GauchoEats

By
Team Mapaches
Derek Wu,
Darren Hsu,
Lidi Yafei,
Supitsara Kara Suharitdumrong,
Zoe Baral
Vision Statement

The Problem

A large percentage of UCSB students rely on the school’s dining commons for sustenance and nutrition. Unfortunately, the dining halls experience is often muddled with bloated wait times, congestion, and shortage of select meal options. Coupled with rigid class schedules and numerous day-to-day activities, these inconveniences force students to make unwanted compromises (e.g. splitting up parties to find seats and forgoing extra servings for the sake of saving time). Ultimately, the current dining hall situation at UCSB leaves students constantly unsatisfied and frustrated.

Current Solution

To remediate the problem of wait time congestion, UCSB dining provides live camera footage of the dining commons’ entrance. However, as of now, the security cameras do not employ any higher functionality. They merely observe the wait lines outside of dining halls as opposed to actually measuring the saturation within each cafeteria. This does little to resolve the issue at hand as dining halls can be packed even when wait lines are relatively short. The current system offers students a false sense of security which calls for drastic improvements to its stale structure. GauchoEats is Team Mapaches’ innovative take on building upon the solution of the current dining camera system.

GauchoEats

GauchoEats solves this problem by providing continuous metrics in real time, so that students can decide which dining commons is least congested. Some of the metrics that will be implemented include the number of individuals present in the dining commons, the peak periods of activity, and also the “best time” to dine—all without having to constantly visit the Dining Cams website. This project responds to a question such as, “How many people are in Ortega?”, by providing the current number of students and the capacity of the dining halls.

Implementation

GauchoEats will be implemented on the Amazon Echo platform as an Amazon Alexa skill, allowing users to prompt the Alexa receive output via audio or visually through the Amazon Alexa app.

Technologies the implementation will utilize include the UCSB Dining API, Alexa Skills Development Kit and API, Amazon Web Services(AWS) Lambda, Amazon Sagemaker, AWS Elastic Beanstalk, AWS Elastic Compute Cloud, the OpenCV library, and Amazon DynamoDB.

This project will be implemented though Dining Camera video API’s and new core technical advances such as machine learning, video processing, and voice recognition. We will record the real time number of students in a dining commons by the difference of students entering and leaving a building and relay the information via any Amazon Echo-enabled device and the official Amazon Alexa app.
Outcomes

GauchoEats aims to give students easy access to information concerning the dining commons, eliminating long wait times and highlighting ideal dining times and locations. Ideally GauchoEats will also collaborate with UCSB Dining to incorporate its image recognition technology into the Dining Cams web page for students to view useful estimates and metrics alongside footage.

Milestones and Sprints

Metrics/Features:
- Number of people in line
- Average wait times to enter
- Ratio of people in dining hall to capacity
- Peak times of each dining hall each meal of the week
- Estimated dining duration
- Dining hall menu access
- Dining hall announcements addition to Alexa flash briefing
- Dining hall hours reference

Sprint 1:
1. Get access to Dining Hall Cam API through UCSB services
2. Explore Alexa API and research how to incorporate OpenCV with Alexa
3. Setup Amazon DynamoDB to store and compile data scraped from dining hall cameras
4. Research how to implement OpenCV libraries to visually detect and track individuals
5. Prototype Echo’s adaptive responses to variable questions regarding our proposed metrics
6. Construct basic Alexa skill card skeleton w/ ability to display information from DynamoDB

Sprint 2:
1. Setup a server for OpenCV algorithm to process real-time video footage from Dining API
2. Implement additional Alexa skills, including dining hall hours and menu
3. Setup DynamoDB to store the additional menu, hours, and announcements information
4. Adapt skill card skeleton to display images upon certain data requests
5. Address issues and leftover tasks from Sprint 1 Retrospective

Sprint 3:
1. Refine and perfect Alexa’s responses to questions about metrics
2. Final testing of user interaction with mobile display and Alexa device
3. Train Sagemaker to make higher end calculations for the predictive metrics
4. Address issues and leftover tasks from Sprint 2 Retrospective

System Architecture

Overview:

Description:
This graphic above depicts the relational structure of the system architecture. First and foremost, the entire process is initiated when the user invokes a “wake word” that turns on the Amazon Alexa speaker. The Alexa speaker then sends AWS Lambda a request based on the intent parsed by the Alexa API. The lambda function reads from the Gaucho Eats DynamoDB table and forms a response that is sent back to Alexa, resulting in audio feedback and visual feedback via the Amazon Alexa app.

Meanwhile, the DynamoDB table is constantly being updated by the OpenCV image processing model that is hosted on Elastic Beanstalk, who also invokes calls to the Dining Cams API to read camera footage for metric processing.
User Interaction:

Description:
In this sequence diagram, a high level overview of our system architecture is depicted, with the user interacting directly with the Amazon Echo device. The first interaction with the Echo Device is the “wake word” that turns on the device. The speaker prompts the user for an intent, such as “ask GauchoEats how long is the line at Ortega.” Echo then makes a database request with AWS DynamoDB and returns the information back to the user verbally. The next set of interactions describe the information displayed on Alexa skill cards. Simultaneously, the Echo device makes a parallel request for excess information and pushes the returned data onto the mobile device screen. In this manner, the user will receive an auditory and visual response for each verbal intent with the Echo device.
Dining Hall Counter Algorithm Class Diagram (UML):

Description:
In the UML diagram above, the picture describes the class structure for the counting algorithm. The four classes outlined are trackableObject, centroidTracker, metricCounter, and DynamoUI. The base class is defined in trackableObject, which is a representation of each individual detected on the dining hall cameras. The centroidTracker class acts as a container object and stores all the instantiated trackableObjects in an ordered dictionary. All of the numerical information regarding the tracked individuals is pushed to a metricCounter instance and organized. These metrics are then written to DynamoDB via a dynamoUI object.
Person Detection Sequence:

Description:
The sequence diagram above depicts a general computational pathway for the person detection algorithm. The main Python script is run on an AWS server. Once it begins running, it establishes a connection with the UCSB Dining Hall API and immediately instantiates a CentroidTracker. The next portion of the script runs in a while-true loop, grabbing frames from the Dining Hall video feeds one-by-one. A pre-trained Caffe model is then run on the images and detects possible “people” blobs, which are then sorted based on a calculated confidence value. All blobs above a 40% confidence will have a trackableObject instantiated and assigned to it via an ObjectID. These IDs are pushed into the CentroidTracker and registered. With each iteration of the main loop, thereafter, the algorithm assigns registered IDs to the closest identified blobs to emulate “tracking.” If the ID is left unassigned for 40 frames or more, it is deregistered. If the ID passes a predetermined line, it is counted (leaving or arriving). At the end of the loop, the algorithm pushes accrued metrics into the metricCounter and uses the dynamoUI object to transfer the data to the online AWS database. This continues to run until the video feeds cut.
Description:
The above UML diagram depicts the relationships between Gaucho Eats, its metric-related functions, DynamoDB, and DynamoDB’s table items. GauchoEats and its respective functions are hosted within AWS Lambda, while DynamoDB contains the wanted attributes. A request to GauchoEats is invoked by Alexa and the user, prompting Lambda to decide which of the 6 getter functions are appropriate to call. Each getter function will prompt DynamoDB for a respective attribute (line, diningCapacity, menu, announcements, hours), using diningCommon as the key to specify a distinct Dining Common.
Functional Requirements

Use Case 1: Dining Hall Announcements
Actors: Amazon Echo, Student, UCSB Dining Hall API, AWS DynamoDB, AWS Lambda, GauchoEats Skill
Precondition: GauchoEats Skill has been invoked by Amazon Echo.

Flow of Events:

Basic Path:
1. Echo prompts Student for question; uses GauchoEats Skill context
2. Echo prompts Student for dining hall
3. Echo request information for specific dining hall from DynamoDB; uses AWS Lambda
4. AWS Lambda returns information to Echo accrued from UCSB Dining Hall API
5. Echo notifies Student about announcements via flash briefing

Alternative Paths:
1. If there are no announcements available for that day, the announcement mentions do not appear in the flash briefing. The Student is explicitly notified that there are no announcements via the skill card.

Postcondition: The Student has been notified one way or the other about the current day’s dining hall announcements.

Use Case 2: Students in Line
Actors: Alexa/Echo device, Student, AWS DynamoDB, AWS Lambda, Gaucho Eats Skill
Precondition: GauchoEats Skill has been invoked by Alexa/Echo enabled device with an intent to request amount of students in line at a specific dining hall.

Flow of Events:

Basic Path:
1. Student invokes Gaucho Eats skill through the Echo Device
2. Alexa/Echo sends AWS Lambda a formatted request, calling a function to receive the line metric
3. The Lambda function prompts DynamoDB for the line corresponding to the specified dining hall
4. Lambda then formulates the formatted response and sends it back to Alexa/Echo
5. Alexa/Echo parses the response, telling the user how many people are in line at a dining hall and showing an informative card to the user’s Alexa app if possible.

Alternative Paths:
1. If there is no line present at the dining commons or the dining commons is closed, let the Student know.

Postcondition: Echo device reports to Student the amount of people in line at the requested dining commons.

Use Case 3: Dining Hall Menus
Actors: Amazon Echo, Echo Device, Student, UCSB Dining Hall API, AWS DynamoDB, AWS Lambda, GauchoEats Skill
Precondition: GauchoEats Skill has been invoked by Alexa/Echo enabled device with an intent to request the meals that they are serving in the dining hall.

Flow of Events:
Basic Path:
1. **Student** uses the “wake word” to invoke **Gaucho Eats skill** through the **Echo Device**
2. **Echo** sends a push request to **AWS Lambda**, requesting the specifying metric
3. In parallel, **Lambda** calls the **UCSB Dining Hall API** to request an update to the menus and store it in **DynamoDB**
4. **Lambda** processes the query entry from **DynamoDB** and sends it to **Echo**
5. **Echo** parses the response, telling the user the main dishes and desserts and showing an informative card about the detailed menu to the **Echo Device**.

**Alternative Paths:**
1. If the dining halls are closed, it will let the Student know.

**Postcondition:** **Echo device reports to Student the menu at the requested dining commons.**

**Other Functional Use Cases / User Stories**

<table>
<thead>
<tr>
<th>Name</th>
<th>Trello Card</th>
</tr>
</thead>
<tbody>
<tr>
<td>User Story 1: Students in Dining Commons</td>
<td><a href="https://trello.com/c/edlJaYeV">https://trello.com/c/edlJaYeV</a></td>
</tr>
<tr>
<td>User Story 2: Least Crowded Dining Hall</td>
<td><a href="https://trello.com/c/kwszKXMi">https://trello.com/c/kwszKXMi</a></td>
</tr>
<tr>
<td>User Story 3: Ideal Dining Time</td>
<td><a href="https://trello.com/c/o1LOIufb">https://trello.com/c/o1LOIufb</a></td>
</tr>
<tr>
<td>User Story 4: Estimated Dining Duration</td>
<td><a href="https://trello.com/c/FWYGNqF3">https://trello.com/c/FWYGNqF3</a></td>
</tr>
<tr>
<td>User Story 5: Dining Hall Menu</td>
<td><a href="https://trello.com/c/1hLYox45">https://trello.com/c/1hLYox45</a></td>
</tr>
<tr>
<td>User Story 6: Dining Hall Hours</td>
<td><a href="https://trello.com/c/ruDwRu9k">https://trello.com/c/ruDwRu9k</a></td>
</tr>
</tbody>
</table>

**Non-Functional Requirements**

**User Story 1:** As a user, I should be able to ask my Echo device about the dining commons in one sentence without being further prompted by Alexa so that I can ask my question in as compact a format as possible.

**User Story 2:** As a user, I want to be able to phrase questions in different ways to the Gaucho Eats skill so that I don’t have to worry about specifically wording my question; using the skill should feel as natural as talking to a real person.

**User Story 3:** As a user, I should be receiving a response from Alexa/Echo in a timely manner so that the skill is more convenient to use, compared to actually looking at the dining footage on its website.
Appendix

Alexa Skills Development Kit:
A development platform on which programmers can create new Alexa skills for publication.

AWS DynamoDB:
An online no-SQL database used to store all of the dining hall statistics and data collected from the UCSB Dining API.

AWS Elastic Beanstalk:
A service that deploys completed applications that utilize other AWS services. The object detection and tracking algorithm, specifically, runs on this server and processes dining hall footage in real time.

AWS Lambda:
An event-driven serverless computing platform used mainly to facilitate access requests from Amazon Echo to our DynamoDB database. Specific data acquisitions are triggered through Alexa requests and are returned to the user using Lambda functions.

AWS Elastic Compute Cloud:
A virtual computing platform that hosts the image detection model to track individuals coming in and out of the dining halls. The virtual computing instances are run on open source Linux images.

OpenCV:
An open source library filled with computer vision tools that involve object detection and tracking. Provided the base program that tracked moving individuals, which was later upgraded to fit the project specifications.

UCSB Dining Hall:
An API that provides access to the UCSB Dining hall menus, hours, announcements, and video feeds from three of the dining halls (Ortega, Carrillo, and De La Guerra).