Smart Wardrobe

Author: Henry Lin, Sung Hyun Back, Yoo Joon Lee,
Electrical and Computer Engineering, Carnegie Mellon University

Abstract

The main goal of this project is to create an integrated clothing management system that manages your clothes and helps you choose outfits. The software will suggest outfit ideas to users based on what clothes the user owns and the user's preferences. Online outfit images will be displayed to visualize the outfit ideas. Once the user chooses an outfit, the outfit will be delivered to them automatically via a rotating hanger design.

***Index Terms* —Design, Neural Network, Web Scraping, Outfit, Automation, Clothing Recognition**

I. Introduction

Picking the right outfits and managing clothes can be an arduous task, yet it's something we deal with everyday. There are many variables that go into choosing our outfits like what our plans for the day are, what the weathers like, and most importantly, what clothes match well together. More often than not, we find these good matches through trial and error, training a clothing matching algorithm within our own brain over years with our clothes. As with most things human, our implicit matching algorithm is flawed. For one, it's difficult to visualize what an outfit might look like without trying it on in the first place. There might be a good outfit in there, we just don't see it. Another flaw is that our outfit choices are directly related to if we can find them and how dirty they are. However, as we own more and more clothes, they become increasingly difficult to keep track of. There are some products on the market that aim to solve these problems, but none are well-integrated (more details in related work).

We propose a new integrated solution, Smart Wardrobe, that provides better outfit visualization for better outfit suggestions and a well-integrated hardware solution to outfit management via a clothing storage and retrieval system. The visualizer will use trending outfit ideas from online as inspiration for the user to pick optimal outfits. All outfits displayed will include clothing articles that the user already owns and will also take the user's preferences into account. Once the user is sure of their outfit, their choices will be sent to the retrieval system that will deliver the outfit directly to them. In order to have good visualizations, the images we find online must match the user's own clothes with a high accuracy and the suggestions must be 100% accurate with the user's preferences. The retriever on the other hand should deliver clothes within 5 cm of the desired dropoff position and it should be able to deliver each article of clothing within 5 seconds.

II. Design Requirements

Our requirements can be broken down into two large subsystems: the visualizer and retriever.

A. Visualizer

TABLE I. Visualizer Requirements

ID	Requirement	Metric
R1	Online Images to User's Clothes Accuracy	color: 90% top-5 clothing category: 90% top-5 other: 50% top-5
R2	User Feedback	direct - 100%
R3	Accurate Images Found	10 per outfit
R4	Runtime	10 seconds max
R5	Disk	Less than 1 GB

R1 and R2 relate directly to our project goals. We use top-5 accuracy instead of standard top-1 because some labels are difficult even for humans to distinguish or could be multi-categorical, eg. magenta vs purple, jean-shorts. 90% accuracy is what's reasonably achievable based on research papers on clothing classification[1]. For feedback, any direct feedback (not inferred) should be correct 100% of the time, eg. If a user rates blue jeans white t-shirt positively, the outfit will be recognized as a positive outfit. It is not a requirement to have high estimated preferences, eg. a user might like blue jeans black t-shirt because they like blue jeans, but it is an important metric to strive for. R3 relates to how we are going to supply trending outfits to the user. By collecting a wide range of photos the user can determine what it will look like and whether they like it. R4 comes from a usability study[2] and R5 is a reasonable size for laptop applications.

B. Retriever

TABLE II. Retriever Requirements

ID	Requirement	Metric
R6	Clothes Tracking	100% accurate
R7	Number of Clothes	20
R8	Distance from User	5 cm
R9	Runtime	5 seconds per article

The system should be able to know what clothes are currently hung in the rack, where they are, and how many times it's been worn 100% of the time (R6). Smart Wardrobe should be able to hang at least 20 articles of clothes. We chose 20 to limit the scope of our project while still providing a good amount of clothes. Because we need to deliver clothes to the user, R8 defines how accurate our system needs to be. R9 is based on the same study[2] as R4, but because we need to deliver two articles, the time is halved for each.

III. Architecture and/or Principle of Operation

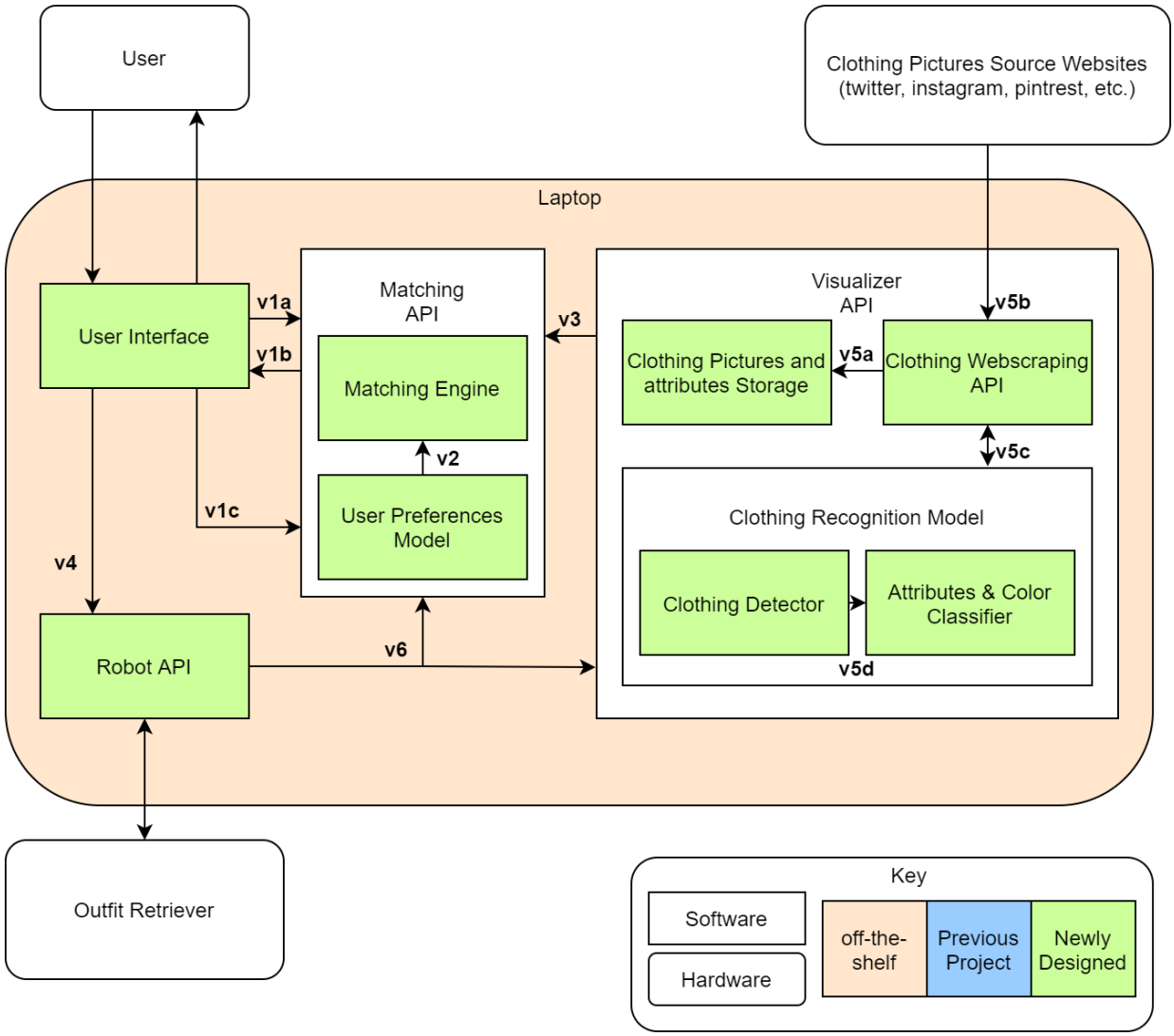

Fig. 1. Software Block Diagram

TABLE III. Software Interface Description

Label	Interface	Description
v1a	setFilter(f)	user-inputted specs like jeans-only or black top
v1b	getMatches()	returns images of the best outfits that meet user specs.
v1c	setRating(p, outfit)	User-inputted rating of a specific outfit combination
v2	getRating(outfit)	returns best estimate of what user's rating of a specific outfit
v3	getOutfitImgs(labels, num)	returns num outfit images that best fit given labels
v4	userInput(type of action, clothes) updateDatabase()	**userInput**: it will take in the type of action (remove, take, return, add) that the user decides, and it will take in what clothes that the user has decided to remove, take, return, or add **updateDatabase**: it updates the database after the user removes, takes, returns, add clothes from/to the rack
v5a	scrapeOutfits(labels, num)	finds and returns num outfit images that best fit given labels online
v5b	scrapeSite(site)	returns all the images on that website
v5c	getLabels(img)	returns labels of img for both top and bottom
v5d	getAttributes(img, bbox)	gets secondary attribute labels and color
v6	getStoredClothes()	sends existing clothes in the database to matching API and visualizer API

When the user wants to take clothes from the Smart Wardrobe, it will be facing the user interface screen with four options, which will be take, return, add, remove clothes. After the user selects "take clothes", the user interface will make an API call to the matching API, which is requesting information about all the clothes that are currently in the rack. Then the user can set filters on specific clothes he wants (v1a). After setting filters,s users can request a recommended combination of clothes' images to the matching API (v1b). Then the

matching API will account for the user preference by communicating with the user preference model (v2) and determine optimal combinations and request the images for these combinations to the visualizer API (v3). After the visualizer API receives information about the clothes that the user wants to see, it will request the clothing web scraping API to scrape the web for images (V5a). Then the clothing web scraping API will scrape the web for the clothing images (v5b). After the API scrapes the web for multiple images, it will validate the image, whether the image contains the correct information of clothes we are looking for (v5c, v5d). Once this process is done, it will display the image to the user interface, and out of the displayed options, the user selects the combination it likes the most. The user selection will be passed into the Retriever API (v4) and then the angle of rotation would be passed into the servo.

After the user retrieves the clothes, it will tell the smart wardrobe that the process is complete. Then the user interface will deliver this information back to the Retriever API and will update the database (v4). Update database is necessary because if the Retriever API updates the database based on the userInput, there might be an edge case where the clothes are not delivered, but the database is updated. The updateDatabase() will prevent such discrepancy. The final step after updateDatabase() will be the Retriever API letting other APIs know about the updated database (v6).

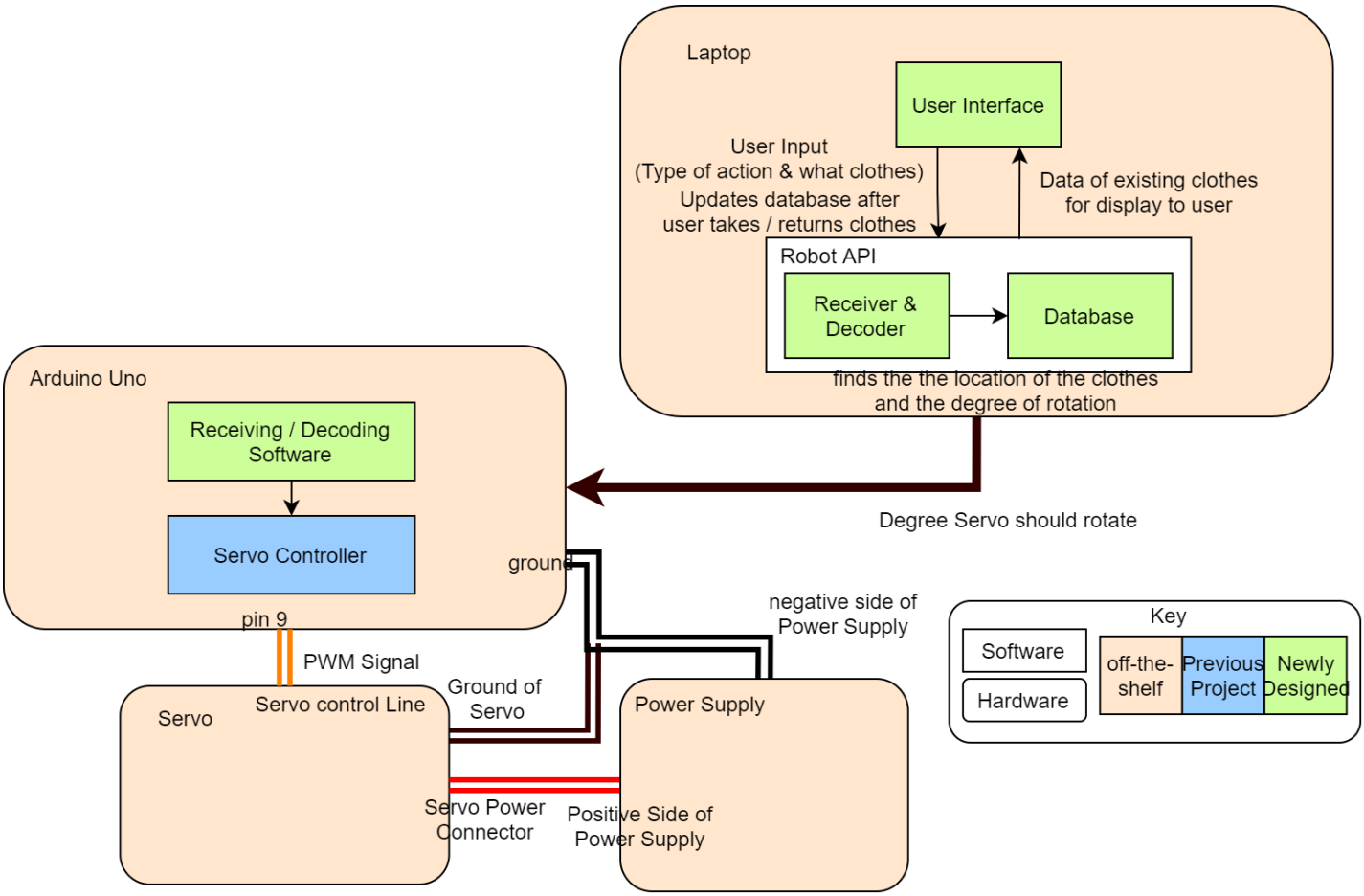

Fig. 2. Hardware Block Diagram

IV. Design Trade Studies

A. Clothing Pictures and Attributes Storage

Because we have both a speed requirement (R4) and a disk requirement (R5), we need to balance our storage subsystem to minimize system runtime and disk space. The following table shows the maximum scope of our system. Assume all images will be sized down to 100 KB.

TABLE IV. Maximum Storage Scope

Maximum Clothing Articles	20
Maximum Outfit Combinations	= 10 tops * 10 bottoms = 100 outfits
Maximum Images Required	= 10 photos * 100 outfits = 1000 images
Maximum Storage Required	= 100 KB * 1000 images = 1MB

This analysis shows that under current requirements, we can effectively store all images necessary for the scope of our system without coming close to our R5 disk requirement. It's even possible to cache the images in memory as typical laptops have gigabytes of memory. Thus, we do not need to worry about the tradeoff between disk and speed as even with the fastest solution we can have, we aren't close to hitting our storage requirements.

B. Clothing Recognition Model

The most important tradeoff for the clothing recognition model is accuracy vs speed. We have both an accuracy requirement (R1) as well as a runtime requirement (R4) that we need to fulfill. A big differentiator for this tradeoff is the architecture we use. There are many state-of-the-art object detection algorithms that offer varying levels of accuracy and speed. Figure 3 shows the performance trade-off of popular state-of-the-art object detectors on the COCO dataset.

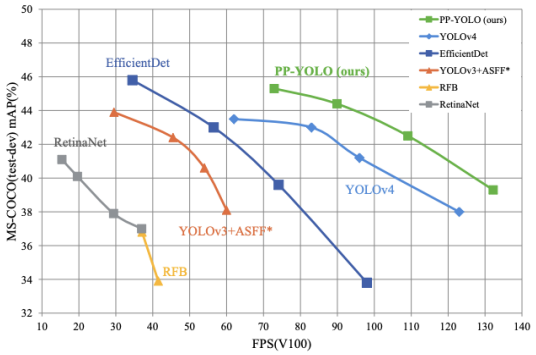

Fig. 3. Performance Graph[3]

Because the figure's definition of accuracy is very different from ours and we're using a different dataset, we will need to train our own models to make an informed decision. At this stage, we are looking at training a YOLOv4 model and an EfficientDet model. We will benchmark the models on the DeepFashion dataset to determine which model we will ultimately be using. We will not be using PP-YOLO as the architecture is not built on tensorflow or pytorch, making the tradeoff between engineering effort and model efficiency not worth it.

C. Outfit Retriever Hardware

When first starting our hardware designs we brainstormed multiple methods of clothing retrieval. Our three initial ideas for retrieval were a rotational hangar, roomba retrieval, and a claw arm. The roomba would navigate to the clothes and then use a vertical actuator to pick up the clothes then bring it to the user. The claw arm would be mounted overhead and grab the clothes like a claw machine and bring it to the user. In order to determine which to use , we looked at degrees of freedom, complexity, budget concerns, and other factors before we chose a design to use.

One of the main reasons we chose not to go with the claw machine was the complexity and budget issues that arose from needing 3 degrees of freedom. The claw machine would need to freely move on the x, y, and z axises requiring 3 actuators to be used. In addition to creating a claw machine we would need to construct a sturdy frame, probably out of metal, that would not move despite all three servos on top moving. Maintaining accuracy of 5 cm (R8) with a frame that had three moving would require construction skills outside of our expertise. In addition the budget concerns rise because we now need three servos and the materials for a sturdy frame.

The roomba did address some of the problems of the claw. Although this solution also required 3 degrees of freedom the roomba was able to provide 2 by itself, and since the roomba was a preexisting material it saved on the budget. This would also only require one vertical actuator placed on top of the roomba. Although we would still need a frame on top of the vertical actuator to retrieve clothing it would not need to be as large or sturdy as the one for the claw as it only moved on one axis. The main drawback came when maintaining the 5 cm accuracy rating (R8). Because the roomba comes with no error correction software in order to eliminate accumulating error we would need to write our own error correction software. This would require training another image recognition model for the roomba on top of the already complex clothing recognition model we were creating. Because of this unnecessary complexity we chose to go with the rotating hanger.

The rotating hanger had one degree of freedom so it removed many of the problems that came with the other designs. In addition the servos we used could detect what angle they were currently at allowing our system to error correct without need to create another image recognition model. We would need a frame but it would be much simpler than the claw machine's frame that would require 3 moving actuators on top of it.

TABLE V. Hardware Design TradeOffs

Idea	Degrees of Freedom	Actuators	Error Correction	Complex Frame
Rotational	1	1	Built in	No
Roomba	3	1	Image Detection	No
Claw	3	3	Built in	Yes

V. System Description and Validation

A. User Interface

When the user first interacts with the Smart Wardrobe, he/she will be guided through a selection screen of the type of action the user would like to perform. The options would be to take clothes, to return clothes, to add clothes, and to remove clothes. User will select the "take clothes" option, when he/she wants to wear clothes that are currently hung on the rack. After the user selects the option, it will lead to the next page, which will contain all the existing clothes that are currently in the Smart Wardrobe. After seeing the existing clothes in the Smart Wardrobe, users can select filters based on the features of clothes. After selecting filters, the user will see a visual description of clothing options that the system has recommended from the Matching API. Then the user can finalize their selection of clothes and the User Interface will send the according request to the Retriever API.

User will select the "return clothes" option when he wants to return clothes that the user has taken. One of the requirements of the "return clothes" option is the clothes that the user would like to return had to be taken from the smart wardrobe. After the user selects this option, it will lead to the next page, which will contain all the clothes that are in the database, but not currently in the rack. User will then select clothes that he is intending to return. After this, the user interface will send the location of the clothes that he is intending to return to the Retriever API.

User will select the "add clothes" option when he/she just bought new clothes and would like to add this to his collection of clothes. If there is no more space left in the rack, the user will get an error message, that he cannot add additional clothes to the system. If there is space, then the system will ask the user detailed information about the clothes he would like to add. After inputting the details, the user interface will send these information to the Retriever API.

User will select the "remove clothes" option when he doesn't want to wear certain clothes anymore and would like to completely get rid of the clothes. After selecting the "remove clothes" option, it will lead to the page, which will contain all the information about existing clothes. User will then choose the piece of clothing he would like to remove and this request will be sent to the Retriever API.

After performing one of these four choices, it will lead into the final screen which will say "Did you finish the process". User will select yes when he/she is done with the action, which will be adding or removing clothes to/from the rack. If the action that the Smart Wardrobe just performed is "return clothes", it will lead to an additional screen, which will have an input box, asking for user feedback.

The most basic validation method for user interface is clicking through each of the options and making sure that the page changes according to the user input. For example, we will ensure with our human eyes that the starting page, there will be 4 options that the users can choose from, and when we set filters, the list of clothes that will be given matches the selected filters. In addition, before connecting the User Interface with Retriever API and Matching API, we will connect the User Interface with a testing API, that will print out all the function calls it has received, to ensure that the correct function calls are being delivered.

B. Matching API

The matching API is divided into two main functionalities; letting the user filter for clothes they want and the matching API giving suggestions based on the user preferences model. When the user would like to take clothes from the Smart Wardrobe, the user is first given a list of clothes that are currently in the rack. Users can then set filters for specific clothes they would like to wear that day by the setFilter() option, which the user can select for eg) blue jeans, black shirt. After the user selects what he wants, we should narrow the combinations down to around 10 per page. In order to do this the matching API will query the user preferences model, to find the top 10 combinations for the user. After this, the matching API will call the visualizer API for images. If the user is not satisfied with the top 10 combinations, it can query the user preferences model which will give the next 10 combinations.

In order to validate the matching API, we can call setFilter() and see if all the clothes that are returned based on setFilter actually satisfies the filter conditions.

C. Clothing Pictures and Attributes Storage

Searching for images that meet the specified labels could be quite a time consuming task. Each image needs to be parsed by our web scraping API and labelled using our recognition model. The entire process needs to be repeated until enough images are found. Despite a long runtime requirement of 10 seconds (R4), we still want to minimize this as much as possible. If the user simply adds an extra filter condition to their search, it should not take another 10 seconds. We thus have a subsystem to store clothing attributes and images. In design trade studies, we determined that disk space is not an issue for our project's scope, thus the design will be simple. Just write new images to disk and retrieve them from disk if necessary. We do not anticipate disk I/O to have a significant impact on runtime, but if it is we will cache images in memory as well.

To validate this component, we will try storing the maximum amount of images necessary (1000) and time how long disk I/O takes.

D. Clothing Web Scraping API

The web scraping API is meant to provide enough outfit pictures to be able to satisfy 10 images per outfit (R3). In order to do this we are using the Selenium Webdriver with Chromium to search through trending outfit databases, including Pinterest, Tumblr, and fashion sites. The main keywords in looking for these databases are outfits, trending, and fashionable. The images found will be then sent to the clothing recognition model to determine the clothing components of the outfit.

If the outfit consists of components the user has it will be registered as a success case and stored until 10 images per outfit are found. In the case where our outfit databases do not have enough photos we can use specific query Google image searches, for example black shirt blue jeans male outfit. We will then parse those results until we fill the quota or have

searched 100 photos to determine we have reached a time limit.

The most important metric for the clothing web scraping api is the quantity of outfit photos we can find. We want 10 photos per possible outfit (R3). Since this api can be run beforehand the runtime concerns can be alleviated and the focus can be placed on quantity. To ensure the quality of outfits we will use trending outfit databases first to find the best quality outfits. We will resort to specific outfit queries on Google in the cases where enough photos can't be found in these databases.

In order to ensure we are finding enough images to meet our requirement we are going to work with small sample sizes to check. This will involve using combinations of 2 tops 1 bottom, 1 top 2 bottoms, and 2 tops 2 bottoms and checking the amount of images generated manually to check we reached the quota of 10 images per outfit (R3). We will check 3 variations of clothes for the 3 combinations to make sure it works on a variety of clothing.

E. Clothing Recognition Model

The purpose of this subsystem is to classify clothing articles so that we can provide users with visualizations of outfits they can make with their own clothes. Because our visualizations use images from online, we need to make sure the outfits consist of clothing that the user owns. The clothing recognition model will use a state-of-the-art object detector to create bounding boxes over clothing articles and categorize it as well e.g. romper, hoodie, t-shirt. There will be 46 categories as that's the number of distinct categories in the DeepFashion dataset.

We also want to augment the categorization with attributes for a better representation of the user's clothes (R1). There are 1000 secondary attributes labelled in DeepFashion, so to avoid cluttering our object detector, attribute classification will be done with a secondary classifier model. Because the object detector will already create bounding boxes for each article. Unfortunately, DeepFashion does not have colors labelled. Instead, we will average the pixels within the bounding box to get the average color. If this method does not produce an accurate representation because of the background color in the bounding box, we will use mask-RCNN to remove the noise.

To validate our model, we will divide the DeepFashion dataset into a training set (75%), a validation set (15%), and a testing set (10%). The testing accuracy must meet the accuracy requirements of R1. Then, we will perform integration testing on our clothing recognition model with our web scraping API. We will ask the API to find 10 images for each random outfit combination we feed it. We will then manually validate the images if the classification is correct. Although web scraped images could be noisier than DeepFashion, we do have the flexibility to ignore lossy images and find other images the model is more confident in. By incorporating this aspect, we should be able to achieve higher accuracy so we should be able to meet, or even exceed, R1.

F. User Preferences Model

As users use our program they will like and dislike different outfits we are showing them. We want to ensure that if a user dislikes an outfit they will never see that variation of outfit again (R2). In order to do this we must take user feedback about each outfit and modify the user preference model based on that.

The mechanism in which this will happen is when the user selects an outfit they will have 3 preference options, like, neutral, and dislike. By choosing like, similar outfits to that will be prioritized in the user interface, and the opposite for dislike. Choosing neutral will not affect the preference of the outfit.

After a few dislikes, neutrals, and likes the model should be able to determine which clothing articles from the outfit are disliked and liked. For example, if the user dislikes a red shirt with blue jeans and a red shirt with black jeans but likes an outfit with blue jeans and chooses neutral for another with black jeans we can set a low priority for the red shirt and not the jeans.

In order to validate this model we can use manual tests where we like, neutral, and dislike 10 outfits each. After each dislike test we can confirm that the object is not suggested and do a similar test for the likes. For the neutrals we can check to make sure that their position in the queue does not change after.

G. Retriever API

Retriever API receives input from the user interface, interprets the request, searches through the database for the location of the clothes, and sends this angle of rotation to the servo. The Retriever API database will be programmed via object oriented programming. Each piece of clothing will be an object which will have information about type of clothing, color, whether it is currently in the rack, and the location. Every time the user adds clothes, a new object will be created in the database. Every time the user removes clothes, the existing object will be removed from the database. Every time the user takes clothes, the currently_in_rack property will be set to False, and when the user returns clothes, the currently_in_rack property will be set to True.

In order to validate the Retriever API, we will be manually inputting consecutive commands of userInput() and updateDatabase(), after which we can print out the database to verify the database is being updated accordingly. In addition, we can also see the rotation of the servos, and verify with our eyes that the servo is rotating to the correct position.

H. Outfit Retriever Hardware

This is the mechanism that is meant to deliver the clothes to the user after they choose an outfit. What we chose was a rotational rack that rotates the clothes of choice to the user. The rotational rack is mounted on a turntable bearing which is then mounted onto a base to support its weight. Under the turntable bearing we are attaching two gears, one attached to the rotating rack and other attached to the servo, as seen in the figure below.

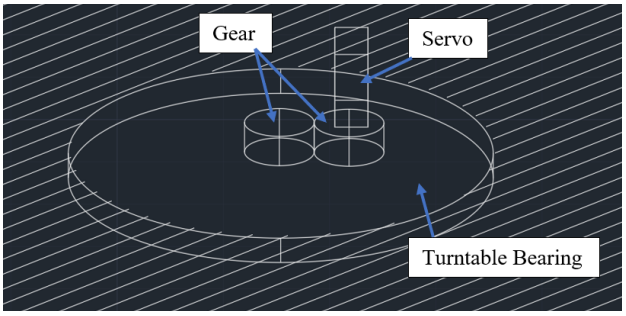

Fig. 4. Rotational Rack Servo Mechanism Displayed Flipped

In order to determine the minimum strength of servo needed we did torque calculations based on our projected load of 35 kg from R7 which can be seen in the table below.

TABLE VI. TORQUE CALCULATIONS

Weight(kg)	35
static friction coefficient	0.01
radius (m)	0.3
torque (Nm)	1.029
torque (oz-in)	145.7064
torque (kg-cm)	10.49198554

Using these calculations we decided to use a 30 kg-cm 360 degree servo to account for error, ensure we can spin the rack a full rotation, and ensure sufficient acceleration to meet our runtime needs. For the gears we have decided on a 1:1 gear ratio because our servo can already turn 360 degrees and we need our rack to turn 360 degrees as well. Additionally, torque should not be a problem.

In order to ensure compliance with our standards of runtime and accuracy we will be performing validation tests to ensure both. Runtime tests will simply involve manually timing runs to ensure the 10 second requirement of R9. For accuracy testing we have determined that the desired clothing should remain within 3° of the pickup location on each side. This degree is translated from the 5 cm of R8 and the 36 in diameter of the clothing rack. We want to ensure that this error doesn't accumulate over runs so to test we will run 100 back to back tests using tape to mark boundaries to ensure accuracy and a timer for runtime.

VI. PROJECT MANAGEMENT

A. *Schedule*

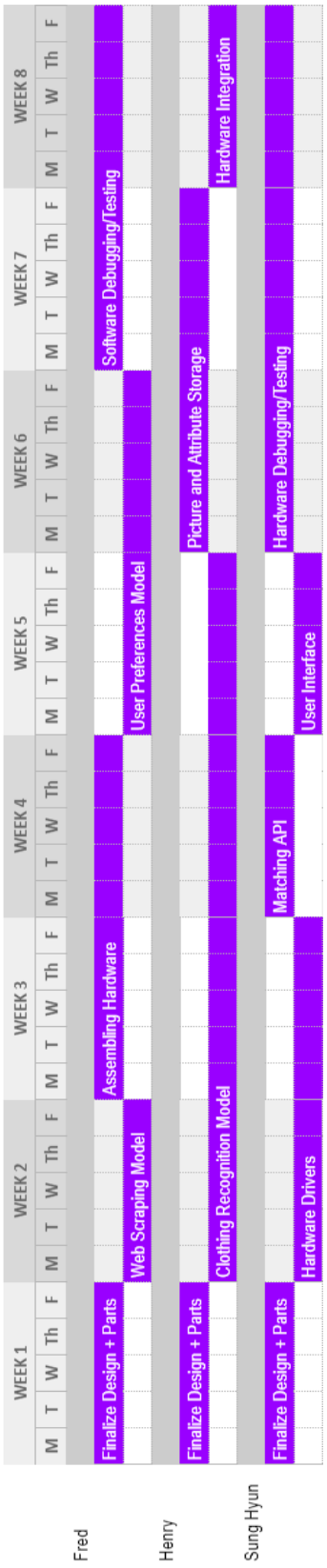

Fig. 5. Milestones

B. Team Member Responsibilities

Team Member	Primary Responsibility	Secondary Responsibility
Yoo Joon	-Assemble Hardware -Web Scraping API -User Preferences Model	-Create Design -Order Parts -Software Debugging
Henry	- Recognition Model -Img & Attr Storage	-Create Design -Integration
Sung Hyun	-Hardware Drivers -Matching API -User Interface	-Create Design -Hardware Debugging

C. Bill of Materials

Bill Of Materials			
Part	Price	Provider	Order Form Sent
Turntable Bearing	24.63	Amazon	Y
Metal-Metal Epoxy	6.99	Amazon	Y
Metal Ring	170	Amazon	Y
Servo	26	Amazon	Y
Power Supply	49.9	Amazon	Y
5 mm Screws	9.49	Amazon	
5 Wood Board (1 in x 12 in x 3 ft)	58.8	Home Depot	
Gear	13.58	Amazon	Y
Total Price	359.39		

We will be writing our software in Python, pytorch for our computer vision, AWS to train our neural net, Selenium Webdriver for web scraping, and github for version control.

D. Risk Management

One risk is our servo choice. The servo needed to have enough power while also being able to rotate 360 degrees. In order to mitigate this risk we did calculations for friction and torque on available constants to ensure the servo we chose would have enough power. However, we recognize that our calculations can only be so accurate and there can also be discrepancies between servo torque listed and how much they can actually generate with load. To mitigate this, our design allows for multiple servos to drive the load at the same time. We also scheduled in a way that allows the member working on hardware to change to a software problem while the new part is ordered. This scheduling allows us to be flexible and not waste time while waiting for a part to arrive.

Another risk moving into the future is the method we are planning to attach our gears to the rotating rack in order for it to spin. We are currently planning on using metal-metal epoxy, a strong glue, to bind the two together. While it does have a strong tensile strength it may not be enough to withstand the pressure of our design. In order to mitigate this risk we have decided on a back up plan of 3d printing parts and gears that allow us to securely attach the two parts. Since this will require more time to implement we have added leeway into our schedule to account for these kinds of problems arising.

In terms of software, the greatest risk is that our ML models trained on DeepFashion will not be accurate enough when it comes to live data. As previously mentioned in our validation plan, we can improve the accuracy of our prediction by removing lossy results as we are free to choose what images we label or not. Regardless, if we can't meet the accuracy we will identify what makes live-images difficult and try to augment our training set to take these differences into account either through data augmentation techniques or by labelling difficult images ourselves. There is plenty of time planned in case we need to do this.

VII. Related Work

The Smart Closet app allows you to input your favorite outfits into a LookBook and plan your outfits for each day of the week. However, it doesn't provide a good visualization of the outfits, nor is it well integrated physically with your closet. Tailor is a better tool that offers outfit management through RFID tags as well as outfit suggestions. It doesn't offer good visualization, only having pictures of the clothes stacked on top of each other, and although the RFID idea is novel, it's a hassle to attach an RFID chip into each article of clothing and it doesn't make physically finding the clothes any easier.

References

[1] L. Ziwei, L. Ping, Q. Shi, W. Xiaogang, T. Xiaoou. DeepFashion: Powering Robust Clothes Recognition and Retrieval with Rich Annotations. 2016.
[2] Jakob Nielsen, Usability Engineering. 1993.
[3] L. Xiang, D. Kaipeng, W. Guanzhong, Z. Yang,D. Qingqing, G. Yuan, S. Hui, R. Jianguo, H. Shumin, D. Errui, W. Shilei. PP-YOLO: An Effective and Efficient Implementation of Object Detector. 2020.