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, 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 (real-time)
- Dining hall capacity (real-time)
- Least crowded dining hall
- Dining hall menu access through Alexa
- Dining hall announcements
- Dining hall hours reference
- Specific food item requests
- Capacity and line plots over time visual 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. Setup crontabs to run scripts periodically every day to update DB w/ real-time information

Sprint 3:
1. Implement food item requests and line plots for the Alexa skill
2. Refine and perfect Alexa’s responses to questions about metrics (edge cases)
3. Final testing of user interaction with mobile display and Alexa device
4. Prepare slides and demos for the final presentation

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. Lambda also reads plotted images from S3 Bucket to package into a skillcard and report back to 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, and S3 Bucket is being maintained with image updates by a C++ IO-Interface of Gnuplot, also hosted on Elastic Beanstalk.
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, mealTime, announcements, hours, foodItem, diningCapacityPlot, linePlot), 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, GauchoEats 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</td>
<td>Students in Dining Commons</td>
</tr>
<tr>
<td>User Story 2</td>
<td>Least Crowded Dining Hall</td>
</tr>
<tr>
<td>User Story 3</td>
<td>Ideal Dining Time</td>
</tr>
<tr>
<td>User Story 4</td>
<td>Estimated Dining Duration</td>
</tr>
<tr>
<td>User Story 5</td>
<td>Dining Hall Hours</td>
</tr>
<tr>
<td>User Story 6</td>
<td>Specific Food Item Requests</td>
</tr>
<tr>
<td>User Story 7</td>
<td>Plots For Capacity</td>
</tr>
<tr>
<td>User Story 8</td>
<td>Plots for Line</td>
</tr>
</tbody>
</table>

Note: strikethrough indicates that the function was not implemented
Non-Functional Requirements

User Story 1: As a user, I should expect concise questions and responses. I want to ask Echo about the dining commons in one sentence without being further prompted by Alexa. Additionally, I want the information returned in a manner that is easy to understand and comprehend.

User Story 2: As a user, I want to have a natural conversation with Echo. The natural language processing model has to be robust to different phrasing of questions and I want to feel that I am 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.
Challenges

**Camera Placement:**
The dining hall cameras are placed in fixed positions facing the entrances. Some dining halls, De La Guerra in particular, impede the field of vision of the cameras which cause numerous problems as GauchoEats relies solely on visual tracking to produce metrics.

**Dining Hall Entrance Orientation:**
The original program was optimized by its creator to utilize a bird’s eye view to track human traffic. In addition, people could only arrive and leave from two directions. The dining hall entrances, on the other hand, permitted students to enter from various directions, forcing the team to adapt specifically to each of the dining halls.

**Stale Object Detection Model:**
Due to the lack of a GPU, GauchoEats uses an object detection model with a low level of functionality, training, and efficiency. This resulted in inaccurate and inconsistent tracking.

**AWS Familiarity:**
Before beginning this project, none of the members in this group were remotely familiar with AWS services. For each of the 5 services we used, we spent hours researching, learning, and setting up, draining lots of time from the actual project implementation stage.

**Python Familiarity:**
Python was a relatively new language to most of the team. Since most of the project was implemented in Python, there was a long delay between research and implementation as we needed to familiarize ourselves with Python syntax.

**C++ Integration:**
It was difficult to find a way to integrate C++ or Java into our project as a whole, as the majority of our project required rapid processes (mathematically counting centroids, making database queries). Sprint 3 was when we decided to add a function to the project that will allow us to integrate C++ along with Gnuplot.

**Dining Hall API:**
The API itself proved easy to use. However, for extended periods of time, the data provided by the API (such as dining hall hours) would register as NULL values, breaking most of our scripts. In addition, information such as “announcements” were never provided by the API so we could not necessarily test if our scripts correctly integrate the API into our virtual assistant skills.

**AWS Lambda Collaboration:**
AWS Lambda has not been optimized for multiple users on the same account to collaborate in real time with one another. Changes applied by one user could sometimes overwrite those of another, which resulted in almost no collaboration outside of conversation.
Future Features and Functionality

**AWS Sagemaker AI:**
Using the collected data, we would like to add a machine learning component to our project. Using AWS Sagemaker, we will be able to leverage and analyze the collected data from the video camera footage in our database. Sagemaker will feed the collected data into DynamoDB for us to use and implement inside the Alexa GauchoEats skill.

**Predictive Insights:**
Using Sagemaker, we will be able to analyze the data collected from the dining halls and create predictive insights. Some insights we have in mind include wait times during peak hours and how long an average meal will take.

**Dining Hall Video Stream:**
Our current implementation of GauchoEats includes skill cards viewable on the Amazon Alex App. It displays the verbal articulation of the Alexa response. We hope to include a live video stream of the current dining hall camera when a user asks “How long is the line at Ortega.”

**Google Home Integration:**
Expand our GaucoEats skill onto the Google platform. We will use the Google Home API’s and SDK to implement a similar skill.
Github Commits

Comments:
- Lidi, Kara, and Zoe contributed to each of the commits updating the lambda_function.py file
- Derek accidentally used both of his Github accounts to commit to the project
- Darren and Derek pair programed in the beginning to run the visual detection algorithm
- Darren focused on most of the AWS Instance exploration and setup, API adaptation, and other tasks that do not reflect on Github commits
<table>
<thead>
<tr>
<th>Time</th>
<th>Description</th>
<th>Author</th>
<th>Status</th>
<th>Code</th>
<th>Links</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Commit 1</strong>&lt;br&gt;on Apr 10, 2019</td>
<td>Create README.md</td>
<td>Lidi Yafei - lidiyf</td>
<td>Verified</td>
<td><img src="image1.png" alt="image" /></td>
<td><img src="image2.png" alt="image" /></td>
</tr>
<tr>
<td><strong>Commit 2</strong>&lt;br&gt;on Apr 9, 2019</td>
<td>Update README.md&lt;br&gt;Wiget committed on Apr 9</td>
<td>Lidi Yafei - lidiyf</td>
<td>Verified</td>
<td><img src="image3.png" alt="image" /></td>
<td><img src="image4.png" alt="image" /></td>
</tr>
<tr>
<td><strong>Commit 3</strong>&lt;br&gt;on Apr 8, 2019</td>
<td>Merge pull request #1 from karamus/stable-branch-1</td>
<td>Lidi Yafei - lidiyf</td>
<td>Verified</td>
<td><img src="image5.png" alt="image" /></td>
<td><img src="image6.png" alt="image" /></td>
</tr>
<tr>
<td><strong>Commit 4</strong>&lt;br&gt;on Apr 7, 2019</td>
<td>Create Calculator.py&lt;br&gt;Wiget committed on Apr 7</td>
<td>Lidi Yafei - lidiyf</td>
<td>Verified</td>
<td><img src="image7.png" alt="image" /></td>
<td><img src="image8.png" alt="image" /></td>
</tr>
<tr>
<td><strong>Commit 5</strong>&lt;br&gt;on Apr 6, 2019</td>
<td>Merge pull request #15 from karamus/stable-branch-2</td>
<td>Lidi Yafei - lidiyf</td>
<td>Verified</td>
<td><img src="image9.png" alt="image" /></td>
<td><img src="image10.png" alt="image" /></td>
</tr>
<tr>
<td><strong>Commit 6</strong>&lt;br&gt;on Apr 5, 2019</td>
<td>Merge pull request #14 from karamus/stable-branch-3</td>
<td>Lidi Yafei - lidiyf</td>
<td>Verified</td>
<td><img src="image11.png" alt="image" /></td>
<td><img src="image12.png" alt="image" /></td>
</tr>
<tr>
<td><strong>Commit 7</strong>&lt;br&gt;on Apr 4, 2019</td>
<td>Merge pull request #13 from karamus/stable-branch-4</td>
<td>Lidi Yafei - lidiyf</td>
<td>Verified</td>
<td><img src="image13.png" alt="image" /></td>
<td><img src="image14.png" alt="image" /></td>
</tr>
<tr>
<td><strong>Commit 8</strong>&lt;br&gt;on Apr 3, 2019</td>
<td>Add sample responses for Alexa</td>
<td>Lidi Yafei - lidiyf</td>
<td>Verified</td>
<td><img src="image15.png" alt="image" /></td>
<td><img src="image16.png" alt="image" /></td>
</tr>
</tbody>
</table>

---

**Commit 9**<br>on Apr 10, 2019

Update README.md<br>Wiget committed on Apr 10 | Lidi Yafei - lidiyf | Verified | ![image](image17.png) | ![image](image18.png) |

**Commit 10**<br>on Apr 9, 2019

Update and rename ImageRecog.py to test_imagesRecog.py | Lidi Yafei - lidiyf | Verified | ![image](image19.png) | ![image](image20.png) |

**Commit 11**<br>on Apr 8, 2019

Create Calculator.py | Lidi Yafei - lidiyf | Verified | ![image](image21.png) | ![image](image22.png) |

**Commit 12**<br>on Apr 7, 2019

Update and rename test_sampl.py to testImagesRecog.py | Lidi Yafei - lidiyf | Verified | ![image](image23.png) | ![image](image24.png) |

**Commit 13**<br>on Apr 6, 2019

Create Calculator.py | Lidi Yafei - lidiyf | Verified | ![image](image25.png) | ![image](image26.png) |

**Commit 14**<br>on Apr 5, 2019

Update and rename test_sampl.py to testImagesRecog.py | Lidi Yafei - lidiyf | Verified | ![image](image27.png) | ![image](image28.png) |

**Commit 15**<br>on Apr 4, 2019

Create Calculator.py | Lidi Yafei - lidiyf | Verified | ![image](image29.png) | ![image](image30.png) |

**Commit 16**<br>on Apr 3, 2019

Update and rename test_sampl.py to testImagesRecog.py | Lidi Yafei - lidiyf | Verified | ![image](image31.png) | ![image](image32.png) |

**Commit 17**<br>on Apr 2, 2019

Create Calculator.py | Lidi Yafei - lidiyf | Verified | ![image](image33.png) | ![image](image34.png) |

**Commit 18**<br>on Apr 1, 2019

Update and rename test_sampl.py to testImagesRecog.py | Lidi Yafei - lidiyf | Verified | ![image](image35.png) | ![image](image36.png) |

**Commit 19**<br>on Apr 1, 2019

Update and rename test_sampl.py to testImagesRecog.py | Lidi Yafei - lidiyf | Verified | ![image](image37.png) | ![image](image38.png) |

**Commit 20**<br>on Apr 1, 2019

Update and rename test_sampl.py to testImagesRecog.py | Lidi Yafei - lidiyf | Verified | ![image](image39.png) | ![image](image40.png) |

**Commit 21**<br>on Apr 1, 2019

Update and rename test_sampl.py to testImagesRecog.py | Lidi Yafei - lidiyf | Verified | ![image](image41.png) | ![image](image42.png) |

**Commit 22**<br>on Apr 1, 2019

Update and rename test_sampl.py to testImagesRecog.py | Lidi Yafei - lidiyf | Verified | ![image](image43.png) | ![image](image44.png) |

**Commit 23**<br>on Apr 1, 2019

Update and rename test_sampl.py to testImagesRecog.py | Lidi Yafei - lidiyf | Verified | ![image](image45.png) | ![image](image46.png) |

---

**Commit 24**<br>on Apr 1, 2019

Update and rename test_sampl.py to testImagesRecog.py | Lidi Yafei - lidiyf | Verified | ![image](image47.png) | ![image](image48.png) |

**Commit 25**<br>on Apr 1, 2019

Update and rename test_sampl.py to testImagesRecog.py | Lidi Yafei - lidiyf | Verified | ![image](image49.png) | ![image](image50.png) |

**Commit 26**<br>on Apr 1, 2019

Update and rename test_sampl.py to testImagesRecog.py | Lidi Yafei - lidiyf | Verified | ![image](image51.png) | ![image](image52.png) |

**Commit 27**<br>on Apr 1, 2019

Update and rename test_sampl.py to testImagesRecog.py | Lidi Yafei - lidiyf | Verified | ![image](image53.png) | ![image](image54.png) |

**Commit 28**<br>on Apr 1, 2019

Update and rename test_sampl.py to testImagesRecog.py | Lidi Yafei - lidiyf | Verified | ![image](image55.png) | ![image](image56.png) |

**Commit 29**<br>on Apr 1, 2019

Update and rename test_sampl.py to testImagesRecog.py | Lidi Yafei - lidiyf | Verified | ![image](image57.png) | ![image](image58.png) |

**Commit 30**<br>on Apr 1, 2019

Update and rename test_sampl.py to testImagesRecog.py | Lidi Yafei - lidiyf | Verified | ![image](image59.png) | ![image](image60.png) |

**Commit 31**<br>on Apr 1, 2019

Update and rename test_sampl.py to testImagesRecog.py | Lidi Yafei - lidiyf | Verified | ![image](image61.png) | ![image](image62.png) |

---

**Commit 32**<br>on Apr 1, 2019

Update and rename test_sampl.py to testImagesRecog.py | Lidi Yafei - lidiyf | Verified | ![image](image63.png) | ![image](image64.png) |

**Commit 33**<br>on Apr 1, 2019

Update and rename test_sampl.py to testImagesRecog.py | Lidi Yafei - lidiyf | Verified | ![image](image65.png) | ![image](image66.png) |

**Commit 34**<br>on Apr 1, 2019

Update and rename test_sampl.py to testImagesRecog.py | Lidi Yafei - lidiyf | Verified | ![image](image67.png) | ![image](image68.png) |

**Commit 35**<br>on Apr 1, 2019

Update and rename test_sampl.py to testImagesRecog.py | Lidi Yafei - lidiyf | Verified | ![image](image69.png) | ![image](image70.png) |

**Commit 36**<br>on Apr 1, 2019

Update and rename test_sampl.py to testImagesRecog.py | Lidi Yafei - lidiyf | Verified | ![image](image71.png) | ![image](image72.png) |

**Commit 37**<br>on Apr 1, 2019

Update and rename test_sampl.py to testImagesRecog.py | Lidi Yafei - lidiyf | Verified | ![image](image73.png) | ![image](image74.png) |

**Commit 38**<br>on Apr 1, 2019

Update and rename test_sampl.py to testImagesRecog.py | Lidi Yafei - lidiyf | Verified | ![image](image75.png) | ![image](image76.png) |

**Commit 39**<br>on Apr 1, 2019

Update and rename test_sampl.py to testImagesRecog.py | Lidi Yafei - lidiyf | Verified | ![image](image77.png) | ![image](image78.png) |

**Commit 40**<br>on Apr 1, 2019

Update and rename test_sampl.py to testImagesRecog.py | Lidi Yafei - lidiyf | Verified | ![image](image79.png) | ![image](image80.png) |
Supitsara Kara Suharitdumrong - karatsuh
added print statements to tests
karanth committed on May 13

testing getNs using assert
karanth committed on May 13

improving os, still testing getNs
karanth committed on May 13

testing getNs() with request
karanth committed on May 13

Merge pull request #38 from karanth/bazar-patches-2 (karanth committed on May 13)

there, dynamic test is effectively fixed
karanth committed on May 13

resetting dynamoad test table
karanth committed on May 13

publishing up code
karanth committed on May 13

...final
karanth committed on May 13

Final Final Final push... Par real
karanth committed on May 13

---

Comments on May 13, 2019

- Merge branch 'master' into kandbranch
  karanth committed on May 13
- done
  karanth committed on May 13
- Merge branch 'master' into kandbranch
  karanth committed on May 13
- stopped deleting files
  karanth committed on May 13

Comments on May 17, 2019

- some plotting code
  karanth committed on May 17

Comments on May 18, 2019

- empir insts?
  karanth committed on May 18
- handle?
  karanth committed on May 18

- formats for separate file new working
  karanth committed on Mar 18

- added separate dynamoad functions script to store all useful dynamoad stuff
  karanth committed on Mar 18

- loading items is string for later parsing
  karanth committed on Mar 18

- adding code to label for plot structure
  karanth committed on Mar 18

- getting down to what prints what
  karanth committed on Mar 18

---

Testing plotting
karanth committed 5 days ago

- added print out test sample because it doesn't use our env variables...
  karanth committed 5 days ago

- using exif for file detection
  karanth committed 5 days ago

- plot file testing
  karanth committed 5 days ago

- Merge pull request #32 from karanth/bazarbranch (karanth committed 5 days ago)

---

Comments on May 29, 2019

- added gui.helfsetGeometry on item with createPicture function
  karanth committed 11 days ago

- Merge pull request #31 from karanth/bazarbranch (karanth committed 11 days ago)

---

Comments on June 5, 2019

- Upgrade README.md
  karanth committed 6 days ago

- Merge pull request #30 from karanth/bazarbranch (karanth committed 6 days ago)
Testing

test_DiningAPI.py: Tests the basic usage of the Dining Common API and that it is up and running. Tests the function of receiving days from the API against the datetime python module, making sure that the dates received from the API are accurate. Also tests that we are parsing the requests for the API correctly using the requests module, by playing with the URL strings.

test_dynamoReadUpdate.py: Tests all functionality we use concerning the DynamoDB database. Functions tested are all contained in dynamoFunctions.py, and we tested: dynamoGet(), dynamoUpdate(),dynamo Scan() the three main functions our project use.

test_envVariableTest.py: This simply checks that os.environ[envVariableName] is receiving an actual environment variable, letting us access Amazon Services later on without publicly posting access keys to Github. The checker() test function was used here.

test_hours.py: This test was to ensure that the Dining Common API was appending values we expected to our DynamoDB table, and that we could read the table and formulate a json response by parsing the table. Functions we used here are: createSkillCard(),dynamoGetMap(),doesNotHaveMeal(),hours(), which are all contained in the hoursFunction.py file

test_leastCrowdedIntent.py: This test used the dynamoGet() and leastCrowded() functions. We wanted to test that our leastCrowded() function would return the dining commons with the smallest capacity stored on our database.

test_ObjectDetection.py: This test used sample video files screen captured from the Carillo Dining Hall cameras. The object detection test case tracks how many people enter and leave the dining hall, and compares it to the actual number expected.

test_ObjectDetection2.py: This test case was similar to the first object detection test. Instead of using the Carillo Dining Hall footage, it uses a video from the Ortega Dining Hall.

test_plotting.py: Tests the creation of .dat files by reading from logs in our DynamoDB table using the functions used in the dynamoData.py file, which include: makeDat(), dynamoGet(), find_all_between(), dynamoScan()
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.

Gnuplot:
A command line program that allows the creation of 2-dimensional and 3-dimensional plots of functions and data, as well as saving these plots as .png files.

Gnuplot IO-Interface:
Allows for C++ code to interface with the Gnuplot module, letting us use IO streams to send commands to Gnuplot.

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).
Burn Downs

Burn Down Chart (Sprint 1)

Burn Down Chart (Sprint 2)

Burn Down Chart (Sprint 3)