Navify: A Unique Campus Pathfinding Solution

For our project, we knew that we wanted to build something with the potential to solve a problem for students on campus. When brainstorming common problems that we could solve, a recurring theme was the issues that students have navigating on campus. UC Davis' campus is huge; there are many buildings, and beyond that, many rooms within each building. Trying to find a room in a building you've never been in before can be a headache, especially when running late on the way to your first day of classes. Current navigation applications, such as Google Maps, help you find the buildings themself, but they are completely unhelpful in finding unique specific rooms. Furthermore, they often direct you to building entrances that are nowhere near the room you are trying to find, further exacerbating the headache that accompanies the search for the desired room. Because of this, we decided to create a pathfinding website specific to campus, while also integrating room to room navigation capabilities. With our website, you will not only be able to find your way to the building, but also to the specific room you are trying to find. Beyond this, our website takes many situational parameters into account, such as methods of transportation, live weather data, crime data, construction data, and traffic flow trends. Using all of this data, our program ensures that you not only reach your destination, but that you take the right route for your situation, with an accurate time estimate. By adding room to room capabilities and pulling from data that is specifically relevant to students, our program provides more navigation utility to UC Davis students than comparable applications, solving a very frustrating problem for students.

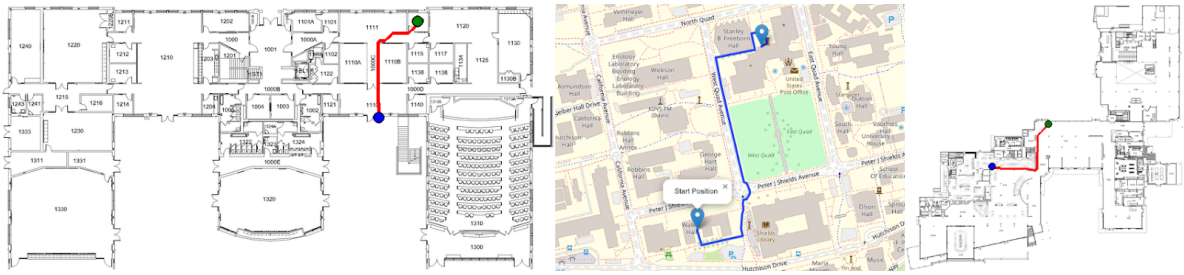

Figure 1: *The paths taken by our pathfinding algorithm between two locations in Walker Hall and Memorial Union.*

The project utilizes the A star pathfinding algorithm to find the shortest path between any two points, computer vision to process the large amount of floor plan data that allows us to pathfind indoors, and Markov chains to calculate approximate traffic flow throughout the day.

The A* algorithm is similar in structure to Dijkstra's shortest path algorithm, except A* is a version of a uniform cost best first search algorithm where we only explore necessary vertices. We choose these necessary vertices using heuristics that inform the algorithm if the steps taken are in the correct direction (we use Euclidean distance from the current node to the goal node as the heuristic for both indoor and outdoor A* pathfinding). In this way, we can cut down on live compute time especially for indoor pathfinding when iterating through individual white pixels.

We use computer vision extensively when processing floor plans in order to automate the process of cropping floor plans, identifying and removing doors for pixel traversal, and identifying and aligning entrance nodes to match a building's indoor floor plan with the same building's outdoor graph representation. Since we have many buildings, doors, and floors of which to process floor plans, we need to automate this process or else spend days tediously poring over each floor plan. Automating this process also creates the ability to easily add new floor plans without prior instruction, as well as making the procedure more friendly to be

replicated in other campuses. We use convolutional filters for identifying door symbols in floor plans with many variations of the door template with the goal of matching the template with each door in each floor plan.

Markov chains are used to calculate the steady state of the outdoor traffic system, assuming that specific nodes (building locations and intersections) are more popular to travel through because of external factors not recorded in our current weight calculations. Markov chains are essentially represented as square matrices in which each row and corresponding column represent an intersection node in the outdoor node graph, where the values in the matrix are the probabilities that an individual traveling from a select node will visit a specific neighboring node. An eigenvector that represents the ratio of travelers at each node at any given time is calculated from this matrix of all intersection nodes in the outdoor graph, which means that if the matrix is multiplied by the eigenvector any number of times (each multiplication representing one step forward in time), the result will be the same eigenvector multiplied by some scalar value, but the ratios of the individuals at each edge will be the same, representing the predicted long-term traffic behavior of the outdoor system, which will be taken into account in the outdoor pathfinding algorithm.

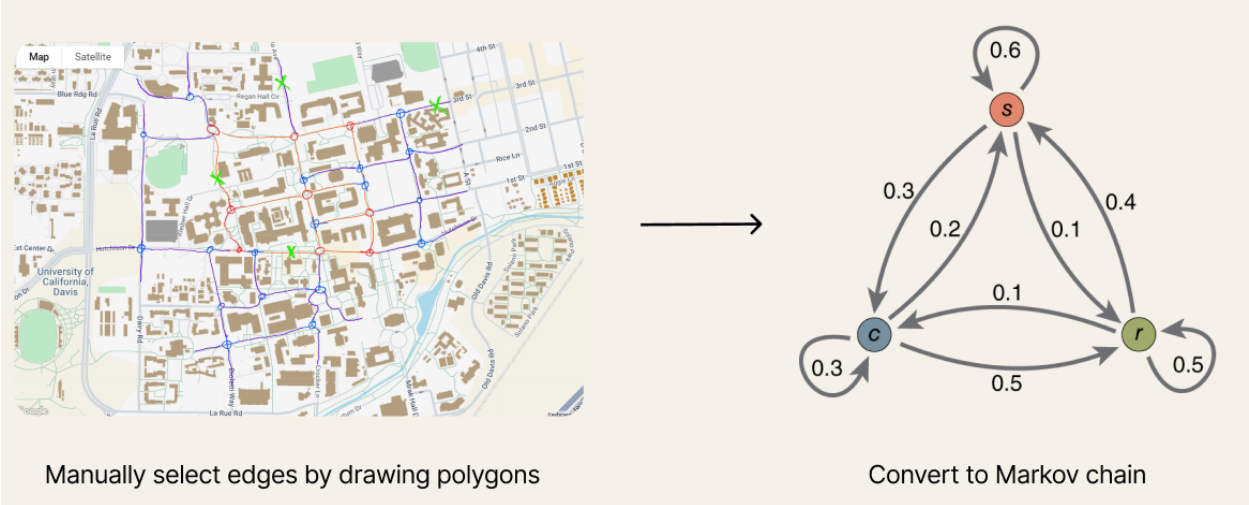

Figure 2: *Showing the process of converting the outdoor campus road system to a Markov transition matrix using a visualization of a simple Markov chain*

The project uses computer vision techniques to process and analyze floor plans, enabling efficient and accurate indoor navigation. The process begins by obtaining a floor plan from the UC Davis architectural database, which is then cropped using the cv module in the OpenCV library to remove unnecessary details and reduce computational load. The cropping technique involves starting at each picture side's center and moving towards the closest black line that denotes the floor plan boundary, cropping to the left and right as we hit that line. The algorithm also removes any floor plan descriptions, and eliminates white spaces to further decrease computational requirements by iterating through white pixels until we hit black pixels: a wall.

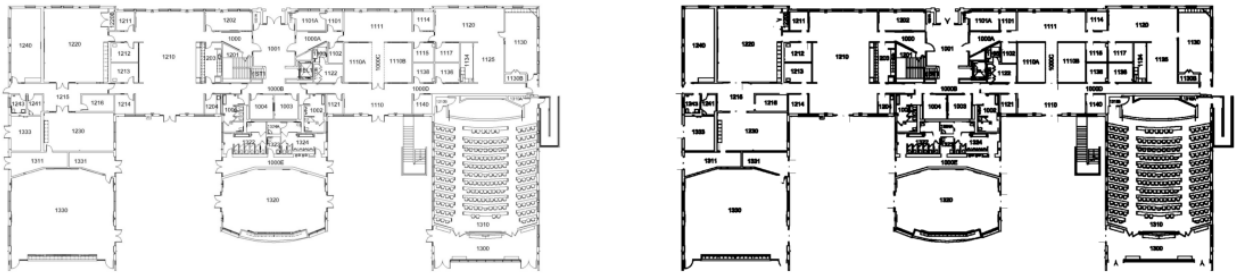

Figure 3: *The cropped floor plan of Walker Hall before and after removing doors*

Once the floor plans are cropped, they are sent to the frontend for users to be displayed in their full resolution. A copy of the cropped floor plan is used in door removal to calculate the shortest room to room path. To achieve this, the convolutional filter based on the smallest door (10px-15px) in the dataset was created to be used on all images that were scaled down to this proportional size for computational efficiency. Then door detection was run across different image resolutions using a grid search to find the highest confidence value. For each scaling, the algorithm rotates the template door at various angles and slightly varies the template size (scale factors of 0.9, 1.0, and 1.1) to get the best predictions for door locations. This detection process uses 'cv2.matchTemplate' to identify positive convolution matches and identify those with a confidence score above a certain threshold as doors. When the detection algorithm is completed, the floor plan will often have multiple overlapping convolution matches due to slight shifts and rotations of the filter. To handle this, the K-D Tree algorithm identifies the final structure and selects the highest confidence detection for each door, through a technique called non-maximum suppression so that there is one detection per door. Once doors are detected, the algorithm turns the black pixels in those areas into white pixels and saves the image scaling information for future reference.

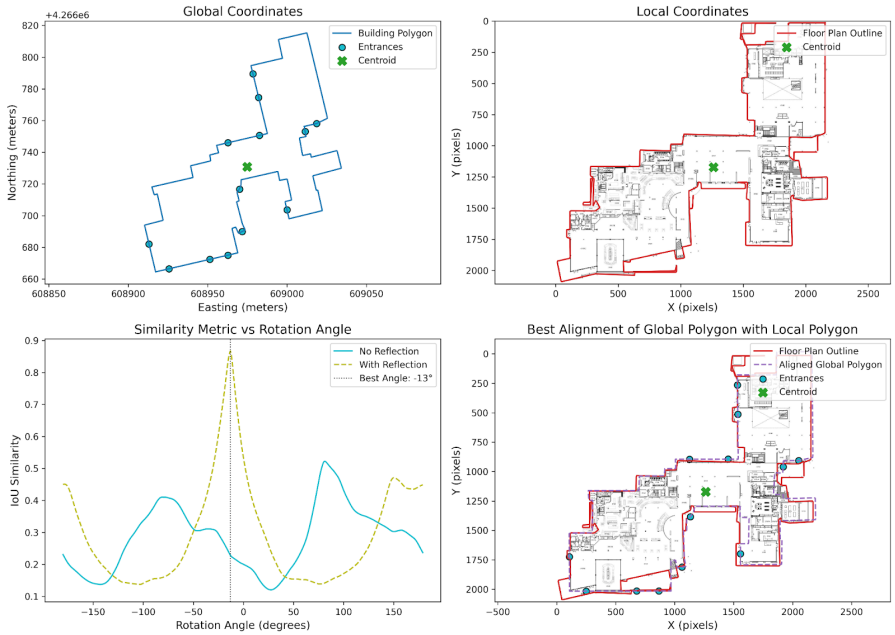

Figure 4: Projection of the global entrances onto the local floor plan: *top left image shows global building with centroid, top right image shows local building with centroid, bottom left image shows intersection union similarity metric for different rotation angles, bottom right image shows projected entrances*

The main goal of detecting entrances is to align their locations on the indoor floor plans with outdoor entrances marked on the Open Street Map. The algorithm aligns the same center of mass coordinates of the floor plan shapes by using lines along the polygon, the floor plan border, and calculating the center of mass using a physics equation. For indoor plans, the floor's outer boundary is detected using 'cv2.findContours', selecting the largest continuous contour and calculating its center of mass. Such a strategy allows to align the indoor and outdoor plans at their centroids.

Afterward, the algorithm scales the global building outline to have the same area since many buildings have complex shapes. Once scaled, the outline will be rotated to match the outdoor entrances, using Intersection over Union (IoU) ratio technique to measure how much the images overlap. This method proved more effective than using average or shortest distances between points. In addition, such a technique implies mirroring the floor plan a couple of times since if the plan has already been mirrored, the algorithm should ensure its alignment with its precise outdoor location. After aligning entrances, they will be moved by 50px inside the building to ensure they are correctly placed for indoor pathfinding. Lastly, the results, showing indoor and outdoor entrances, are stored in a database with their pixel locations.

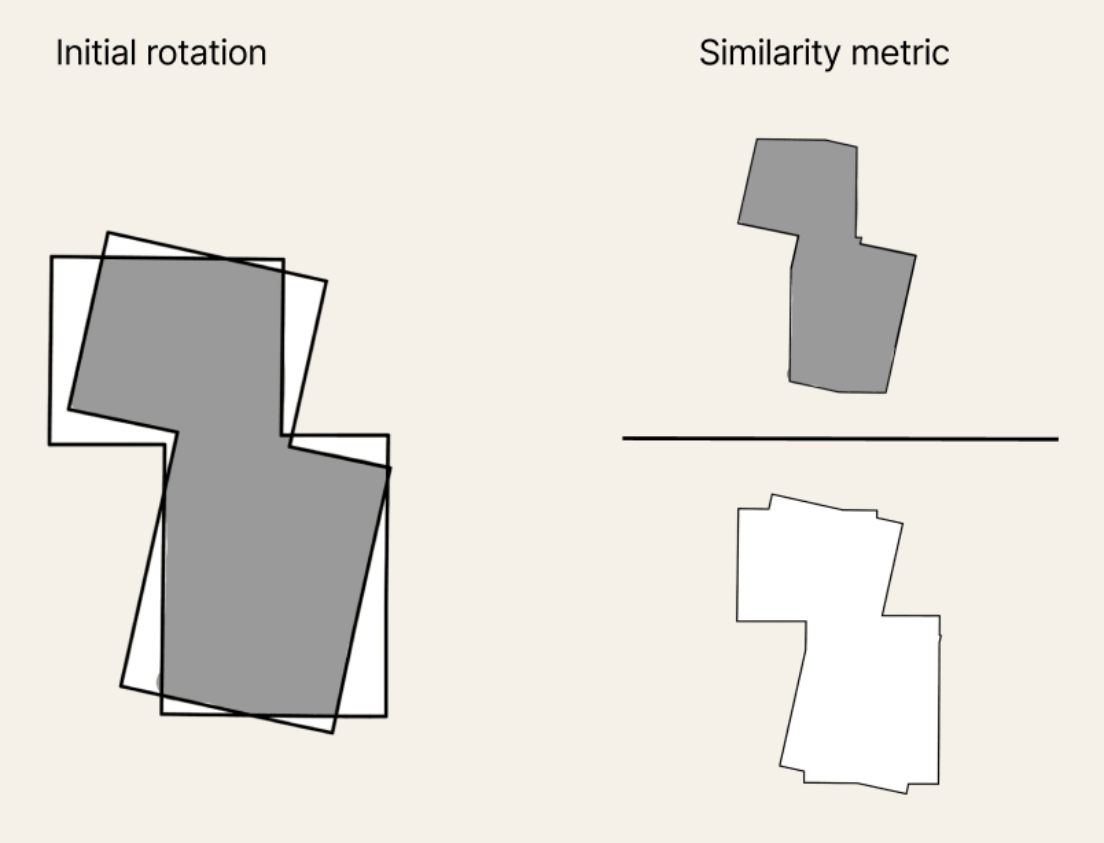

Figure 5: *Example of Intersection over Union similarity ratio used to determine ideal outline rotation to achieve aligned building outlines*

The relative scales are still maintained to display high-resolution images for the user while performing pathfinding on a lower resolution to save computing power. In other words, a scaling factor is used to convert between these resolutions. For pathfinding, the images are converted to a binary format where the A* algorithm is used to navigate through white pixels and

treat all black pixels as impassable walls, using 4-connectivity. The heuristic used for this algorithm uses the Euclidean distance from the goal position, while edge costs between adjacent pixel nodes are constant.

We used OpenStreetMap to plot outdoor building perimeter coordinates, building entrance nodes, and street intersection nodes in order to export the outdoor map as an OSM file. The outdoor longitude and latitude coordinates for each building node are used for the outdoor A* pathfinding heuristic of Euclidean distance from the destination node, as well as for baseline edge weight calculations. Each outdoor edge weight represents the travel time for that edge, which is adjusted by several parameters: transportation type (walking or biking), live weather/precipitation impact, historical crime data and proximity to blue light emergency call stations, construction road blockages, and per-edge traffic flow calculations.

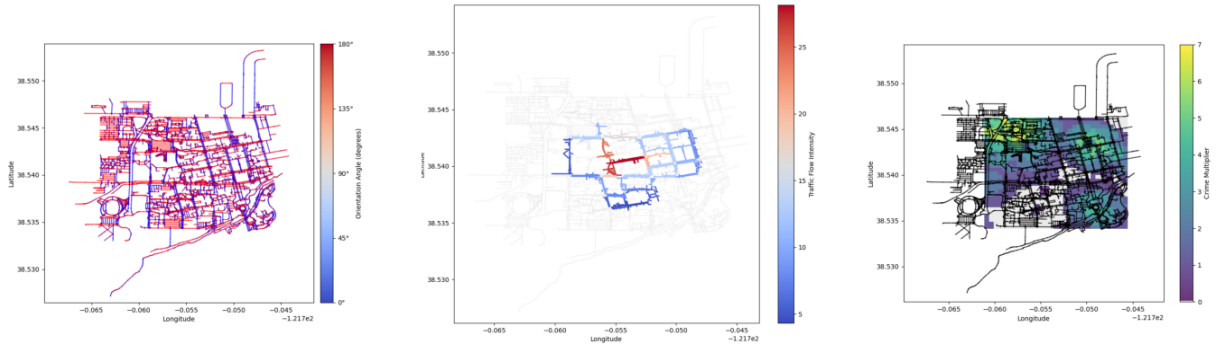

Figure 6: *Plots of different filters used for weight updating,* ***(left)*** *The orientation of the edges used for wind calculation,* ***(middle)*** *the traffic flow stationary distribution,* ***(right)*** *the heatmap with proximity to crime locations*

The impact of some of the above parameters can be applied trivially through simple weight multiplication factors, while others require more complex calculations. Differences in transportation involve multiplying the edge weights by a large factor for walking, compared to a smaller factor for biking (taking into account that the edge weights represent travel time and not distance). Live weather also is somewhat trivial, as we take wind direction into account using dot products from the wind direction and speed and the outdoor road directions in order to model the effect of headwinds and tailwinds on travel speed. We draw from live updated construction closure data from the university api to ensure that edges that are closed will be reflected as having infinite edge weight so that the algorithm is disincentivized to pass through them. We include an optional safety prioritization feature in our website that makes our outdoor A* algorithm prefer paths that are closer in proximity to the blue light emergency call stations, as well as avoid areas that historically are close in proximity to reported crime alerts.

Modeling traffic flow is a difficult task, especially for live updates per edge. In order to save time that would potentially be spent gathering traffic data throughout the week every day, we modeled all outdoor intersections as nodes in a Markov chain, with the edge representing the Campus Corridor containing the Third Street bike counter being an essential part of our traffic flow calculation. We select particular roundabout intersections in campus as priority nodes, such that the surrounding nodes will prioritize sending more traffic towards these priority nodes. This simulates how these particular intersections see significantly more traffic over the course of the day. We encountered an issue when calculating the steady state of the Markov chain at this point, since the cyclical nature of the campus roads cause positive feedback loops resulting in essentially infinite traffic flow. To combat this, we implemented a self-loop for each node (so

10% of students would not leave the starting node for each iteration), representing how some percentage of students finally reach their destination and stop traveling through the road system. Additionally, we included a sink node that connects to all the leaf exit nodes on the campus map (all the edges leading to off-campus locations). This sink node receives 50% of the traffic from each leaf node and does not introduce new traffic into the system, allowing us to bleed off excess traffic that would otherwise create a feedback loop. The sink node represents traffic with a destination that is off campus.

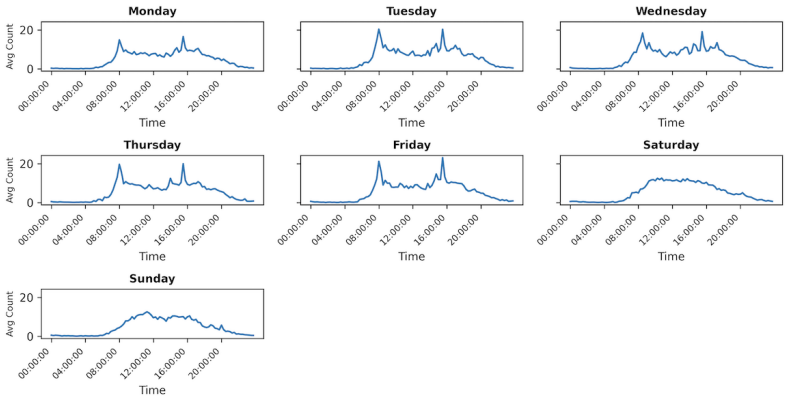

Figure 7: *The average traffic intensity at different times of day from live counter at 3rd street in 2019*

In the end, we end up with calculating all traffic flow probability coefficients (Markov transition matrix values) for each node, and we can calculate the eigenvector of the matrix which represents the long term steady state of the system, and the ratio of people likely to be on each edge at any given time. We use the number of people recorded by the bike counter on Third Street at any given time and day to give us a sample factor to multiply with the eigenvector to estimate how many people are on campus outdoor paths at any given time and day, resulting in traffic flow values for each edge. We have public access to data from 2019 for the bike counter, so the daily traffic distribution is fairly accurate, although that was during the Covid era of lockdown so today's real traffic numbers are most likely inflated compared to our estimated numbers. This should not affect relative traffic flow estimates, however, so we can use the time-based per-edge traffic flow calculation as additional edge weight calculations.

Combining all indoor and outdoor pathfinding algorithms into a global pathfinding app with a Flask website UI, we found that while the pathfinding algorithms worked, the door detection and removal processing section did not reliably make paths through doorways by removing door arcs. This results in many points on floor plans being unreachable to the pathfinding algorithm. Additionally, there were small inaccuracies in manually placing entrance nodes in the process of outdoor mapping. This led to many entrance nodes being unreachable from the outdoor node graph, which caused many paths to fail. In spite of this, we did find particular starting/destination pairs that allowed us to generate proper paths between buildings and to specific rooms. Given more time, we would most likely improve our door detection algorithm, or use convolutional neural networks to more accurately identify and remove doors. Additionally we could take time to replace inaccurate entrance nodes in the outdoor building OSM file.

We've learned that AI has immense potential, grounded in mathematical operations and logic-based processes. For example, computer vision can identify objects like doors and efficiently navigate through hallways and classrooms, connecting rooms to building entrances or

exits. However, there were challenges recognizing doors which required things such as scaling images for consistent resolution, adjusting filters for orientation changes, and rotating filters to accurately detect doors.

Additionally, we discovered that linear algebra plays a key role in analyzing traffic flow and its interaction with nodes and edges. By integrating advanced heuristics into pathfinding, we improved both efficiency and accuracy, resulting in better travel time estimates and safer, optimized routes. This comprehensive approach enhances the practicality and functionality of navigation systems.

Holistically, we can conclude that using computer vision like various algorithms allowed us to create and analyze data that could have taken a long time to gather if not done with CV.

This project addresses a critical need for seamless navigation within the UC Davis campus, providing a robust solution that integrates both indoor and outdoor pathfinding into a unified system. Navify leverages advanced computer vision techniques to process architectural floor plans, accurately detect and refine door locations, and align indoor layouts with outdoor maps using methodologies such as IoU metrics and centroid alignment. These innovations enable efficient room-to-room navigation, addressing a longstanding gap in existing navigation tools.

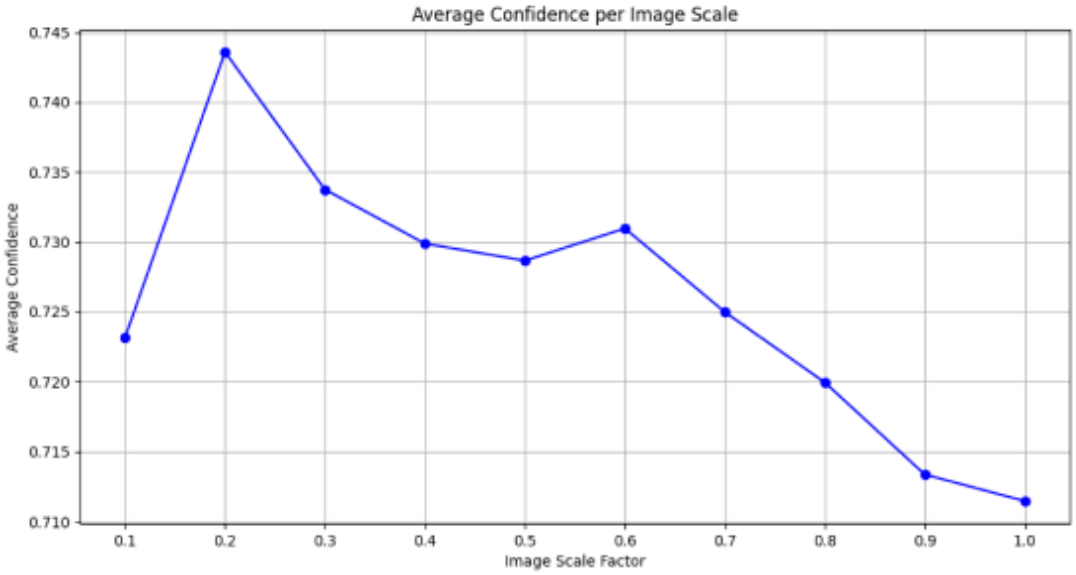

Figure 8: *A graph showing the average confidence value representing how closely the positive convolution matches in a single floor plan fit with the given door filter template, using different image resolution scale factors. This is automatically calculated by the computer vision and selects the image scaling with the highest confidence.*

The outdoor navigation system further enhances Navify's functionality by employing OpenStreetMap data and the A* algorithm, augmented with contextual adjustments for real-time factors such as weather, crime, traffic flow, and construction. Advanced traffic modeling through Markov chains and eigenvector analysis ensures accurate representation of campus mobility patterns, while safety features like proximity to emergency call stations add an essential layer of utility for users. The system's design reflects a meticulous consideration of computational efficiency, utilizing scaled image resolutions and heuristic optimization to balance accuracy with performance.

Navify exemplifies how interdisciplinary approaches, combining computer vision, graph theory, and real-time data integration, can solve complex navigation challenges. This project not

only provides a tailored solution for the UC Davis campus but also establishes a scalable framework applicable to similar environments. By addressing both practical and technical dimensions of campus navigation, this project demonstrates the transformative potential of computational methodologies in enhancing accessibility and user experience.

Liam mapped outdoor entrances for a third of campus, and built the Safety Mode feature for outdoor weight calculation. Shintaro mapped outdoor entrances for a third of campus. Ryan built the image cropping part of the floor plan processing pipeline, updated the database by running the preprocessor code, and helped design the door convolution filter. Tyler built the live weather feature for outdoor weight calculation. Daniel built the construction road closures feature for outdoor weight calculation, worked on local pathfinding code, generated floor plan data to enter into the preprocessor database. Ken gathered three preliminary floor plans to use before access to official floor plans was granted. Albin mapped outdoor entrances for a sixth of campus, wrote the global and local pathfinding algorithms, designed the traffic flow algorithm, created the website UI, and generated figures for documentation and presentation. Artem mapped outdoor entrances for a sixth of campus, built initial computer vision approaches for indoor wall detection, worked on initial secure website UI designs, worked on node graph structure for global pathfinding, parsed OpenStreetMap data for database nodes, and wrote initial image recognition for bike flow detection.