SNACBot

Audrey Cooke, Tom Krolikowski, Hersh Vakharia

***Abstract*—SNACBot utilizes a robot arm to deliver food to a person autonomously. The robot uses computer vision to locate food on a surface, and navigate the arm toward it to pick it up. SNACBot will also detect a user with an open mouth and deliver the food to their mouth safely.**

I. Introduction

SNACBot utilizes a robot arm to deliver food to a person autonomously. The robot uses computer vision to locate food on a surface, and navigate the arm toward it to pick it up. SNACBot will also detect a user with an open mouth and deliver the food to their mouth safely. This allows disabled users to enjoy a meal on their own without the assistance of a caregiver.

II. Related Work

At the University of Washington, similar work has been done in regards to automated feeding [1]. The research focused primarily on picking up food with a fork. First, a CNN (convolutional neural network) would identify each of the food items on the users plate. Once identified, images of the chosen food item would be passed through a second CNN that calculates the best angle at which to pick up the food. Finally, with help and guidance from another CNN, the item would be delivered to the user.

III. Methodology

A. Overview

For the food and face detection, a camera with depth measurement capabilities will be attached to the end of the robot arm. The image data and depth information will be used to determine the location of food relative to the robot. The arm will then execute a plan to grasp the food. Once the food is grasped, the arm will point at the user's face. The camera and depth information will determine the location of the user's open mouth, and compute a plan to safely navigate toward it to deliver the food. Fig. 1 shows a simplified routine of SNACBot.

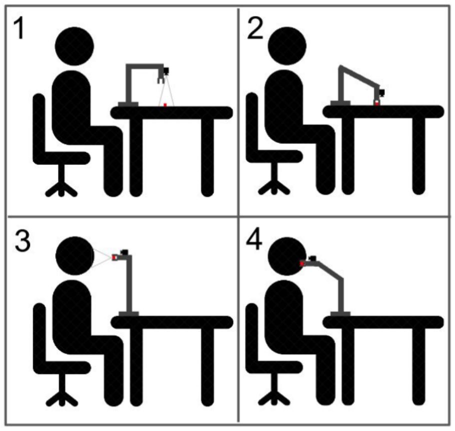

Fig. 1: Overview of SNACBot functionality.

B. Hardware

The SNACbot will be built on a 5-axis robotic arm with a tong-like gripper, configured as shown in Fig 5. Dynamixel servos will be used to actuate rotation along the axes prescribed in Fig 5. Axes E1-E4 are driven by MX-28 servos, with E2 being driven by 2 MX-28 servos. Axes E5 and the gripper are driven by AX-12 servos. Dynamixel servos utilize a UART-based serial communication protocol, and can be daisy-chained to allow for clean wiring. A USB interface called the USB2AX is used to control the servos by sending serial data over USB.

We used an Intel RealSense camera in order to perform environment mapping and drive computer vision. The RealSense camera features a 1080p RGB sensor, and uses active stereo infrared technology to calculate a depth map which can be used for mapping and object distance detection. The RealSense camera was positioned on the gripper section of the 5-axis robot arm so that it may be dynamically positioned to gather data necessary for state detection and movement algorithms.

C. CAD Design

In order to create SNACBot, our group created the 3D model on Fusion 360, as shown in Fig 2. Doing so allowed us to set movement constraints to test movement of the robot. This also gave us the ability to export the model to STL files and create a URDF (unified robot description format) file of the robot to obtain the various coordinate frames on the arm and how they are connected. These coordinate frames can be seen in Fig. 3.

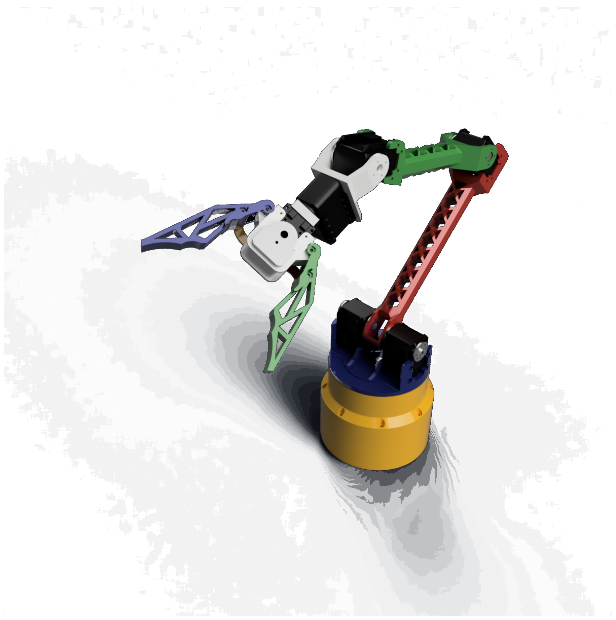

Fig. 2: The final CAD model of SNACBot designed in Fusion 360

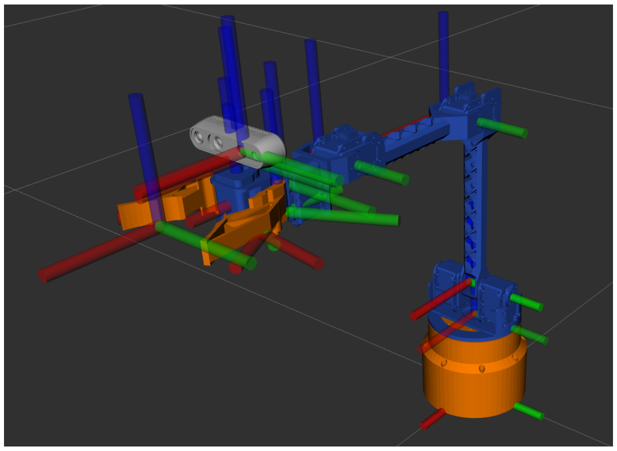

Fig. 3: Image showing the separate coordinate frames of SNACBot's URDF

D. Software Architecture

SNACBot's software is implemented using the Robot Operating System (ROS) middleware (version Noetic). ROS provides software frameworks that allow for control and signal data flow in a robot. The majority of our hardware has existing ROS drivers, which simplified integration. For motion planning and kinematics, we will utilize a highly capable ROS motion planner framework called MoveIt.

To ensure that the neural networks only run when necessary, we implemented two ROS services to run only when requested. For example, the CNNs only need to be run when trying to pick up the food and when delivering it to the user; running it during any other time would be computationally wasteful.

In ROS, we run nodes for the Dynamixel servo controller, MoveIt, and the Realsense camera. The robot's state machine is not a node, but utilizes the MoveIt API and our object and face detection services and then gives the robot commands. Fig.4 is a diagram of our ROS architecture.

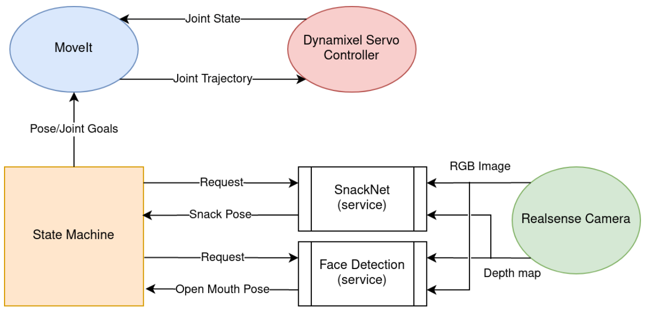

Fig. 4: ROS architecture diagram.

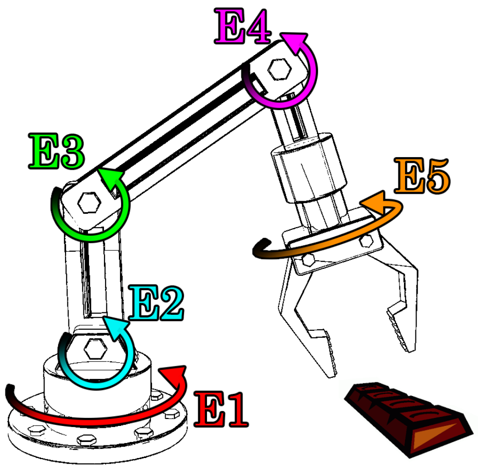

Fig. 5: Diagram demonstrating rotational axes of the 5-axis robot arm to be used in the SNACBot project.

E. Face Detection

To accomplish the face detection and mouth detection, we utilized two existing pre-trained neural network, both of which are found in the dlib python library [2]. The first neural network uses a histogram of oriented gradients (HOG) and an SVM classifier as the final layer. The HOG portion of the neural network relies on images that are filtered into gradient images. Therefore, this required converting each frame into a grayscale image before applying it to the forward pass of the model. After completing the forward pass, the output is a list of portraits, one for each face identified in the image. These portraits are then passed through a second neural network to map facial landmarks to the face.

The landmarks on each face were used to determine two things, center of the mouth for the robot arm to travel to and whether or not the mouth is open. The first of which is done by taking an average of the coordinate on the image. Detection if a mouth is open was done by taking the distance of the vector horizontal to the mouth and the vector vertical to the mouth and computing a ration between the two. Through experimentation, if the ratio was greater than 0.25, then the mouth was open, otherwise it was closed.

One the proper information was obtained, the center of the mouth along with the depth at that point are converted using the realsense cameras coordinate transform, then passed through the URDF coordinate transforms as well. This converts the mouth location into word coordinates for the robot to move to.

F. Snack Detection

Similar to the University of Washington paper, we used a convolutional neural network for object detection of the food [1]. However, instead of using a second neural network to identify the angle and orientation to pick up the object, we used traditional computer vision methods.

Using a RealSense Camera and it's position on the arm, the depth from the arm to the object can be measured. Moreover, the x and y coordinates of the object can be retrieved from the output of the CNN to determine the center of the bounding box surrounding the item. Once the pose is determined, the arm will start moving the gripper closer to the object while simultaneously running the CNN and grabbing the z coordinate from the depth map to determine if it's reached the target pose.

1) YOLOv5: In order to perform object detection of the snacks, we used transfer learning on a pre-trained YOLOv5 Nano single shot multiBox Neural Network [3]. The biggest constraint of this method the absense of an existing dataset that of the food items we want: oreos, cheetos, and whoppers. Instead, we took 749 photos and hand labeled each of the bounding boxes using Roboflow, a machine learning website for managing datasets [4]. From the 749 labeled photos, 523 were reserved for training, 150 for validation, and 75 for a final test, this way we avoid the potential for overfitting. To futrther expand our training data, we applied augmentations such as rotations between $-15°$ and $15°$, shear ±15 horizontal and ±15 vertical, hue $-25°$ and $25°$, and Guassian blur up to 2px. Each augmentation was randomly applied to each image twice allowing us to increase our training set by three times to a total of 1569 training images. The various data augmentations can be seen in Fig. 7.

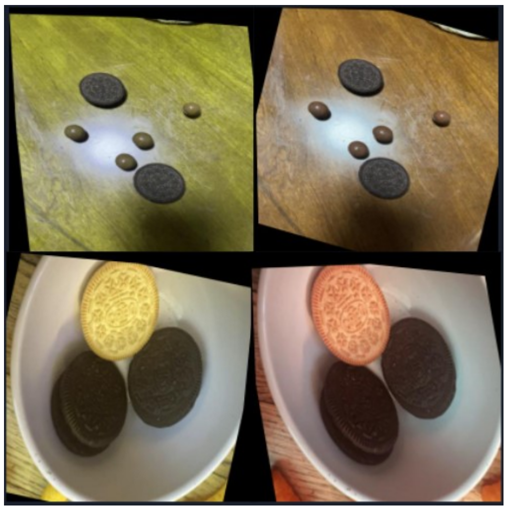

Fig. 7: An example of the different data augmentations randomly applied to the training dataset over 150 epochs.

The snack detection model was trained using a stochastic gradient decent optimizer for a total of 150 epochs. The model progress was tracked using the website Weights & Biases and can be seen in Fig. 6 [5]. By viewing the training loss for the box loss, object loss, and class loss from Fig. 6 and contrasting them with the validation counterparts, the validation curves are flattening while the training curves are still steadily decreasing. Therefore, we have not began overfitting where the validation losses start increasing while the training losses decrease. Moreover, graphs of the models precision and recall over each epoch of training can also be seen in Fig. 8. The mean average precision under an intersection over union (IoU) threshold of 0.5 and under IoU thresholds 0.5 to 0.95 can also be seen in the same figure. The most important metric from this that we used to measure our model performance was the mean average precision with an IoU threshold of 0.5 which was 0.98. This final result can also be seen in Fig. 8 where each class in the dataset is graphed along with the average between them.

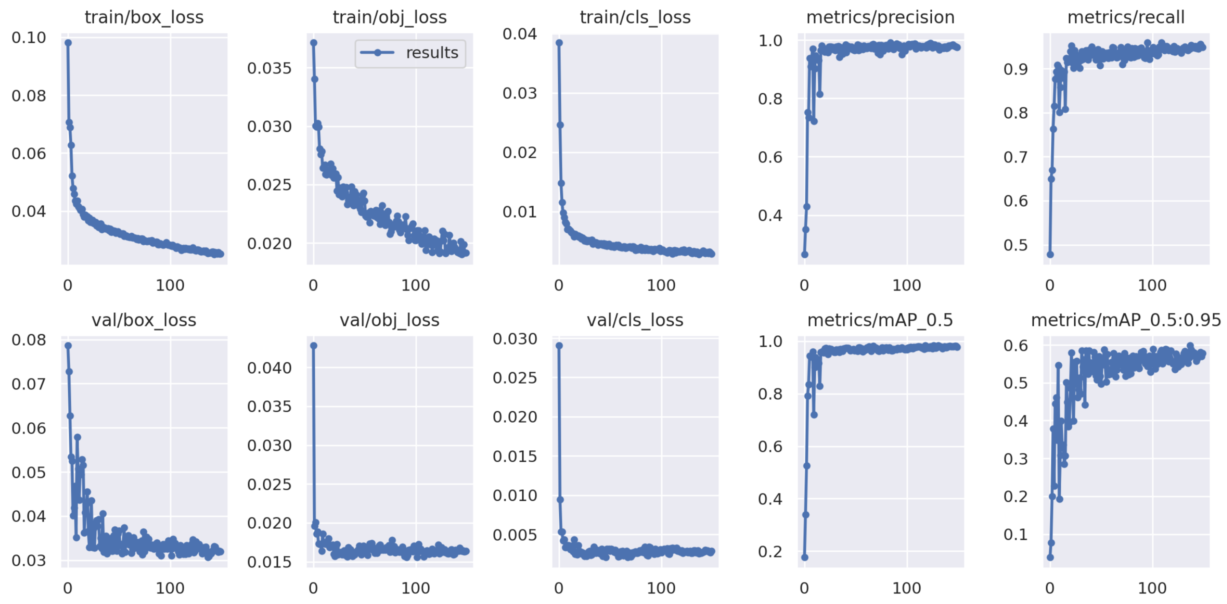

Fig. 6: Training plots obtained from training the YOLOv5 neural network on the custom dataset.

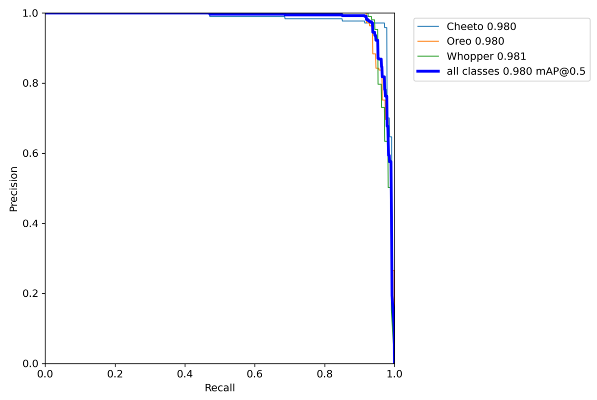

Fig. 8: The area under the precision-recall curve (AUPRC) graph under an IoU threshold of 0.5 for each class in the dataset and their mean.

2) Segmentation: Our first implementation of segmentation involved using a basic Sobel filter to obtain the edges of the image. However, this was subject to noise and overlap of other object. Our next method was implementing an algorithm that uses k-mean clustering to segment the different sections of the image to isolate the object in focus. The first attempt was using k-means with 3 clusters on the aligned depth map. This was successful for most of our test cases except when an object wasn't laying flat. To overcome this issues, we decided to try appending the 3 channels of the RGB image to the depth map and running k-means clustering with 4 clusters. Our idea was that the colors from the RGB image would provide useful information for the model to overcome the case where the object wasn't flat. This improved the result, however, if objects were placed on similarly colored object, it had poor results. After these approaches, we felt it would be better to transition to an analytical method as opposed to an unsupervised learning approach. Fig. 9 shows the results of our different segmentation techniques.

For our analytical approach, we cropped the depth map by the bounding box provided from the object detection step. Then, we took each of the pixels on the edge of the map and added their depth if their value was greater than a threshold of 20mm. Afterwards, we used least squares regression to fit the coordinates to a plane. Once the ground plane was established, each measurement in the depth map is checked. If the measurement is greater than than a defined threshold of 20mm and the distance from of of this measurement to the calculated ground plane is greater than 10mm, then the measurement is kept in the segmented image. Otherwise, the measurement is ignored. From this segmented image, a center of mass is calculated, and eventually an optimal pose orientation is determined.

3) Orientation: After applying segmentation, the optimal orientation of the grasp angle needs to be determined. Our first approach was using the minimum bounding box algorithm, however, it was sensitive to noise from the depth map. Instead, we chose to find the covariance matrix given each point in the segmented image. From this matrix, we solved for the eigenvectors and found that the eigenvector associated with the largest eigenvalue corresponds to the direction the object is

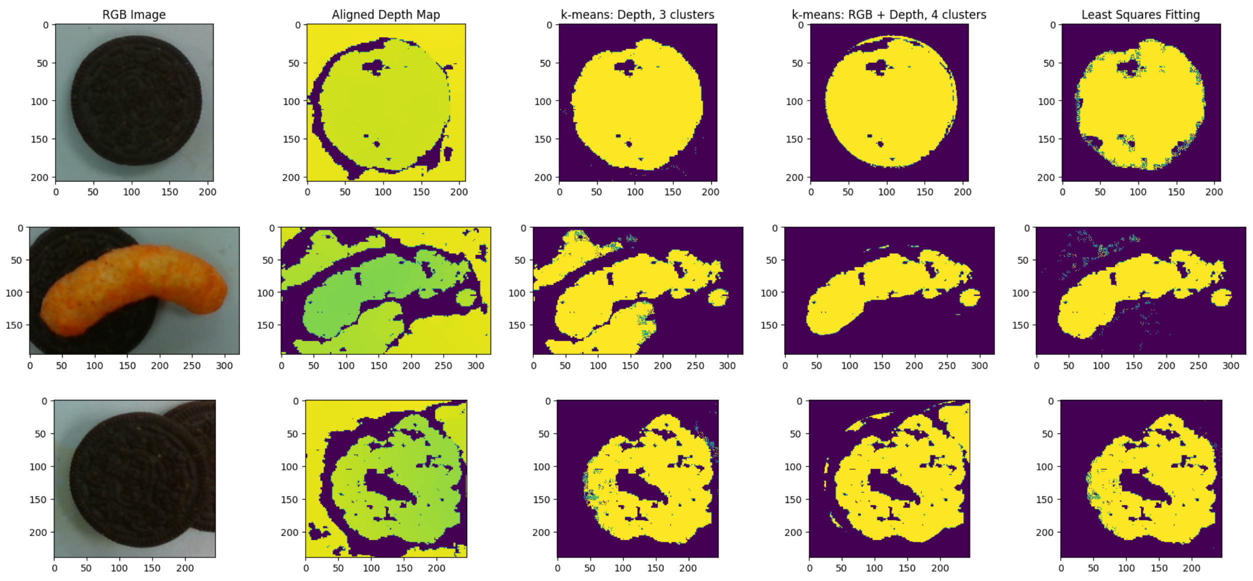

Fig. 9: Three test cases. From left to right, RGB image, aligned depth map, k-means on Depth map with 3 clusters, k-means on RGB and depth channels with 4 clusters, least squares fitting of a plane.

pointing. Therefore, the optimal orientation angle to pick up the object is 90° from this.

G. Motion Planning

The MoveIt framework was utilized to perform the forward/inverse kinematics and motion planning robot. [6]. MoveIt is a powerful framework that integrates into easily into ROS. Initially, we provided MoveIt with our URDF file, and it generated a configuration package that we modified as needed.

We utilize MoveIt's integrated Open Motion Planning Library (OMPL) planners for a sampling-based solution to motion planning. The plans are collision aware, as long as collision bounds are provided by the URDF file.

On top of the OMPL based planning, trajectory smoothing ensures the resulting motor trajectories produce a smooth and stable path without large accelerations. The filter we used was provided by the *industrial_trajectory_filters* package [7].

The built-in inverse kinematics plugins in MoveIt favor 6-DoF arms, which causes problems with our 5-DoF arm. Therefore, we opted to use an inverse kinematics solution called IKFast, by OpenRAVE [8]. IKFast takes a URDF and generates an analytical solution to inverse kinematics. Since SNACBot is a 5-DoF arm, it cannot reach any arbitrary pose. Therefore, an IKFast solution is generation using an IK type called "TranslationDirection5D," where the yaw of the end effector is constrained to be in line with the origin of the robot. Compared to other inverse kinematic methods, an analytical solution produces very fast, stable results.

MoveIt also has to communicate the robot's movement with our low-level motor controllers. For a given motion plan, MoveIt generates a joint trajectory - a list of joint angle positions, velocities, and accelerations that the motors must move with to achieve that motion plan. The joint trajectory is sent to our controller package, *dynamixel_workbench*, which integrates controls the motors based on the given joint trajectories [9]. Fig. 4 shows the flow of data between MoveIt and the other systems.

In our state machine, we can use the MoveIt API to send joint, position, or pose goals to the robot to make it reach a desired end position. Since it is 5-DoF arm, pose goals have to have some parts of the orientation constrained, as another axis is needed to reach an arbitrary pose goal.

H. State Machine

We designed our state machine to start in an initial state where the gripper opens up in case food was left in the gripper during shutdown. Afterwards, the robot enters the search for humans state where it rotates 15° and searches for a human with an open mouth. The robot starts at 0° and rotates to -105° and then changes direction until reaching 105°. Once a human with an open mouth is found, the SNACBot moves to a particular location to search for food. Once in position, the state machine calls the SnackNet service to find the world pose of the highest confidence food object. The robot then moves to that pose to grasp the food. Once food is obtained, SNACBot rotates to the previous angle the human was at and confirms whether or not they're still there with their mouth open. If so, SNACBot calculates the world coordinate to the center of their mouth and moves to that position to deliver the food. Otherwise, if SNACBot doesn't detect a human with a mouth open after 25 tries, the food is returned to the previous pose it was picked up in. Regardless of finding a face or not, SNACBot will return back to scanning for a human with an open mouth. A diagram of the state machine can be found in Fig. 10.

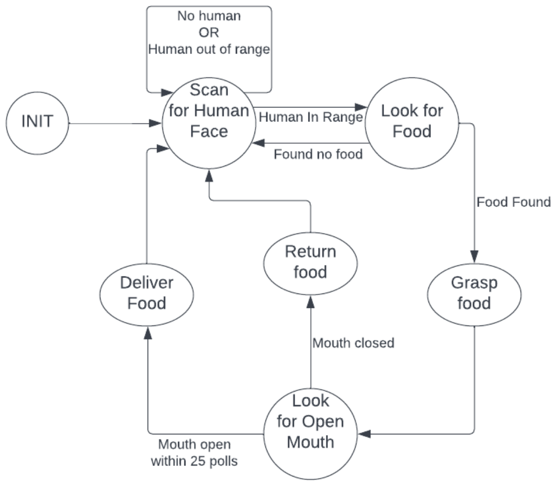

Fig. 10: High level state machine of SNACBot's workflow

IV. Results

Overall SNACBot performed better than we expected considering the time constraints and the small number of available resources.

V. Demo Video Link

Demo video: https://drive.google.com/drive/folders/1kQ-WsziACWKZgInJrIMiQMpFYa_M_rkk

Project code: https://github.com/hvak/SNACBot.git

VI. Discussion and Future Work

After accomplishing all of the initial goals for our project, the future work required is implementing our reach goals. One of which is expanding the neural network to identify liquids in a cup. Then, the arm could grasp the drink and safely transport it to the user for them to drink. In order to accomplish this, the SNACBot would need to be expanded from 5 degrees of freedom to 6 degrees of freedom since the constraints on the yaw position restrict the robot from finding an optimal pose for the human to drink.

Another adjustment that would make our system more robust is the addition of a sensor in the center of the gripper to check whether or not food was successfully grabbed. As of right now, the robot lacks the ability to confirm whether or not it picked up the item so the addition of this sensor could be used to reattempt food detection and grasping if the food wasn't picked up.

VII. Contribution

Hersh Vakharia set up ROS, and implemented the arm control and kinematics using MoveIt. Audrey Cooke designed the CAD, constructed the arm, and developed the grasp pose rotation algorithm. Tom Krolikowski trained and developed the computer vision aspect of the project, including SnackNet and the face detection system. All members contributed to general debugging and the design of the state machine.

VIII. Acknowledgement

Dr. Xiaoxiao Du and GSI Xiaotong Chen provided support throughout the project.

Prof. Peter Gaskell provided the Dynamixel servos and RealSense Camera from the ROB 550 Lab, and offered guidance in their use.

Prof. Shai Revzen provided guidance on least-squares plane fitting and covariance matrix-based grasp angle approach.

References

[1] D. Gallenberger, T. Bhattacharjee, Y. Kim, and S. S. Srinivasa, "Transfer depends on acquisition: Analyzing manipulation strategies for robotic feeding," in *2019 14th ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, 2019, pp. 267–276.
[2] [Online]. Available: https://github.com/mauckc/mouth-open
[3] [Online]. Available: https://github.com/ultralytics/yolov5
[4] [Online]. Available: https://roboflow.com/
[5] [Online]. Available: https://wandb.ai/site
[6] [Online]. Available: https://moveit.ros.org/
[7] [Online]. Available: https://github.com/ros-industrial/industrial_core
[8] [Online]. Available: http://openrave.org/docs/latest_stable/openravepy/ikfast
[9] [Online]. Available: https://github.com/ROBOTIS-GIT/dynamixel-workbench