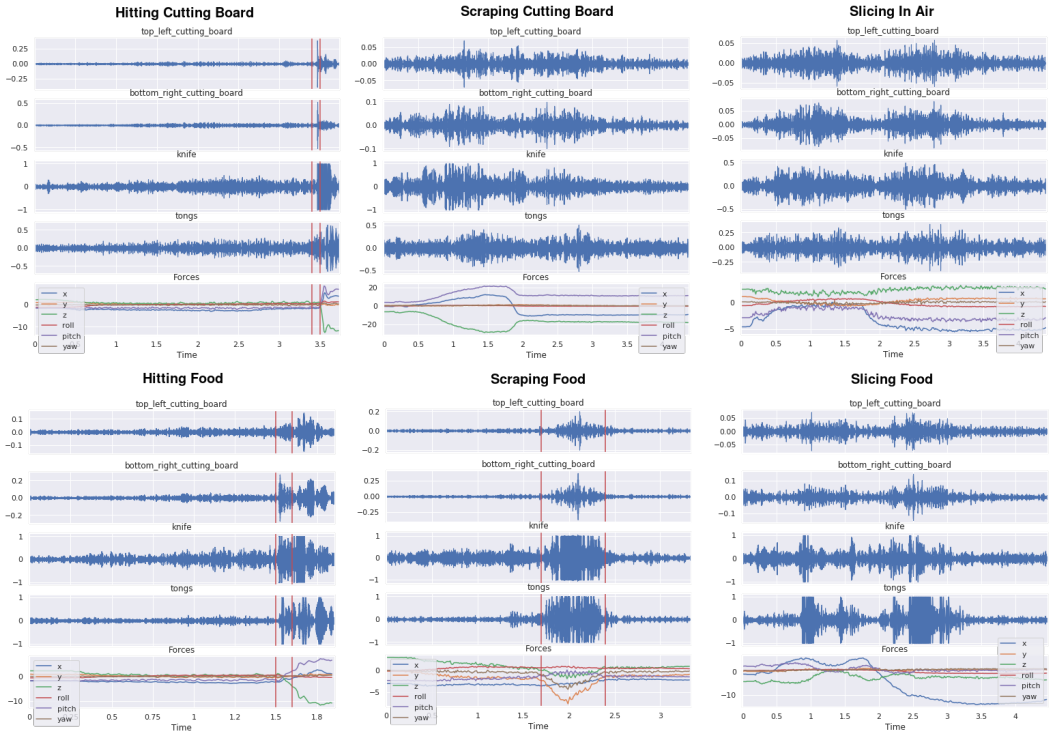



Figure 6.5: Sound signals from each of the 6 events that may occur when cutting

1. Hitting the cutting board: The knife robot hit the cutting board at random locations 60 times.

2. Scraping the cutting board: The knife robot scraped the cutting board at random locations 10 times.

3. Hitting an object: The knife robot hit each type of object between 10 and 15 times depending on the length of the object.

4. Scraping an object: The knife scraped each object twice, once from left to right and once from right to left a distance between 5cm to 10cm depending on the length of the object.

5. Slicing an object: The knife robot executed between 20 and 40 DMP slicing actions on each object depending on the thickness of the object resulting in around 10 to 15 slices cut from each object.

6. In the air (Background): The knife robot executed DMP slicing actions 10 times at random locations in the air.

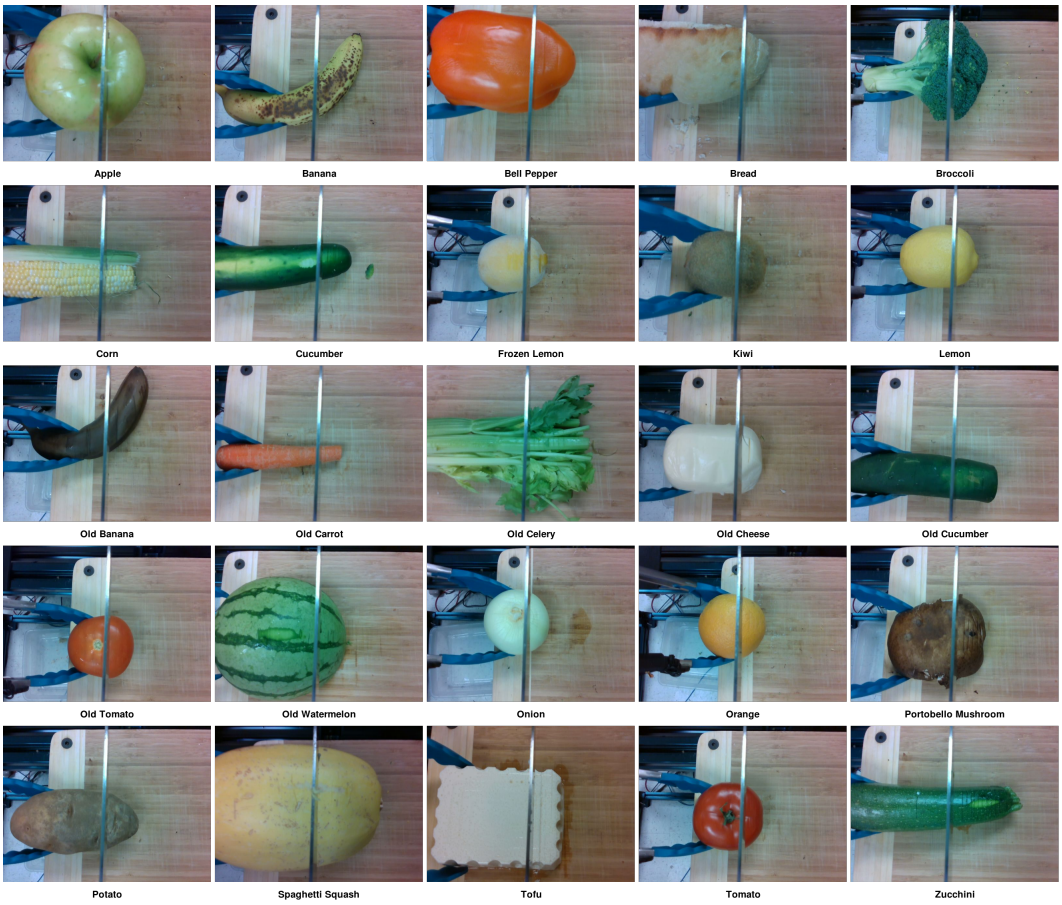



Figure 6.6: Collage of all the Food Items

An example from each class is shown in Figure 6.5 above. In order to label the data, we segmented the audio based on the skill the robot was executing. Then, we used an online Bayesian changepoint detection algorithm [1] to segment out the audio of the actual event. For example, when the robot was moving down to hit the cutting board, we knew the robot was in the air until a change was detected in both sound and forces. In skills where we knew the robot was completely in a specific state, we simply labeled the entire skill's features

the same. For example, if we saw that the robot was still above the cutting board when executing a DMP slicing skill, we would know that it was continuously in the slicing an object state.

Using these labelling methods, we constructed the SliceNet dataset. The red vertical lines in some of the events in Figure 6.5 signify the changepoints that were detected by the online Bayesian changepoint detection algorithm.

### 6.4.2 FoodNet Dataset

We reused the hitting dataset from the SliceNet Dataset, but instead, we labeled skills based on the object the Tong Arm was holding. In addition, we collected empirical data on the parameters that could cut the objects fast and robustly. These are the $\phi_j$ parameters of the DMP, which influence the amplitude and height displacement of the cutting motion. We cannot guarantee that they were the optimal values for each object, but they worked considerably better and faster than the constant DMP parameters we used for the slicing data collection. The full list of food items is shown above in Figure 6.6 and their DMP parameters are detailed in Table 6.1 below. Old food items refer to ones that were in the refrigerator for longer than a week.

| Food Item | Amplitude (cm) | Height (cm) |
|---|---|---|
| Apple | 5.5 | 2 |
| Banana | 4.2 | 2 |
| Bell Pepper | 5.5 | 2 |
| Bread | 8 | 1 |
| Broccoli | 7 | 2 |
| Corn | Not Possible | Not Possible |
| Cucumber | 6.9 | 4 |
| Frozen Lemon | 6 | 1.5 |
| Kiwi | 4.2 | 2 |
| Lemon | 6 | 2.5 |
| Old Banana | 4.2 | 1.5 |
| Old Carrot | 3.3 | 2 |
| Old Celery | 6 | 2 |
| Old Cheese | 6 | 3 |
| Old Cucumber | 6.9 | 2 |
| Old Tomato | 6 | 1.5 |
| Old Watermelon | 7 | 1.5 |
| Onion | 5 | 2 |
| Orange | 5 | 2 |
| Portobello Mushroom | 8 | 0.75 |
| Potato | 6 | 3 |
| Spaghetti Squash | Not Possible | Not Possible |
| Tofu | 0 | 5 |
| Tomato | 6.4 | 2.5 |
| Zucchini | 5 | 2.5 |

Table 6.1: Food items and their DMP parameters

## 6.5 SliceNet

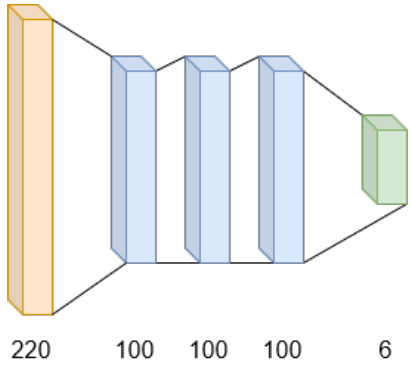



Figure 6.7: SliceNet

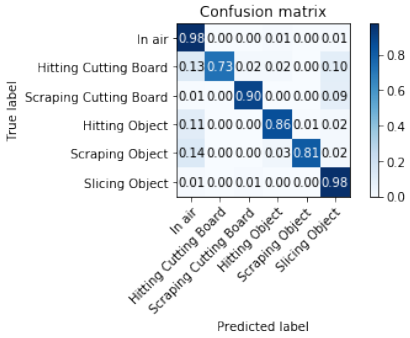



Figure 6.8: SliceNet Normalized Confusion Matrix

Given the SliceNet Dataset, a simple 3 hidden layer neural network shown in Figure 6.7 with 100 hidden units in each hidden layer and sigmoid activation was able to achieve 95.9% F1 score on a held out test set of 20% of the total data. A normalized confusion matrix is shown in Figure 6.8. The main confusions were between in the air and scraping an object, which may be because the knife only makes a slight sound when scraping some objects. In addition there are confusions between hitting the cutting board and object and in the air, which might be due to the sound being heard first in some cases before the forces and the neural network might be utilizing the forces to classify hitting events. Finally there are confusions between hitting and scraping the cutting board and the slicing object classes which may be due to occurrences where cutting forces in a hard object are similar to contact with the cutting board.

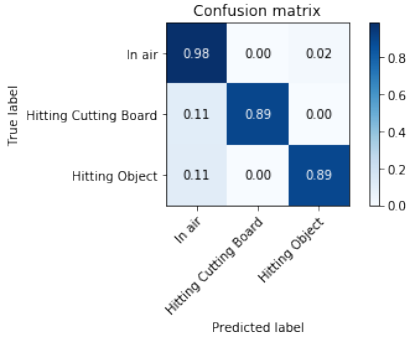



Figure 6.9: Hitting Normalized Confusion Matrix

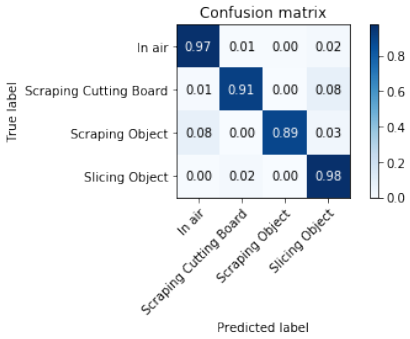



Figure 6.10: Slicing Normalized Confusion Matrix

To try and avoid misclassifications between classes in skills where certain events cannot occur, we split up the training of SliceNet into two separate networks. One classifies events for moving down to contact skill where only hitting the cutting board, in air, and hitting object classes can occur. The other classifies events for the slicing skill such as slicing object,

scraping object, scraping cutting board, and in air. We used the same network architecture above and the results are shown in Figures 6.9 and 6.10 above.

In the hitting classification network, although there are still some misclassifications between hitting cutting board or object and in air, it is fine because we still have a force threshold that stops the robot movement when it has collided with something. More importantly, there are no misclassifications between hitting the cutting board and hitting the object, which is what we care about most. In the slicing classification network, there are fewer confusions in general. There are still some confusions between scraping the cutting board and slicing an object, perhaps due to a few mislabeled data as we do not fine-grainely label when the knife has hit the cutting board.

Since SliceNet is such a small network, it can easily run in real-time on the workhorse PC and send feedback to the robot. In addition, the input to SliceNet is only 220 instead of 832 due to ablation studies we will cover in a later section.

## 6.6 FoodNet

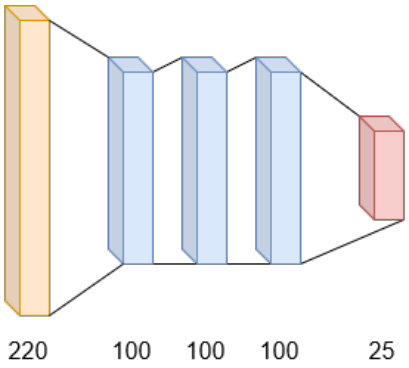



Figure 6.11: FoodNet

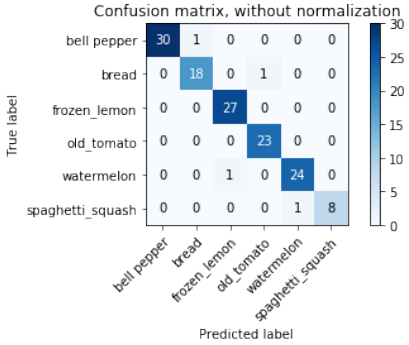



Figure 6.12: FoodNet Confusion Matrix

We again tried a simple 3 hidden layer neural network shown in Figure 6.11 with 100 hidden units in each hidden layer and sigmoid activation on the FoodNet dataset, and we were able to achieve 99.15% F1 score on a held out test set of 20% of the total data. As there were so few confusions, we only visualized the confused classes in Figure 6.12. Bread, bell pepper, and old tomato all had one confusion perhaps due to the similar textures of their skin. Frozen lemon, watermelon, and spaghetti squash also all had one confusion which may be due to their hardness.

We also tested out a regression network for FoodNet to directly predict the $\phi_j$ parameters for the $X$ and $Z$ DMPs. We utilized the same network structure as above except that we used ReLu as the activation function after each layer and used mean absolute error as the loss function. As a result, the network had a mean absolute error of 0.019cm in the $\phi_j$ for the X axis DMP and 0.013cm in $\phi_j$ of the Z axis DMP on a test set of 20% of the total data.

In Figure 6.13 on the left, we plotted the human labeled values from Table 6.1 above. In Figure 6.14 on the right, we plotted the mean absolute error for each class when we used Leave-one-out cross validation. As can be seen in the plot, Tofu had a large mean absolute

error in both the X and Z axes most likely because nothing else in the training data was similar to Tofu once it was taken out of the training dataset. Corn and Spaghetti Squash also had large inaccuracies because they were both hard objects that were unable to be cut, so we gave them X and Z $\phi_j$ values of 0. Cucumber had only a large mean absolute error in the Z axis most likely because the learned network thought that it was similar to the old cucumber, which has a 2cm difference in the Z $\phi_j$ value as compared to the regular cucumber. Finally, most everything else was fairly accurate except that sound is not a good predictor for X $\phi_j$ values in general as different objects have different thicknesses and it is more suitable to use vision to determine how far the back and forth motion should be. Sound and forces should probably only be used to classify the Z $\phi_j$ values.

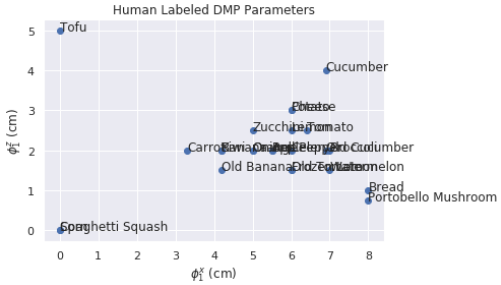



Figure 6.13: Human Labeled Parameters

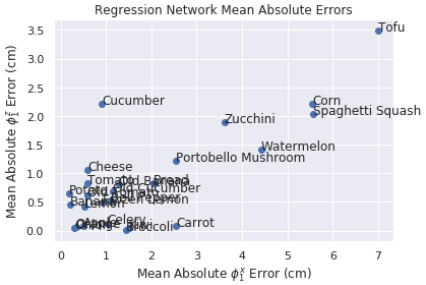



Figure 6.14: FoodNet Regression Error Plot

Since FoodNet is also a simple network, it can easily run in real-time on the workhorse PC and send feedback to the main control program. Once the class of the object is determined, the appropriate slicing DMP parameters from Table 6.1 are substituted and used to execute the slicing actions. Finally, the input of the network is again only of size 220 instead of 832 due to the ablation study in the next section.

## 6.7 Ablation Studies

As we were curious about which features and which microphones were most useful for both SliceNet and FoodNet, we did an ablation study. We tested each feature individually and each microphone individually. The results are shown in Table 6.2. We were surprised to find that just using the MFCC features was enough to achieve the majority of the accuracy for SliceNet and FoodNet. This may be because MFCC features capture the timbre of the vibration of each food item, which it is able to distinguish across while also being useful for distinguishing between the different events. In addition, we also found it interesting that for FoodNet, the contact microphone on the Tong Arm was the most useful microphone. It makes sense because the robot is always vibrating a little bit due to its motors and when grasping a vegetable, the sound is noticeably dampened, which allows it to distinguish the class of object. Finally, it makes a lot of sense that both forces and the Knife Arm microphone were great for SliceNet.

In the end, we decided to use only the MFCC features and the robot's forces for the inputs into both SliceNet and FoodNet because together, their combined accuracy they had was close enough to using all of the sound and force features. Since there are 40 MFCC features for each microphones, 4 Microphones x 40 MFCC + 60 Force Features = 220 Total

Input Features to both SliceNet and FoodNet. Calculating only the MFCC features also decreased the latency of both the networks slightly.

| Input Features | SliceNet | FoodNet |
|---|---|---|
| Combined Sound and Force Features | 95.7 | 98.44 |
| Forces | 92.73 | 17.92 |
| Sound | 93.64 | 98.3 |
| | | |
| Under Cutting Board Mic 1 | 82.03 | 71.33 |
| Under Cutting Board Mic 2 | 83.67 | 73.48 |
| Knife Arm Mic 3 | 90.7 | 61.54 |
| Tong Arm Mic 4 | 89.77 | 93.66 |
| | | |
| MFCC | 93.53 | 97.45 |
| Chroma | 74.96 | 45.74 |
| Mel | 93.5 | 96.03 |
| Spectral | 71.9 | 30.5 |
| Tonal | 53.6 | 5.61 |
| | | |
| MFCC and Forces | **95.9** | **99.15** |

Table 6.2: F1-scores with Various Features

# 7. Conclusions and Future Work

In this work, we have outlined our entire slicing robot pipeline from the ground up. Starting from our experimental setup to our new robot interface, we have built this system and engineered it to be robust by leveraging multi-modal feedback. Our use of DMPs for teaching the robot the slicing action can be reused in the future to teach the robot a variety of new cooking skills. In addition, we have shown how useful vibrations from the contact microphones can be for distinguishing the toughness of objects when they are placed in varying locations.

Some immediate future work would be using SliceNet as a reward signal for reinforcement learning and have the robot learn better slicing DMP parameters than those in Table 6.1 that were gathered from just human trial and error. This would allow the robot to earn rewards after each slicing action and adapt to the material after each DMP action instead of our current method of adapting once when hitting the object.

In addition, we would like to have the Tong Arm learn how to counteract the forces from the Knife Arm in the future to produce even cleaner cuts. Currently if the Knife Arm is cutting a really hard object such as a watermelon, it is exerting tremendous amounts of forces that the Tong Arm is unable to resist, and thus the object being cut moves a little bit while being sliced.

Finally, we have not completely solved the robot cutting problem with this simple back and forth cutting action. There are still a lot of issues with handling and slicing smaller objects such as garlic. All of the objects that we tested with were able to fit within the tongs of the Tong Arm and were of decent size. However, there was usually always a little bit of a food item that we couldn't cut because the Tong Arm was holding it. We may be able to solve the issue temporarily by rotating the Knife Arm and cutting inside the tongs, but then there is the issue of whether the tongs would squish the object once it is cut such as a tomato. Dicing onions would also be extremely difficult with just a parallel jaw gripper. Additionally, we haven't tried picking up the cut slices yet. So, there is still an extremely difficult challenge of dexterously grasping objects for cutting and manipulating cut slices reliably.

There is still a long way to go for complete autonomous robotic cooking, but I hope that this work is able to contribute a step towards that future.

# Bibliography


[1] Ryan Prescott Adams and David JC MacKay. Bayesian online changepoint detection. *arXiv preprint arXiv:0710.3742*, 2007.

[2] Tapomayukh Bhattacharjee, James M Rehg, and Charles C Kemp. Haptic classification and recognition of objects using a tactile sensing forearm. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4090–4097. IEEE, 2012.

[3] Vivian Chu, Ian McMahon, Lorenzo Riano, Craig G McDonald, Qin He, Jorge Martinez Perez-Tejada, Michael Arrigo, Trevor Darrell, and Katherine J Kuchenbecker. Robotic learning of haptic adjectives through physical interaction. *Robotics and Autonomous Systems*, 63:279–292, 2015.

[4] Samuel Clarke, Travers Rhodes, Christopher G Atkeson, and Oliver Kroemer. Learning audio feedback for estimating amount and flow of granular material. In *Conference on Robot Learning*, pages 529–550, 2018.

[5] Alin Drimus, Gert Kootstra, Arne Bilberg, and Danica Kragic. Classification of rigid and deformable objects using a novel tactile sensor. In *2011 15th International Conference on Advanced Robotics (ICAR)*, pages 427–434. IEEE, 2011.

[6] Dan Ellis. Chroma feature analysis and synthesis. *Resources of Laboratory for the Recognition and Organization of Speech and Audio-LabROSA*, 2007.

[7] Barbara Frank, Rüdiger Schmedding, Cyrill Stachniss, Matthias Teschner, and Wolfram Burgard. Learning the elasticity parameters of deformable objects with a manipulation robot. In *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1877–1883. IEEE, 2010.

[8] Matthias Geier. python-sounddevice. `https://github.com/spatialaudio/python-sounddevice`, 2015.

[9] Mevlana C Gemici and Ashutosh Saxena. Learning haptic representation for manipulating deformable food objects. In *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 638–645. IEEE, 2014.

[10] Christopher Harte, Mark Sandler, and Martin Gasser. Detecting harmonic change in musical audio. In *Proceedings of the 1st ACM workshop on Audio and music computing multimedia*, pages 21–26. ACM, 2006.

[11] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 2961–2969, 2017.

[12] Dan-Ning Jiang, Lie Lu, Hong-Jiang Zhang, Jian-Hua Tao, and Lian-Hong Cai. Music type classification by spectral contrast feature. In *Proceedings. IEEE International Conference on Multimedia and Expo*, volume 1, pages 113–116. IEEE, 2002.

[13] Roland S Johansson and J Randall Flanagan. Coding and use of tactile signals from the fingertips in object manipulation tasks. *Nature Reviews Neuroscience*, 10(5):345, 2009.

[14] Mohsen Kaboli, Philipp Mittendorfer, Vincent Hügel, and Gordon Cheng. Humanoids learn object properties from robust tactile feature descriptors via multi-modal artificial skin. In *2014 IEEE-RAS International Conference on Humanoid Robots*, pages 187–192. IEEE, 2014.

[15] Oliver Kroemer and Gaurav Sukhatme. Meta-level priors for learning manipulation skills with sparse features. In *International Symposium on Experimental Robotics*, pages 211–222. Springer, 2016.

[16] Oliver Kroemer, Christoph H Lampert, and Jan Peters. Learning dynamic tactile sensing with robust vision-based training. *IEEE transactions on robotics*, 27(3):545–557, 2011.

[17] Ian Lenz, Ross A Knepper, and Ashutosh Saxena. Deepmpc: Learning deep latent features for model predictive control. In *Robotics: Science and Systems*. Rome, Italy, 2015.

[18] Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-end training of deep visuomotor policies. *The Journal of Machine Learning Research*, 17(1):1334–1373, 2016.

[19] Sergey Levine, Peter Pastor, Alex Krizhevsky, Julian Ibarz, and Deirdre Quillen. Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection. *The International Journal of Robotics Research*, 37(4-5):421–436, 2018.

[20] Hongzhuo Liang, Shuang Li, Xiaojian Ma, Norman Hendrich, Timo Gerkmann, and Jianwei Zhang. Making sense of audio vibration for liquid height estimation in robotic pouring. *arXiv preprint arXiv:1903.00650*, 2019.

[21] Beth Logan et al. Mel frequency cepstral coefficients for music modeling. In *ISMIR*, volume 270, pages 1–11, 2000.

[22] Brian McFee, Colin Raffel, Dawen Liang, Daniel PW Ellis, Matt McVicar, Eric Battenberg, and Oriol Nieto. librosa: Audio and music signal analysis in python. In *Proceedings of the 14th python in science conference*, pages 18–25, 2015.

[23] Ioanna Mitsioni, Yiannis Karayiannidis, Johannes A. Stork, and Danica Kragic. Data-driven model predictive control for food-cutting. *CoRR*, abs/1903.03831, 2019. URL `http://arxiv.org/abs/1903.03831`.

[24] Lerrel Pinto and Abhinav Gupta. Supersizing self-supervision: Learning to grasp from 50k tries and 700 robot hours. In *2016 IEEE international conference on robotics and automation (ICRA)*, pages 3406–3413. IEEE, 2016.

[25] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. *arXiv preprint arXiv:1804.02767*, 2018.

[26] Serpil Sahin and Servet Gülüm Sumnu. *Physical properties of foods*. Springer Science & Business Media, 2006.

[27] Stefan Schaal. Dynamic movement primitives-a framework for motor control in humans and humanoid robotics. In *Adaptive motion of animals and machines*, pages 261–280. Springer, 2006.

[28] Stanley Smith Stevens, John Volkmann, and Edwin B Newman. A scale for the measurement of the psychological magnitude pitch. *The Journal of the Acoustical Society of America*, 8(3):185–190, 1937.

[29] Shinya Takamuku, Gabriel Gomez, Koh Hosoda, and Rolf Pfeifer. Haptic discrimination of material properties by a robotic hand. In *2007 IEEE 6th International Conference on Development and Learning*, pages 1–6. IEEE, 2007.

[30] Tzutalin. Labelimg. Git code. `https://github.com/tzutalin/labelImg`, 2015.

[31] Thiemo Wiedemeyer. IAI Kinect2. `https://github.com/code-iai/iai_kinect2`, 2014 – 2015.