Software Overview

Year: 2024 Semester: Fall Team: 5 Project: PunchBot

Creation Date: ­01/23/2024 Last Modified: January 27, 2024

Author: Carlos Cotera Email: ccoteraj@purdue.edu

1.0 Software Overview

PunchBot has a couple main requirements for success. From the perspective of a user, here is how the interactions are expected to go, as well as a highlight of where the key software is implemented:

After turning on the device, the user should be able to stand in view of the cameras, within arm’s reach, then a second user should be able to initialize punch detection mode1. At this point, the cameras continually analyze the movement of the player’s gloves and once the start of a punch is detected by the tracking script, we can now trigger the trajectory prediction algorithm2. Here, after quickly estimating the general direction of the punch, the robot should be able to determine an optimal dodging direction. Then, the STM and jetson chips use UART via EtherCAT to give the motors SPI commands3. The first motor is responsible for orienting PunchBot circularly to face a direction, while the second motor is meant to rapidly tilt the target forward to perform a dodge that imitates a real-life boxer by pivoting at an approximation of the hips.

At this point, the punch is completed and the bot will have to re-calculate how to return to its starting point as fast as possible, during the return, we expect that it would be best to continue in punch detection mode, so the bot is ready to interrupt itself and make another dodge, but this might prove to be either unnecessary given the final speed or too complex for the hardware, so we can still determine this later. Once back in punch detection mode we want the main script to cycle between these modes efficiently until the bot is deactivated via user input.

As a potential advancement, we could complete a game design for the project as described in the flowchart in the appendix 1 below, the main machine state functionality that we need to achieve is in appendix 2.

**2.0 Description of Algorithms**

Utilizing the OpenCV library, NumPy, and PyTorch for computer vision. The real-time video processing from a camera feed is done as follows, following the KMeans clustering approach to analyze motion via the kmeans PyTorch import:

Initializing

frame\_resize: Resizes a video frame.

calibration\_circle: Adds a calibration circle to the frame.

K-Means Clustering Functions:

kmeans\_gpu: Applies K-Means clustering to identify objects in the video feed and returns their coordinates and cluster centers.

Initialize parameters like pixel center, camera height, user height, and camera orientation.

Main Loop:

Continuously captures video frames, converting frames to the HSV color space and applying color filtering to identify objects in the specified color range (yellow gloves). With this processed data, we use K-Means clustering to identify the objects' pixel coordinates and cluster centers. We can also track the movement of the identified objects and calculate avoidance maneuvers for a simulated robot.

3.0 Description of Data Structures

Based on our mode functions, there would also be a need of data structures for motion tracking and motion analyzing. One of the key data structures we are taking advantage of for the PunchBot is a displacement vector for the following uses (examples from current testing script):

*img\_2\_vector and vector\_2\_*image: Convert image to vector and vice versa.

pixel\_2\_cords and cords\_2\_pixel: Convert pixel coordinates to real-world coordinates and vice versa.

To track the motion of user, we would need to set data points from the image of the user using camera. Inside this data structure, it will capture and tracks each data points of the user’s glove motion. By comparing x, y, z values, we can detect the punch because user will move their arms forward. In analyzing motion structure, it will calculate possible pathway of the punch based on the data point. I the main loop, punch detection mode captures frames from the camera, processes them, performs color segmentation, and tracks objects using k-means clustering. The tracked object's position and movement are used for some form of robotic control or object avoidance.

Examples of other key data structures:

lower\_bound\_gloves = np.array([20, 100, 100])

upper\_bound\_gloves = np.array([30, 255, 255])

color\_mask\_gloves = cv.inRange(hsv\_frame, lower\_bound\_gloves, upper\_bound\_gloves)

Above, we use np arrays to produce a data structure that can perform vector like operations efficiently and with short, readable syntax. As for object Tracking with k-means:

marked\_pixel\_coords = np.column\_stack(np.where(color\_mask\_gloves > 0)) marked\_pixel\_coords = torch.tensor(marked\_pixel\_coords, dtype=torch.float16, device=device) cord\_list, punch\_pixel\_cord\_centers = kmeans\_gpu(marked\_pixel\_coords, 2, punch\_pixel\_cord\_centers, num\_punches)

This extracts coordinates of colored pixels and performs k-means clustering to identify objects.

The tracked object's centroids are calculated and used for further analysis.

4.0 Sources Cited:

E. Ecosystem, “Understanding K-Means clustering in machine learning,” *Medium*, May 17, 2022. [Online]. Available: <https://towardsdatascience.com/understanding-k-means-clustering-in-machine-learning-6a6e67336aa1>

Appendix 1: Program Flowcharts (camera test) A blue and orange rectangles

Description automatically generated

A blue and orange rectangles

Description automatically generated

Appendix 2: State Machine Diagrams

A diagram of a machine

Description automatically generated