

Pitstop Pal Proposal

Executive Summary

There are three pieces of real-time information that drivers need: gas, food, and directions. However, the current ways to get this information provide a bad user experience in the context of driving.

Specifically, by querying Yelp, drivers are forced to manually interact with phones in order to find a place to eat or get gasoline. Using the phone in the car for text input is dangerous and inconvenient. **To back this idea up with fact,** according to analysis by distraction.gov, in 424,000 people were injured and 3,154 people died from distracted driving incidents in **2013.** By decreasing the amount of manual user interaction, this app may decrease risk of death or injury for drivers compared to other mobile methods.

Another problem that current solutions have not solved is intelligently providing recommendations based on activity of the user. For instance, Yelp does not provide answers for you given that you are driving. Yelp queries, through the mobile app, only give nearby top-rated local business, filtered by the radial distance from the car. **This does not make any sense** for driving. The driver should not have to turn the car around to get to a destination.

There are many problems with current solutions, and they all boil down to not intelligently providing drivers with the information that they need. Thus, we propose building Pitstop Pal. **To point out the obvious,** its name refers to the pit stops made for food and gas. **The purpose of Pitstop Pal mobile app is to fill the role of an intelligent driving companion.** ~~In other words,~~ this app will **do all intelligent processing of data available on the mobile phone** in order to recommend where a driver can stop to get food or gas.


High-Level Description

Our application focuses on solving issues with the driving experience, through the use of an intelligent driving companion. ~~Currently~~ there are three main pieces of real-time information that drivers need; access to directions, food, and gas. However, current methods require explicit user interaction **that don't necessarily provide relevant information.** For example, a user driving while entering text in a Yelp query is dangerous, and the results are given within a radial distance from the vehicle. This is both inconvenient **and irrelevant,** because it distracts the user from driving and includes results that are in the **opposite direction of travel.** The focus of our intelligent driving companion is to solve these issues. This application will be most relevant to users that do a lot of traveling in areas that are unfamiliar to them. Our app will be helpful in recommending pit stops along their route. This will make for a **better travel experience.** The user will enjoy each pit stop **as if they found the perfect place,** without the hassle of actually looking.


There are many different ways to design an intelligent agent so it can solve these problems. In order to effectively accomplish this, the application needs to address the following criteria. Two

different approaches are discussed below, each of which solves the problems using the outlined criteria.

Design and Engineering Criteria

1. Access to user location
2. Access to restaurant and gas station locations
3. Access to route of travel / direction of travel
4. Recommendation system based on location / route of travel 
5. Reduced / minimal user interaction, for safety and engagement
6. Real-time interaction (both user and application perspective)
7. Behavior prediction, to improve precision of recommendations

Approach 1

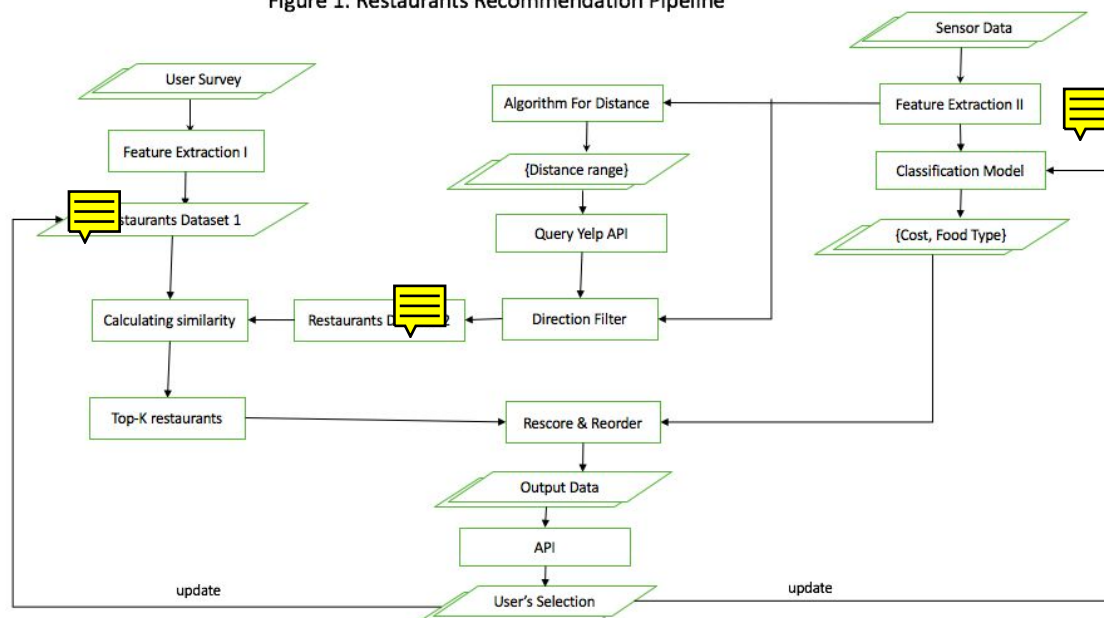
One approach requires the user is traveling. GPS from the user's phone can provide their location (1). The application would require explicit knowledge of a user's destination to know their direction of travel (3 & 7). The application would have a field where the user can provide this. This information can be feed to external API's such as Google Places, Yelp, and MyGasFeed to retrieve relevant information for restaurants and gas station locations (2). Information gathered from these services can be used within our recommendation system. After filtering out restaurants that are not in the direction of travel, Yelp restaurants can be sorted by their star rank (4). Gas stations would be sorted similarly based on cheapest price after querying MyGasFeed's API (4). This would determine which restaurants and gas stations are best, and each would be presented to the user as a swipeable list within the app. The user can then select the location most relevant to them. The selected locations will appear in a map view with the user's location, and from this point the user can choose whether to get directions. This would be accomplished by setting a route within a third-party app, such as Google Maps. The restaurant and gas station results would be queried automatically by the application as the user traveled, will a fast refresh rate. This would allow our application to have reduced user interaction, while maintaining real-time data (5 & 6). 

Approach 2

Approach 2, aims to identify a user's information and provide recommendations more intelligently. The UI is essentially the same as approach 1, where recommendations are presented as a swipeable list within the app. The user can select the location most relevant to them. The selected locations will appear in a map view with the user's location. The user's location is available through the GPS on their phone (1). From this point, the user can choose whether to get directions. This would be accomplished by setting a route within a third-party app, such as Google Maps. The restaurant and gas station results would be queried automatically by the application as the user traveled, will a fast refresh rate. This would allow our application to have reduced user interaction, while maintaining real-time data (5 & 6).

The intelligence of this system is within our recommender, which decides what restaurants and gas stations to make available to the user based on predictions of future behavior. This is mostly accomplished through a combination of behavior models, and general heuristics. Our recommender system plans on using a combination of item-based filtering, food type and cost classification, and a direction filter to determine top food locations. Our pipeline for this is shown below in Figure 1. We will use less complex pipeline for gas recommendation.

Figure 1. Restaurants Recommendation Pipeline



First, we do a user survey to find out what restaurants the user likes (7). In the survey, we will require that **the user enters in 10 restaurants that the user likes** before the user can enter the application. We will take k restaurants from the survey to populate an initial restaurant list for a user. As we will mention later, the output of our pipeline will be the user choosing to go to a restaurant. We will simply add that restaurant to the list of restaurants and will get k + n restaurants, where n is the number of times the user uses the recommendation system (4 & 7). These restaurants will be encoded as Yelp Business objects.

For each time step, we will gather sensor data about the user. The sensor data includes speed, latitude, longitude, temperature, and transport status (driving or walking) (1 & 3). Some of the raw sensor data will be kept as features for downstream processes. However, we will also do feature extraction to build higher level features. The requirements for feature extraction are that they are fast to compute, and the high-level features are more informative than the sensor data (6 & 7). For example, the query distance feature will be calculated as a tabular function of walking or driving. For the sake of this application, we will use our prior knowledge to make the query distance one mile for walking and 25 miles for driving. Another higher level feature will be, "in the city" or "in the country". This will be a function of latitude and longitude.

After feature extraction and during query time, we will classify **what type of place the user wants to eat at, quantified by cost and type**. In our model selection, we will try using SVM with RBF kernel, SVM with linear kernel, and decision tree. The features we plan on using will be the raw sensor data and higher level features. These features include time of day, “in the city” or “in the country”, walking or driver, temperature, speed of travel, and longitude and latitude. The data we will start with to do this classification will be **100 examples** generated from uniform distributions over the features, which will act as a prior. **We choose 100 examples so that we get a uniform prior properly learned by the model**. 100 examples can be overcome by future non-generated data. We will experiment with encoding our prior knowledge of what type of place the user wants to go to based on the **user survey**, which helpfully has labelled data on price and type of restaurant.

Based on distance, we will query the Yelp API, and will build a restaurant list of where the user can go (2). We will filter this list based on the direction the user is traveling and by their travel mode. Specifically, the filter will be applied based on the **identity function of the user driving**. We should not filter by direction if the user is walking. We will filter by a convex quadratic function based on travel direction if the user is driving.

Next, we will compute item-based similarity of restaurants in our **filtered query** list from Yelp and the k+n restaurant list. In other words, using the filtered query list, we will perform a business-business similarity calculation with the user’s stored restaurant list, providing a new order of restaurants based on what places the user will enjoy, through a similarity function.

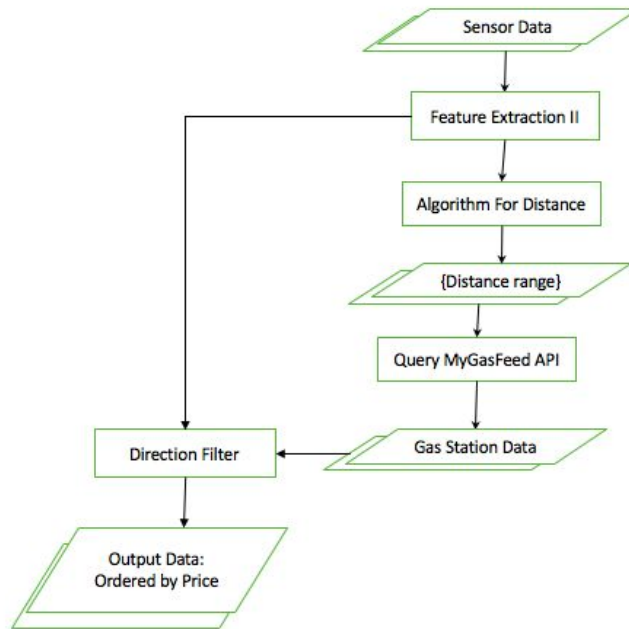
The similarity calculation will be done using Yelp’s database. First, we have the nearby restaurants via the Yelp API using the user’s current latitude and longitude. Then, we will fetch all reviews these restaurants received, which include the Yelp users who wrote them and how many stars they rated the business with. **We will also have the user’s pre-selected restaurant list** and have fetched their reviews. Knowing which users rated each business and how they rated them, we can compute the business-business similarities using a cosine similarity between each pair of vectors with the index being the user id and the value being their ratings. Cosine similarity is appropriate because it is very efficient for sparse vectors.

The **higher the similarity between nearby restaurants and pre-selected restaurants**, the higher on the recommendation list they will appear. Finally, we will compute the top-k similar restaurants.

We will reorder the similar restaurants as influenced by our cost and type, gathered from the classification result. Based on these ordered top k restaurants, the user will pick to go to a recommended restaurant, which will feed back into the pipeline, as written above (7).

A similar, but much more simple pipeline will be used for gas as shown in Figure 2. Instead of querying Yelp, we will query MyGasFeeds API (2). We will order the list by cheapest gas.

Figure 2. Gas Stations Recommendation Pipeline



Our application is going to be based on approach 2, as outlined above. There are several reasons for this, which are derived from our application goal; to enhance the driving experience by making a system that is more automatic and relevant for the user. Each approach is automatic and requires little explicit user interaction. However, the main advantage with approach 2 is its recommender system. This recommender system relies on making behavior predictions of the user, to determine where they are likely to eat / get gas next. This is an improvement on the naive system from approach 1, and should be more relevant to the user by providing recommendations with higher precision.

Implementation Issues

First, we will address some technical issues regarding reliability and production requirements. Reliability will not be an issue if we have our backend on a cloud service and if we have decent backups to APIs of interest. These APIs are specifically referenced above section. This app makes driving more safe than usual because the driver will interact with it less than alternatives. The production requirements are: React Native in order to ship to all mobile platforms, Heroku to host the backend of JavaScript and Python scripts, and some database system like MySQL.

The first implementation issue is deciding when to refresh the places to eat and to get gas. We think a good option is refreshing every 5 minutes, barring that the user has not interacted with the gas or food modules within the previous five minutes.

The second implementation issue is how to filter places to eat and get gas, so the driver does not have to drive backwards to reach those locations. The app could keep a 2D map of latitude,

longitude pairs from the query to Yelp and then draw a classifier orthogonal to the direction that the the car is going. This is especially important on a highway. A naive solution would be a linear classifier with features of a linear equation that removes all points behind the car. A better solution might be a quadratic equation as a classifier, to provide some more flexibility to the classification, while still enforcing that the driver does not drive backwards.

However, what if we don't have access to where the car is going and only have real-time location data? This is a problem because we cannot classify pit stops without further work. To solve this, we will create windows between **where the trip was started** and where the car currently is. The number of windows is a hyperparameter that can be tuned. Then, we will do a locally weighted average of the directional vectors of the windows. The vectors are created by looking at the linear distance between two window points.

A final implementation issue has to do with sizing the map. We will get the map based on passing coordinates to the Google Maps API. However, one implementation issue is how to make the size of the map a good size. This problem can be formulated as a search problem over a continuous domain of two real numbers, assuming a rectangular map of fixed proportions.

We can use A* search to find a size of the map where all possible food and gas stations could appear on the map. We will make the goal test easy to achieve, so this search does not take too much time. The map distance travelled by the car must be a reasonable size. We do not want the map to be too far zoomed out. We can use these constraints to define the heuristic function and partial path cost function. For instance, to define the heuristic function, we could sum the distances of the line segments from each unsatisfied pit stop to the nearest map border where the line segments are orthogonal to the nearest map border, given that the heuristic is admissible.

There are many other implementation issues, but **we am** unable to exhaustively address them all in this proposal.

Way Forward and Schedule

Our project timeline will be divided in three parts. The first part goes from 01/26 to 02/23, which represents Milestone 1. The second part goes from 02/24 to 03/30, which is Milestone 2. The third and last part is from 03/30 to 04/14, which is the final deadline for this project. We will divide our project in different tasks and decide which tasks should be ready by each milestone. Furthermore, we will have dedicated time for testing and debugging right before each deadline. That will insure we have enough time to test our system and not have to postpone tasks.

Part I (01/26 - 02/23)

The first task (task 1) in part I is to write our final project proposal. We will be getting together and agreeing on the specific project ideas, implementation details, work and group expectations. Next we will start two other tasks in parallel. The first (task 2) is to conceptualize

the user experience for our application. That includes thinking about what kind of interaction we want in the app's lifecycle and designing the main views for the app that will include the relevant information for the user. The other task (task 3) in parallel will be collecting online data that will represent the restaurants and gas stations recommended by our system. We will need to understand the possibilities and limitations of using the data sets we find, and make implementation decisions based on what we discover.

Afterwards, each of these tasks (task 2 and 3) will result in a follow up task (task 4 and 5) that depend on them. We will code the basic features for the app's main view (task 4), ensuring we have some functional UI that will allow for testing of our data processing. We will also start connecting the data to the front-end (task 5), providing the app with suggestions. At this point, the suggestions will not be intelligent, but we should be able to get real data connected to the app.

We will stop a 4-7 days before the first milestone to test what we have and ensure that our progress so far is sound and won't need to be worked on again during part II of the project.

Part II (02/24 - 03/30)

We will start part II working on intelligently learning businesses that will appeal to the user (task 6), effectively, developing a recommender. This task will be more focused on restaurants than in gas stations, since they depend more on user taste. It will consist of creating some kind of account for each user where they can set their preferences (task 6.1.) We plan on processing Yelp data and compiling a list of **Yelp reviewers with similar taste than our user** and storing it offline (task 6.2.) When the app is in use, we will access this list and look for positively rated businesses by reviewers in this list that are close to our user's current location (task 6.3.)

Parallely, we will be developing a way to find out which businesses are still ahead of our user's path and which one's the user has already passed (task 7.) We plan on first finding out the direction our user is currently driving to (task 7.1.), based on analyzing difference in location over a short period of time. Then, the service can analyze the nearby suggestions and decide which ones are "ahead" of our user based on their current location and direction (task 7.2.)

Afterwards, based on the functionality of our data retrieval, we will develop new front-end features that will help usability of our application (task 8.) Finally, we will reserve the last week before the milestone for testing and debugging, including using the app on the road and checking for effectiveness of real results.

Part III (03/30 - 04/14)

Part III will consist of finalizing our application and preparing our presentation. We will keep debugging and testing our app during this period (task 8.) Furthermore, we will prepare our interactive demo presentation for the class (task 9), and create a poster that reflects the work we did all semester (task 10.) Finally, we will write our final report with a thorough description of the whole process (task 11.)

Final Presentation

At the end of the semester we plan to have several features not only working but free from bugs. Once it comes time to present we want the application to demonstrate the utility of having

an application that requires minimal interaction but can make quality recommendations to the user on their trip.

With the goal of demonstrating the usefulness of the application, we need to have a fully functional and intuitive menu and main screen. This will require full integration with the Google Maps API. Users must be able to view recommended locations in relation to where the user is located.

While showing potential rest stops on a journey is important to our final presentation, we will be sure to have a functional recommendation system. This system will need to pull restaurant, and gas station information from Yelp. In addition to that the data must be filtered based on what locations are in front of the user's path of motion and match the user's interests.

In order to create a successful recommendation system for our application, we must be able to collect a user's preference in a way that requires minimal interaction. Currently, we plan to have this application collect information by measuring how long a user stays in one location and determining if that location is registered in Yelp. After a certain amount of time in one location, the application should assume that the user is at that business as a customer.

Tools and Materials

As mentioned above in approach 2, we need to obtain data about speed, latitude, longitude, temperature, and transport status. Native GPS can help us to get the longitude and latitude of user location. And we will know speed and transport status of the user via CLLocationManager(IOS) or Detected Activity(Android).

The APIs we plan on using are the World Weather Online API, Yelp API, Mygasfeed API, and the Google Maps API. The first API can provide temperature data. Yelp API and Mygasfeed API provide data necessary for food and gas respectively. The third API will allow us to display a map on the app's UI, which helps the user flexibly pick amongst options.

The framework we are planning on using is React Native, which is a "write once, run anywhere" framework for mobile apps.

Appendix A:
Goals and Relative Importance

Design Criteria	Importance	Will/Expect/Stretch
Access to user location	fundamental	will
Access to restaurant and gas station locations	fundamental	will
Access to route of travel / direction of travel	important	will
Recommendation system based on location / route of travel	important	expect
Reduced / minimal user interaction	important	will
Real-time interaction (both user and application perspective)	important	will
Behavior Prediction	optional	stretch

Appendix B: Milestone Deliverables

task	subtask	ID	PreReq	Who?	time (est)	time spen	Proposal Due				Milestone1				Milestone2				Demo Day	
							1/26	2/2/	2/9/	2/16	2/23	3/1/	3/8/	3/15	3/22	3/29	4/5/	4/12		
							Week0	Week1	Week2	Week3	Week4	Week5	Week6	Week7	Week8	Week9	Week10	Week11		
Proposal	Write-up	1		ALL	7	0														
Designing	UX conceptualization	2		Flavio	10	0														
	Design app views	3		Spencer&Aus	15	0														
	Code home screen	4	3	Spencer	15	0														
Data	Code live screen	5	3	Austin	15	0														
	Pulling data from Yelp	6		Jing	15	0														
	Integrate front-view with data	7	6	Chris	10	0														
Testing	Thorough functionality checking	8		ALL	20	0														
Milestone 1					107	0														
Recommender	User info storage	9		Jing	15	0														
	Compiling list of users offline	10	9	Flavio	20	0														
	Finding businesses online	11	9	Chris	20	0														
Location	Discovering direction	12		Spencer	15	0														
	Determining what is "ahead"	13	12	Austin	15	0														
Front-end	Incorporate features to UI	14		ALL	20	0														
Testing	Thorough integration testing	15		ALL	20	0														
Milestone2					125	0														
Final Stage	Debugging App	16		ALL	20	0														
	User Testing	17		ALL	20	0														
	Interactive Demo Prep	18		ALL	10	0														
	Writing poster	19		ALL	10	0														
	Final Paper Writing	20		ALL	10	0														
Final					70	0														

Milestone 1

Write Proposal

Code home and live screens

Retrieve some sort of data from Yelp

Integrate front-end with data

Start recommender system

Milestone 2

Store info about users

Compile offline list of users

Finding businesses online

Discovering direction

Determining what is "ahead"

Integrate new features to UI

Final
Prepare demo
Make poster
Write report
Test all functionality

Appendix C:

Proposed Set of Software Interfaces to Hardware Components

RESTful Interface:

```
#lat is latitude
#lon is longitude
#returns JSON about each food or gas place- haven't specified yet
#HTTP VERB: GET
/api/gas/:lat/:lon/
/api/food/:lat/:lon/
```

```
#lat is latitude
#lon is longitude
#name is the name of the gas or food place stopped at
#HTTP VERB: POST
/api/gasdatacollection/:lat/:lon/:name/:username
/api/fooddatacollection/:lat/:lon/:name/:username
```

Appendix D: Group Agreement

For the first sprint, Austin, Flavio, and Spencer are responsible for building the homeview, the options view, and the live view. Jing and Chris are responsible for collecting food and gas data based on location and providing a RESTful interface for the front-end team.

Responsibilities by person:

Austin Urlaub: Design app views, code live screen, thorough functionality checking, determining what is “ahead”, incorporate features to UI, thorough integration testing, debugging app, user testing, interactive demo prep, writing poster, final paper writing

Spencer Buja: Design app views, login screen, discovering direction, thorough functionality checking, incorporate features to UI, thorough integration testing, debugging app, user testing, interactive demo prep, writing poster, final paper writing

Flavio Fisman: UX conceptualization, compiling list of users offline, thorough functionality checking, incorporate features to UI, thorough integration testing, debugging app, user testing, interactive demo prep, writing poster, final paper writing

Chris Baier: Integrate front-view with data, thorough functionality checking, finding businesses online, incorporate features to UI, thorough integration testing, debugging app, user testing, interactive demo prep, writing poster, final paper writing

Jing Sung: Pulling data from Yelp, thorough functionality checking, user info storage, incorporate features to UI, thorough integration testing, debugging app, user testing, interactive demo prep, writing poster, final paper writing

Data for each Group Member:

Jing Sung:

Class Schedule -

Time	Monday Mar 7	Tuesday Mar 8	Wednesday Mar 9	Thursday Mar 10	Friday Mar 11	Saturday Mar 12	Sunday Mar 13
8:00AM							
9:00AM							
10:00AM		EECS 388 - 001 Lecture 10:00AM - 11:30AM 1518 CCL	EECS 388 - 001 Lecture 10:30AM - 12:00PM 1670 BEYSTER	EECS 388 - 001 Lecture 10:00AM - 11:30AM 1518 CCL			
11:00AM					EECS 484 - 013 Discussion 11:30AM - 12:30PM 1005 DOW		
12:00PM		EECS 484 - 001 Lecture 12:00PM - 1:30PM 1013 DOW	EECS 498 - 021 Discussion 12:30PM - 1:30PM 2150 DOW	EECS 484 - 001 Lecture 12:00PM - 1:30PM 1013 DOW			
1:00PM							
2:00PM		EECS 498 - 002 Lecture 1:30PM - 3:00PM 1008 FXS		EECS 498 - 002 Lecture 1:30PM - 3:00PM 1008 FXS			
3:00PM	ASTRO 127 - 005 Lecture 3:00PM - 4:00PM 3118 AH	EECS 388 - 014 Laboratory 3:30PM - 4:30PM ARR	ASTRO 127 - 005 Lecture 3:00PM - 4:00PM 3118 AH				
4:00PM							
5:00PM							
6:00PM							

Known Conflicts:

- Grader meeting Thursday at 4:45, grade for exam on Feb 19, April 1 and April 2
- Not in AA from Feb 27 - Mar 3 during spring break

Spencer Buja:

Schedule to Work Around:

Mon 1	Tue 2	Wed 3	Thu 4	Fri 5
	Groundhog Day			
9 AM EECS 388 Lecture	8 AM Physics 241 1209 RAND	9 AM EECS 388 Lecture		
10:30 AM TCHNCLM 498 Lecture		11:30 AM Vijay Nair 277...		
	1:30 PM EECS 498 Lecture		1:30 PM EECS 498 Lecture	2:30 PM EECS 498 STAMPS
	4:30 PM EECS 388 Lab			
	5:30 PM MSAIL Tutorial			
	7 PM MSAIL Readin...	7 PM Msail hack ni...		

Known Conflicts:

- [Machine Learning Research with Honglak Lee](#)
- Michigan Student Artificial Intelligence Lab (MSAIL) Leadership Board (Sunday Meetings at 3)
- Courses

Austin Urlaub:

Schedule:

	Mon 1	Tue 2	Wed 3	Thu 4	Fri 5
all-day		Groundhog...			
10 AM					
11 AM		10:30 AM (1... Eecs 492 Lec		10:30 AM (1... Eecs 492 Lec	
Noon	TCHNCLC...		TCHNCLC...		Eecs 492 Dis
1 PM			Eecs 498 Dis		
2 PM		1:30 PM (1... Eecs 498 Lec		1:30 PM (1... Eecs 498 Lec	
3 PM	2:30 PM (2:... Stats 412 Lec	FXB 1008	2:30 PM (2:... Stats 412 Lec	FXB 1008	2:30 PM (2:... Eecs 496 Lec STAMPS
4 PM		Musperl 30...		Musperl 30...	
5 PM					
6 PM		MSail Tutorial			
7 PM			MSail Hack...		
8 PM					

Flavio Fiszman:

- Schedule (including meetings):

	Sun 4/3	Mon 4/4	Tue 4/5	Wed 4/6	Thu 4/7	Fri 4/8
GMT-05						
9am						
10am		10 – 11:30 WOMENSTD 434	10 – 11:30 SI 301	10 – 11:30 WOMENSTD 434	10 – 11:30 SI 301	
11am						
12pm		11:30 – 12:30p TCOMM 496		11:30 – 12:30p TCOMM 496		
1pm			12p – 1p Flavio and Blu One On One	12:30p – 1:30p EECS 498 DIS		1p – 2p SI 301 DIS
2pm			1:30p – 3p EECS 498		1:30p – 3p EECS 498	
3pm						2:30p – 4:30p EECS 496
4pm						
5pm						
6pm		6p – 8p NQ Staff Meeting	6p – 8p DPE Staff Meeting	6p – 8p CC Shift		
7pm						
8pm						

Chris Baier:

Schedule:

	Sun 2/7	Mon 2/8	Tue 2/9	Wed 2/10	Thu 2/11	Fri 2/12	Sat 2/13
GMT-05							
5am							
6am		6 – 7 Navy PT		6 – 7 Navy PT			
7am							
8am						8 – 10:30 MSIS Work	
9am		8:30 – 10 NAVSCI 402		8:30 – 10 NAVSCI 402			
10am							
11am		10:30 – 12p EECS 270	10:30 – 11:30 TCHNCLCM 496 - 009	10:30 – 12p EECS 270	10:30 – 11:30 TCHNCLCM 496 - 009		
12pm			12p – 1:30p EECS 484		12p – 1:30p EECS 484		
1pm		12:30p – 5p MSIS Work	1:30p – 3p EECS 498 - 002	12:30p – 1:30p EECS 498 Discussion	1:30p – 3p EECS 498 - 002	12:30p – 1:30p EECS 484 Discussion	
2pm				2p – 5p MSIS Work			
3pm						2:30p – 4:30p EECS 496	
4pm			3:30p – 5:30p NAVSCI 402				
5pm							
6pm	6p – 10p NROTC Study Hours	6p – 9p EECS 270 Lab					
7pm							
8pm							
9pm							
10pm							