TORCHBEARER:

A MULTI-PIPELINE APPROACH TO LANDMARK-BASED NAVIGATION

by

Fredric Muller Vollmer

A thesis submitted in partial fulfillment
of the requirements for the degree

of

Master of Science

in

Computer Science

MONTANA STATE UNIVERSITY
Bozeman, Montana

July 2018

DEDICATION

This thesis has truly been one of the greatest challenges, if not the greatest challenge, I have faced so far. But to say it was a result solely of my own labor would be immensely far from the truth. The end result is due to the support, understanding and love of so many people in my life, without whom I would never be writing this today.

To my wife, Annie, who has stood by my side through thick and thin, who has allowed me to devote so much of the time that belongs to us to this work. To her I promise to make up the time, and then some.

To my parents, Jan and Dick, whose relentless help with whatever passion I might be pursuing has led to opportunities I am incredibly fortunate to have had. While I didn't always realize it, their passion for science, ingenuity and worldly understanding was always driving me towards this point. They are truly the only role models, the only inspiration I will ever need.

To all of my family: Chris and Lori, Nana and Opa; Gwen, Jim, Carey and Mark; Jo and Michael, Dorian, Gina and Carl. Thank you for being a part of my life.

ACKNOWLEDGEMENTS

First, a huge debt of gratitude is owed to my advisor and committee chair, Dr. Mike Wittie. Without his guidance on technical issue and writing, this project could not have been completed. I must also thank Dr. Laura Stanley, whose expertise in human factors and driving research shaped the goals and evaluation methodologies of this work.

The Torchbearer Mobile App, without which this project could never have been put into the hands of drivers, was given a great deal of time and code by Brendan Smith.

The dataset of Google Streetview images was meticulously annotated to include bounding boxes and labels by Cole Homan.

VITA

Fredric Muller Vollmer was born in Deming, Washington on August 25th, 1991, to Jan and Henry Vollmer. He attended Mount Baker Senior High School in Deming, Washington. In 2015, he received a Bachelor of Science degree in Economics with a Statistics minor from Montana State University in Bozeman, Montana.

TABLE OF CONTENTS

TABLE OF CONTENTS – CONTINUED

TABLE OF CONTENTS – CONTINUED

LIST OF TABLES

LIST OF FIGURES

LIST OF FIGURES – CONTINUED

Figure Page

LIST OF FIGURES – CONTINUED

Figure Page

LIST OF FIGURES – CONTINUED

Figure Page

LIST OF ALGORITHMS

ABSTRACT

The task of navigation adds cognitive distraction to the already demanding task of driving. Most popular navigation aids provide verbal directions based solely on distances and street names, but the inclusion of landmark descriptions in these instructions can improve navigation performance, decrease unsafe driving behaviors and reduce cognitive load. Current approaches to selecting landmarks and building landmark-based instructions rely on a single source of data, thereby limiting the set of potential landmarks, or use a single factor in choosing the best landmark, failing to account for all characteristics that make a landmark suitable for navigation. We develop a multi-pipeline system that leverages both human (crowd-sourced) input and machine-based approaches to find, describe and choose the best landmark. Additionally, we develop a mobile application for the delivery of navigation instructions based on landmarks. We evaluate the cost and performance differences between these pipelines, as well as study the effect of landmark navigation prompts on cognitive load, safe driving behavior and driver satisfaction via an in situ experiment.

1

INTRODUCTION

In 2016, there were nearly 35,000 deaths resulting from motor vehicle crashes [18] in the United States. Yet despite the danger of driving, automobile transportation remains an integral part of people's daily lives: in that same year, Americans drove a collective 3.17 trillion miles [18].

A large majority of automobile fatalities are the consequence of driving under the influence, adverse weather conditions, or speeding. However, in 2016, 16 percent of all vehicle crashes were the result of driver distraction [19]. Tasks, which a driver must perform in conjunction with operating a vehicle (secondary tasks), impose cognitive load, which in turn leads to the driver being distracted from vehicle operation. Distraction leads to dangerous driving behavior, such as hard braking, manifested as sharp changes in longitudinal acceleration, or sudden steering corrections, resulting in sharp lateral acceleration [23].

Some secondary tasks, such as texting or applying makeup, are best refrained from altogether. However, other secondary tasks are requisite to the primary task of driving from origin to destination. The use of electronic, turn-by-turn navigation aids, such as Google Maps, is one such task: while it has been shown to produce a significant cognitive load [43], it is a valuable tool, which allows drivers to efficiently reach a destination. Indeed, in-car navigation is a common task; 67-percent of smart phone users indicate that they use their device for this purpose [58]. Be it utilizing an alternate route to work to avoid construction, trying to find a new restaurant, or getting from the airport to a hotel in a never-before-visited city, the real-time auditory directions offered by navigation aids have done away with the need for a driver to

take her eyes off the road to glance at a paper map or digital map display [61]. By reducing the cognitive load induced by navigation aids, drivers will be enabled to exhibit safer vehicle operation characteristics while still enjoying the benefits of turn-by-turn navigation.

Instructions delivered by the most popular navigation aids generally consist of street names and numeric distances, requiring the driver to perform a visual search for small street name signs and to estimate driven distances. The addition of landmark descriptions could lessen this cognitive load, for example "turn right at the Dairy Queen" instead of "turn right in 600 feet". A salient landmark, here "Dairy Queen", provides more obvious information than the numeric distance. Even if a person is driving in a city previously unknown to them, the distinctive appearance of a Dairy Queen can distinctly identify a turn. Previous research has suggested that if electronic navigation aids could include relevant landmarks in their instructions, the cognitive load of the driver could be decreased [8].

Including landmarks in navigation instructions requires several computational frameworks. First, a method for locating candidate landmarks, or physical features located near a maneuver point. Second, a means to lexically describe a landmark, in a detailed manner, which allows the driver to easily recognize it. Lastly, an approach for determining the best landmark out of a set of candidates—the landmark which is most recognizable to the driver.

Current approaches to automated landmark-based navigation are limited, many being restricted to pedestrian scenarios, others relying on pre-compiled sets of landmarks and still others using only point-of-interest datasets for selection, without incorporating visual analysis of maneuver points. We present Torchbearer, a system which leverages multiple approaches, or pipelines, to locate candidate landmarks, provide lexical descriptions of the same and determine which landmark is best-suited

to be included as part of a verbal navigation instruction delivered to a driver at a particular maneuver point. Given the coordinates of an origin and destination, Torchbearer leverages standard pathfinding algorithms to find the least-cost (fastest) route. For each point, where the end user will need to perform a driving maneuver, such as a turn or merge, Torchbearer determines the landmark best suited for helping the end user locate that point. Torchbearer then builds a verbal instruction, consisting of the street name, distance, description of maneuver to be executed, and description of the landmark, delivered to the driver via an audio-based mobile application. The system extends existing navigation technology to offer landmark-based navigation assistance.

Torchbearer's novelty comes from its hybrid, pipeline-based approach: we use four distinct pipelines to find landmarks and select the most suitable for a given maneuver point. First, a fully human-based approach, which uses crowdsourcing to find landmarks near a location, select that which is best suited for navigation, and generate a description of the landmark. Second, a human in the loop approach, which uses a state-of-the-art saliency detection algorithm to find the most obvious, easiest-to-see landmark, but leverages crowdsourcing to generate a description of that landmark. Third, a pipeline that uses a database of local businesses and points of interest, as well as a deep learning-based object detection algorithm, to find landmarks, and utilizes crowdsourcing to select the optimal one. And lastly, a fully-automated pipeline which uses the saliency-detection algorithm for finding the most visible, easiest to spot landmark and the point-of-interest data source, and object detection algorithm to describe that landmark.

Torchbearer differs from existing solutions in three principal aspects. First, its pipeline-based approach uses and analyzes several landmark selection methodologies interchangeably. Second, it incorporates multiple landmark features into its selection

process–visual, data-based and human recognition; this allows Torchbearer to consider a wider range of landmark types than previous systems. Additionally, Torchbearer relies only on publicly available data sources which have very wide geographic coverage across the United States; some existing work relies on expensive data sources such as laser range mapping.

The Torchbearer system is designed to reduce drivers' cognitive load, reduce erratic driving behavior, and lessen perceived workload. We evaluate the system using a standard Peripheral Detection Task (PDT) to measure cognitive load and the NASA Task Load Index survey to measure perceived workload. Additionally, we monitor extreme gravitational force occurrences, as an indicator of driving behavior associated with distraction. We also survey subjects on their perception of landmark goodness and ease of navigation. To provide insight into the costs and benefits of particular pipelines, we also provide an analysis of pipeline performance, examining cost, runtime and result similarity.

Torchbearer presents a completely automated solution to selecting and describing landmarks for use in navigation instructions, using multiple pipelines of varying approaches capable of selecting a wide range of landmark types ranging from road infrastructure, to buildings, to businesses. While we fail to find significant reductions in cognitive load, erratic driving behavior or perceived cognitive load in our small-scale field study, Torchbearer can serve as a robust platform off of which to incorporate other algorithmic or human-based landmark selection ideologies.

BACKGROUND

Distraction and Cognitive Load In the Context of Driving

While driving is a dangerous endeavour due to a wide array of factors, including environmental, human and vehicle equipment related circumstances, a significant contributor is driver distraction, which accounts for 16 percent of vehicle accidents [19]. Distraction, in the context of driving, is the diversion of attention away from the task of safely and efficiently operating the vehicle, onto some secondary task [49]. If we consider the driving task to consist of applying lateral (right and left steering) and longitudinal (braking and forward acceleration), then distraction is dangerous primarily because it inhibits the driver's ability to quickly and accurately apply these actions in response to changing situations in the environment [45].

Sources of Distraction

Broadly, a source of distraction is classified as in-vehicle or out-of-vehicle. Out-of-vehicle distractions include visually abnormal occurrences such as police actions, accidents, or billboards [14]. In-vehicle distractions can be further refined as technology-based or non-technologically based. Talking with a passenger, applying makeup, eating, or smoking all pose a potential non-technological distraction. Technological distractions are receiving rapidly increasing academic attention due to the rising penetration of in-vehicle information systems (IVIS) and smartphones [5]. IVIS pose a significant issue in regards to distraction, as they often require the driver to look at a screen, or interact with the system in some way, creating both a visual and cognitive distraction [7]. Cognitive distraction results in unsafe driving behavior, including steering errors (lane departures), increased variability in accelerator position, and the sharp breaking due to a shorter window in which to respond to

a change in the environment [31]. Mobile devices, such as smartphones, lead to driver distraction via the introduction of a physical (holding and tapping/swiping) visual and cognitive load upon the driver One study estimates an increase in reaction time to a pedestrian crossing the path of travel of 204 percent when the driver attempts to text and drive. [11].

Navigation systems, implemented via IVIS, or a mobile device, represent a unique form of distraction in that the interaction with the system (supplying a destination, looking at a map, listening to instructions) presents one secondary task, while the execution of the system's instructions (scanning for upcoming turns) presents another. Together these tasks can cause the driver to disengage from the environment [33]. This disengagement leads to an increase in reaction time while using a navigation system, which is more pronounced for navigation apps that have a visual interface than those which are entirely audio-based [25]. The task of entering an address using a touch screen poses a particular problem, with one study finding a increase in the standard deviation of lateral vehicle position of 60 percent. [60].

Landmarks in Navigation

Mainstream navigation aids tend to heavily utilize distance-to-street-name instructions, which require the driver to conceptualize distances and perform a visual search for small road signs. [8]. Humans, on the other hand, tend to provide navigation instructions using landmarks [63]. One study found that instructions provided by a passenger, which were primarily landmark-based, resulted in fewer navigation errors, shorter trip duration, lower perceived workload and a higher quality of driving as rated by an expert, leading to the conclusion that the inclusion of landmarks in automated navigation instructions could be beneficial [9].

Lovelace [34] examines the components of good navigation instructions for both

familiar and unfamiliar routes. They found that in general more information provided in an instruction resulted in higher perceived quality. Additionally, they found that the inclusion of landmarks, both at maneuver points and intermittently along the route, significantly increased perceived route quality.

Golledge [21] asserted that landmarks can aid in the navigation task because they serve as both global reference point, allowing the driver to mentally organize the space he is traveling through, and also as a sort of marker for decisions (maneuver) points. Indeed, the substitution of landmark-based instructions for distance-based instructions has been shown to decrease navigation error count and improve driver confidence [37]. Interestingly, while the quality of landmarks did have a significant effect on these measures, both good and poor landmarks were significantly better than distances alone [37]. Completing a study in a real traffic environment, another work found that the use of landmarks (as opposed to distance) resulted in fewer glances at the navigation aid's display and better driving performance as measured by lane departure count and improper turn signal use. [36].

Landmark Saliency: What Makes A Good Landmark

Saliency is to the property of being particularly noticeable, prominent or important [54]. A landmark is a physical feature that serves as a point of reference within the environment; it is distinctive from its surroundings to such a degree that it is easily recognizable and represents an exact point in space. Because of this importance of uniqueness, the saliency of a landmark is not a function of the attributes of an individual landmark but rather how distinctive those features are relative to nearby objects. Indeed, being a good, salient landmark is a *relative* property [48].

Landmarks can be broadly classified as *global*, visible form the entire route and relevant throughout, or *local*, important to a specific maneuver point (turn). Driving

directions do not usually include global landmarks [59]. Local landmarks are best for navigation, and are most useful to the driver near decision (maneuver) points [32].

Saliency is represented by a tripartite typology, where three distinct dimensions, *visual*, *semantic* and structural, compose the overall saliency of a landmark [59].

Visual Saliency

Visually saliency is analogous with visual attractiveness. In general, visual saliency is based on behavior observed in most vertebrates, in which they alter their gaze so as to focus more attention on relevant details in a scene while ignoring unimportant areas [24]. A region within in the scene, or a specific object in the space, is salient if it receives a significant portion of attention. In the context of navigation, a landmark is visually salient if it has sharp contrast with its surroundings and is prominent (easily in view) from the driver's location [59].

Reubel and Winter [48] show that the visual saliency of a landmark is calculated by comparing several physical properties. (Of course, the calculated value for a landmark has no meaning until compared to that of a nearby landmark–saliency is relative.) The *facade area* represents the total physical area that is visible to the driver. (Essentially, the bigger the landmark, the better.) The oddity of the shape also plays a role; the larger the deviation between the shape of the landmark's silhouette and a rectangle, the more visually attractive it is. Color is the final factor, specifically how different the landmark's color is from the surroundings.

Semantic Saliency

Sorrows and Hirtle [59] define what they coin a *cognitive landmark*, a landmark whose meaning, history or cultural importance makes it prominent in the environment. Such a landmark has an atypical level of importance relative to its surroundings, possibly in spite of a typical level of visual attraction. The house of

a university president, for example, likely has a high degree of semantic saliency due to its significance in the community, even if visually it may be quite similar to surrounding homes.

Reubel and Winter [59] refine the notion of a cognitive landmark to obtain a more formalized definition of semantic saliency. Specifically, they include a Boolean value for whether, or not, a landmark has historical or cultural significance to the area. Additionally, they include a Boolean value for whether the landmark his discernible commercial semantics, that is, is it a business of a type people are familiar with (such as coffee houses or grocery stores.)

Duckham and Winter [46] expand this definition by suggesting that the semantic saliency of a landmark is also a function of its ubiquity. The ubiquity of a landmark is important, they argue, cultural significance is less meaningful to people unfamiliar with a given area, as what is culturally significant to the area may be unknown to them. Accounting for ubiquity in the semantic saliency measure accounts for the fact that the more instances of a landmark there are, the more widespread its significance is. As an example, consider a 50-year old local burger joint situated near a McDonald's that opened a year ago: while the cultural and historical significance is much higher for the burger joint at the local level, the ubiquity of the McDonald's belies its much higher significance on the global level.

Geosocial data streams, such as FourSquare, Facebook and Google Places also have the potential to provide semantic saliency information. Quesnot and Roche [47] argue that geosocial data, which encodes information about who visits a landmark, can offer valuable insight into the importance of that landmark. If a large number of people frequently visit a landmark, it is likely to be more important than one which receives few visitors. It essentially acts as a proxy for cultural significance, with the enhancement that it provides a quantitative, real-time measure.

Uniqueness is also an important component of semantic saliency [10]. Just as a green house is visually salient among a group of red houses, a library is semantically salient among a group of restaurants. The uniqueness of a landmark's intended purpose within its surroundings is an important consideration [10].

Structural Saliency

The final tenant of landmark saliency is structural saliency, which broadly refers to the pertinence of a landmark in the context of its location in the physical space of its surroundings [59]. At a more applied level, a landmark is structurally salient if its location (relative to the route) is easy to conceptualize cognitively and linguistically [29].

Klippel and Winter [29] developed the first formal syntax for structural saliency. They provide a hierarchy of structural saliency in terms of a landmark's position in relation to the intersection where a turn is to occur. While the hierarchy is extremely thorough, the key takeaway is that it is best for a landmark to be located on the corner of an intersection where a turn is to occur. The location of such a landmark is easy to describe linguistically: "turn left after the McDonald's" or "turn left before the McDonald's", depending on whether the landmark is on the near or far side of the intersection. If a landmark is located significantly before, or after, the entire intersection, then it becomes difficult to summarize into an instruction, and potentially even more difficult for a driver to conceptualize. Instructions such as "at the intersection after where the McDonald's is" are more complex both linguistically and conceptually. Roser [52] offers empirical evidence, based on an ergonomic study in a virtual environment, which supports this hierarchy.

Prior Art: Automated Landmark Detection

Multiple approaches have been implemented in attempts to automatically select landmarks for navigation, spanning a wide range of goals, working definitions of landmark saliency and data sources. Much work has also been done in the context of pedestrian navigation, to a greater extent than has been done for vehicle-based navigation.

Hile et al [28] leverage a dataset of geotagged images to generate landmarks for pedestrian walking instructions. For a given path a pedestrian will walk, a database of points of interest is used to select and annotate an image. The photograph, along with the description and navigation instruction, are displayed on the user's device. Selection criteria is based on the proximity of a landmark to the user's path of travel, as well as how closely the angle of the photograph matches the heading the user is traveling.

Beharee and Steed [6] also used geotagged images to provide navigation aid to pedestrians, but selected a series of landmark photos to show along each leg of the route. Proximity to the route was used as the selection criteria. Landmarks were not given lexical descriptions. A between-subjects study revealed that in areas not familiar to the subject, the addition of photographs to the navigation application allowed subjects to arrive at target destinations in less time than when with textual directions alone.

In another application targeted at pedestrians, Wenig et al [62] developed a system for finding global landmarks that can be used to orient the user. For example, a user looking for a destination in Paris might be given instructions in terms of the relative location of the Eiffel Tower. Global landmarks are used based on the authors' argument that local landmarks are difficult to select accurately. Candidate

landmarks for a given region are predefined; the best landmark is chosen based on level of visibility throughout the entire route to be traveled. The visibility of a landmark at a given point is determined in a binary fashion using Google Street View images and a deep neural network. The authors show that this approach leads to greater confidence and more accurate cognitive map building among subjects.

Elias and Brenner [15] use visual saliency to select landmarks for driving-based navigation instructions. Using a Geographic Information System (GIS) dataset, the authors mine candidate landmarks (always buildings), where a landmark is a candidate if it has some unique or distinctive feature compared to its surroundings. Features examined include building use or purpose, land use type and building extremities, such as outbuildings or carports. The best landmark is chosen based on how visible it is to the driver as she approaches; this is determined using a three-dimensional aerial laser scanning model of the area and modeling the the area of a landmark which is within the drivers cone of sight. The system does not offer detailed landmark descriptions, and was not evaluated by a human-based experiment. Torchbearer provides meaningful landmark descriptions via human and algorithmic input, and we perform a small-scale but thorough field study with human subjects.

Electronic Navigation Aids

There currently exist a number of commercial, as well as academic or open-source, electronic navigation platforms. Most provide only distance-based instructions, but some prototypes do exist which incorporate some form of landmark descriptions (especially among systems designed for pedestrian use.)

Google Maps

Google Maps is a mobile application for iOS and Android devices which is capable of providing turn-by-turn driving directions between an origin and destination point. Users provide the destination via voice or keyboard, and can enter addresses, coordinates or points-of-interest. The app provides primarily distance-based instructions, complete with street names. Some instructions will use road topology to describe the maneuver point, such as "turn left at the end of the road."

Routing is based on finding the shortest travel time, and includes traffic and construction delays in its optimization.

As of mid-2018, Google Maps has, reportedly, begun to include landmarks into its spoken directions [17]. It is not yet a documented feature, and has been enabled on only a small number of devices. It remains unclear what types of landmarks it incorporates and what methods it uses for selection [17].

Waze

Waze, owned by Google since 2013 [55], provides turn-by-turn navigation instructions in a similar manner to Google. Waze is novel because, along with a base of OpenStreetMap data, Waze considers travel time, police traps, construction delays and other data from its users, which it incorporates into its map and routing decisions. Spoken instructions consist of distances and street names.

14

ARCHITECTURE

Architectural Overview

The goal of the Torchbearer system is simple: given the latitude, longitude and approach bearing of a maneuver point, render a string describing the most salient landmark at that location. The problem which Torchbearer solves the problem expressed by

$$f(lat, long, bearing) \longrightarrow description \tag{3.1}$$

where f is some method of landmark selection and description. The Torchbearer system provides multiple implementations of f, which we call pipelines. Each pipeline consists of an ordered set of tasks, T. Each task $t_i \in T$ accepts some input from the previous task t_{i-1} and returns some output to be input to the next task t_{i+1}—the obvious exceptions being the first task, which takes a tuple $(lat, long, bearing)$ as input, and the last task, which outputs the selected landmark—the final result of the pipeline. Each task progressively solves a small part of the landmark selection problem, such that at the end of the pipeline Torchbearer has computed a lexical description of the most suitable landmark. It is natural to consider a pipeline as a composition of functions:

$$P = t_{n-1} \ldots t_1(t_0(lat, long, bearing)) \longrightarrow description \tag{3.2}$$

where $n =\mid T \mid$. As an implementation detail, it is important to note that tasks can be performed in parallel if they all take the same input form the previous task. Examples of such parallelization are shown in the descriptions of each specific pipeline.

The Torchbearer system receives input from a client mobile application in the

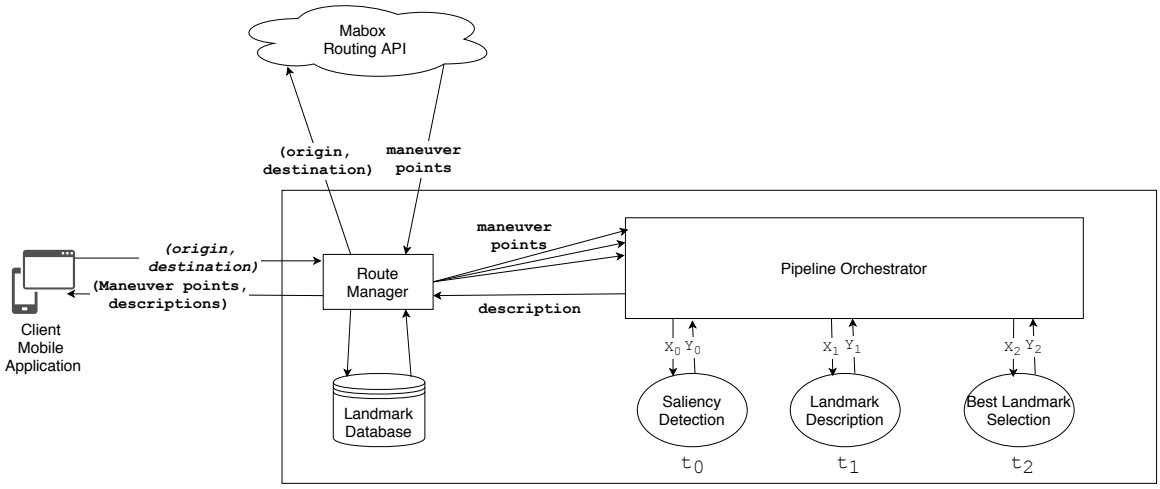


Figure 3.1: A high-level view of the Torchbearer system.

form of a $(origin, destination)$ tuple. After gathering a set of maneuver points from the Mapbox Routing API, the Orchestrator receives a list of $(latitude, longitude, bearing)$ tuples corresponding to each maneuver point for which a landmark description should be computed. The Orchestrator manages the execution of each Task in the pipeline, and returns the final selected landmark to be saved in a database, where it can be queried by the mobile client. We discuss each component of the system in further detail below.

Orchestration

In order to implement the function composition approach discussed above, each pipeline requires a system to progress a maneuver point through each task in the pipeline. We call such a system the pipelines *Orchestrator*. The Orchestrator is the manifestation of the pipeline, in the sense that it is solely responsible for the intake of new maneuver points to be processed and overseeing the ordered execution of each task for that maneuver point.

The Orchestrator is centralized service which acts as a specialized message

broker. For each task t in the pipeline, the Orchestrator maintains a FIFO queue q_t of maneuver points for processing through that task. Queue items are tulpes containing the unique identifier of the maneuver point, a token representing the specific task instance and the input to the given task, I_t.

A task *worker*, described in the next section, polls the Orchestrator for a task in need of completion; if such a task is available the Orchestrator pops it the from queue and returned to the worker. (We discuss polling in greater detail in the following section.) When the worker has the completed the task, it sends the results back to the Orchestrator, which then adds a new item to the queue corresponding to the next task t_+, including the output of t (which is the input to t_+). If the execution of the task results in an error, the worker sends the details of the error to the Orchestrator, which then halts further execution of the pipeline for that maneuver point.

The Orchestrator supports parallel execution of tasks, by placing a maneuver point into two queues simultaneously, and pausing progression of the pipeline until **both** tasks complete. The input X to the next task $t + 1$ is then the union of the outputs of the n parallel tasks:

$$X_{t+1} = Y_0 \cup Y_1 \cdots \cup Y_n \tag{3.3}$$

An Orchestrator also maintains a database of pipeline state, including, for each maneuver point, the output of each task, or error details if one occurred. The execution for a specific point can be traced or monitored throughout the pipeline by querying this database.

Task Implementation

A task receives a tuple X as its input and yields a tuple Y as its output. A tasks output must be inclusive of its inputs, that is, $X \subset Y$. Let $p_+ = Y \setminus X$, then p_+ is the context contribution of a given task—the information which that state has added to the pipeline's overall knowledge, or state. For example, a task devoted to describing landmarks might receive a list of candidate landmarks as input, and output both that list and a list of computed descriptions for each landmark.

It is important to note that each task has a binding contract in terms of its input and output, based on where it sits in the pipeline. For example, task t_1 must accept input corresponding to the output of task t_0, and must provide output corresponding to the input to task t_2. This contract presents a not insignificant constraint in regards to rearranging tasks within a pipeline: even if $t_1(t_2) == t_2(t_1)$ (that is, the order in which the pair of tasks is executed is not important) the two tasks could not change positions in the ordered set of tasks for the pipeline unless their inputs and outputs were identical.

A task is solved by a worker. A worker is an independent, isolated computational entity which is responsible for the execution of a specific task. There can be any number of (identical) workers for a task active at one time, essentially functioning as a cluster, allowing for multiple instances of the given task to be executed simultaneously. (Each instance of a task will only be run on one worker.) For example, multiple worker instances for the Landmark Description task could run at the same time, allowing for parallel execution.

Workers are stateless: in order to complete its task, a worker relies only on the input it receives from the Orchestrator, without regard to previous tasks completed by other workers. To describe a landmark, for example, the Landmark

Description worker does not require any information about other landmarks within the Torchbearer system. Workers have no awareness of the context in which they do their jobs, in the sense that workers can handle tasks from multiple pipelines and do not care about the order in which they are asked to handle tasks.

A worker performs three essential functions: first, to find new tasks needing execution, it polls the pipeline Orchestrator. Second, it carries out the computational operations needed to complete the task, utilizing inputs from the Orchestrator. Lastly, it returns outputs to the Orchestrator upon successful completion of the task, or alerts the Orchestrator of a failure.

Polling for Tasks

The first function of a worker is to poll pipeline Orchestrators for tasks in need of completion. A worker will ask only for the task or tasks it is capable of executing. It is important to note that the Orchestrator utilizes a pull mechanism for task assignment, as opposed to a push mechanism: rather than Orchestrators routing tasks to specific Workers, each Worker is responsible for finding its own work by asking Orchestrators for available jobs. While an Orchestrator serves as a manager for tasks, having state related to the precise status of all pending tasks in its pipeline, it does not serve as a manager for Workers. Indeed, no component of the Torchbearer system maintains state related to the Worker pool. Workers can be stood up or can fail without disruption.

The polling mechanism runs within its own thread and runs in a continuous loop. Each iteration of the loop consists of the following: to initiate the polling sequence, a Worker sends an HTTP GET request to the Orchestrator. If the Orchestrator has any tasks which the worker is capable of completing, it immediately responds with a payload containing a list of tuples, where each tuple contains inputs and a unique

identifier corresponding to each specific task. If no such task is currently available, the Orchestrator holds the request open for up to 60 seconds, waiting for a task to become available. As soon as a task becomes available the payload is sent and the request ended. If no task becomes available during the 60 second window, the Orchestrator terminates request with an empty response.

When a worker receives a response from the polling request, it first checks to see if the response contains a payload (indicating that at least one task was received.) If the response contains a payload, it spawns a thread for each received task to process (complete) the given task, passing in the input and unique identifier received from the payload. If the response does not contain a payload, the current iteration of the loop completes, and the process repeats with a new polling request immediately being invoked.

Task Execution

The execution step is responsible for solving or completing the workers task. The majority of this steps procedure depends on the task, and is discussed for each task in depth later on. It is important at this juncture to understand only that the execution step for a given task runs asynchronously in its own thread, spawned by the polling thread and being provided with both the inputs to the task and unique identifier of the task. The runnable routine of this thread consists of a program which will accept the tasks inputs and yield the tasks outputs–that is, it solves or completes the task.

Submitting Results

If the task completes successfully, the task execution thread sends an HTTP POST request to the Orchestrator, consisting of the task token as well as the yielded output. If any error occurred during execution, the worker sends an HTTP POST

request to the Orchestrator containing the task token, the error message and any additional data about the error, such as a stack trace.

Worker Deployment and Operations

Torchbearer is a microservice-based system; workers have complete flexibility in implementation. Besides conforming to the input/output contract specific for the given task, Torchbearer is entirely agnostic to how a worker completes its task and where (on what machine) it does so. This flexibility provides incredible power in terms of optimizing compute resources and designing solutions which are best-suited to a particular task. Workers can implement solutions, in any language, run on any operating system and run on hardware suited to their particular demands. For example, we implement a task for looking up a location in GIS database in Scala, and, due to its lightweight computational demands, it runs on a single-core machine with 256 MB of RAM. On the other hand, we implement a deep neural network-based computer vision task in Python, and run it on a multi-core machine with a 3072-core GPU and 16 GB of RAM.

In order to facilitate this level of microservice-based independence, Torchbearer we implement Torchbearer workers as Docker containers and run them on Amazon's Elastic Container Service (ECS). A Docker container enables a worker to define the exact specifications for a virtual machine, and ECS runs this container on an appropriate hardware node. The container is a self-contained bundle consisting of the virtual machine definition and the binary for the worker program (the code which actually handles the task.)

By containerizing workers and running them on a container management service such as ECS we also gain the ability to horizontally and vertically scale compute resources at the task/worker level. We can run multiple instances of each container

simultaneously, and we can adjust the number of instances in real time according to changing demands for a service—this provides us with horizontal scalability. For example, a task utilized by every pipeline will, in the steady-state, require more instances than one which is used by only a single pipeline. Since the demand for Torchbearers services is in constant flux (in general, a higher number of active users corresponds to a higher load requirement) the ability to add and remove instances of a worker as the demand for that task fluctuates is highly important to the cost-effectiveness and efficiency of the system.

Route Manager

The Route Manager (RM) service is the contact point between the Torchbearer backend and users (via the client mobile application, discussed below.) While the RM service is not directly responsible for solving the landmark description problem, RM serves as the gateway for client applications wishing to use the Torchbearer system. RM exposes a public-facing Application Programming Interface (API) consisting of the following endpoints:

POST /route

A client (generally an end user's mobile phone) calls this endpoint both to determine the shortest-path route to a destination as well as to initiate landmark processing in the Torchbearer system. This endpoint accepts an origin tuple consisting of $(latitude, longitude)$ (generally, the user's current location) as well as a destination tuple of the same form, and returns the shortest-path route in the form a list of maneuver points. A maneuver point is a tuple consisting of the latitude, longitude, and bearing of the maneuver, the unique ID of the maneuver point within the Torchbearer system and and an instruction to be spoken to the user as they near

that maneuver point. If immediately available, the tuple also includes the landmark description computed by the specified pipeline; we discuss this in more detail shortly.

When RM receives this type of request, it must first determine the shortest-path route between the origin and destination points. We use Mapbox, a third-party service which offers a public street routing API. While there are no special requirements for the routing algorithm Torchbearer uses, Mapbox was chosen for its unique trait of including approach bearings for each maneuver point. Another routing service, or a custom solution, could be integrated into Torchbearer in place of Mapbox, so long as it accepts origin and destination coordinates as input and returns a list of maneuver points, each consisting of latitude, longitude, approach bearing and maneuver type (right turn, left turn, merge, etc.)

Once the route has been determined, RM queries the Torchbearer database for each maneuver point. If the maneuver point exists in the Torchbearer system, and has already been processed by the specified pipeline, the computed landmark description is returned in the response. If the maneuver point does not exist, RM inserts a record for it into the database, and initiates processing by sending an HTTP POST request to the Orchestrator of the desired pipeline. The list of maneuver points are then returned to the client.

It is important to note that while this endpoint will immediately return all maneuver points for a route, some maneuver points will be in a processing state (by the given pipeline). If a point is still processing, RM returns it without a landmark description, and the client will need to ask RM for an updated description at later time using the `GET /maneuverpoint/landmark` endpoint.

GET /maneuverpoint/landmark

This endpoint accepts a maneuver points unique identifier and a pipeline as input and returns a description of the landmark computed for that maneuver point using that pipeline, if one is available. This endpoint is used for checking if Torchbearer has completed processing a maneuver point after the initial route has been returned. For example, if a client navigation application did not yet know the landmark description for an upcoming maneuver point, it might query this endpoint immediately prior to speaking a navigation instruction, to see if a landmark description was now available.

User Interface

Users interact with Torchbearer via a native mobile application, developed for both iOS and Android devices. The primary screen of the application is shown in Figure 3.2. The application has two principal functionalities: first, the ability to search for a destination and submit the route to the Torchbearer system, second, the delivery of spoken turn-by-turn navigation instructions containing the landmark description-based instructions created by one of Torchbearers pipelines.

Usage of the application during a typical navigation consists of the following flow: first, the user selects the pipeline they wish to use for computing instructions. By default, the application selects the machine-machine pipeline, which we discuss in a subsequent section. Second, the user enters a desired destination using the keyboard or text-to-speech capability of her device. The destination can be an address, business, point-of-interest, or general area, such as a city. Using geocoding services provided by Google, the application determines the most relevant geographical coordinates for the destination description entered by the user. The geocoding service takes into account the provided description and the user's current location in determining the most likely

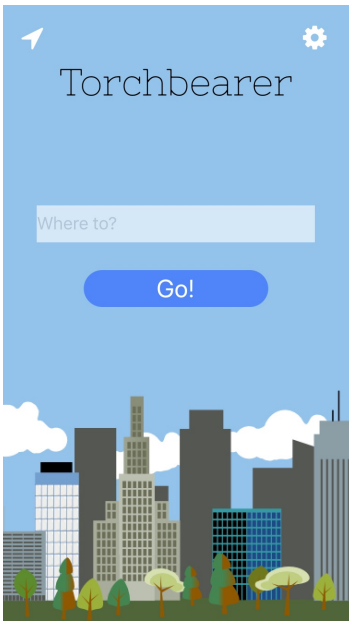


Figure 3.2: The Torchbearer mobile application for spoken navigation instructions.

destination. The app displays the address derived by the geocoding service to the user, and asks her to confirm its correctness.

After confirmation from the user, the application submits an HTTP POST request to Torchbearer's Route Manager service, which returns a list of instruction tuples for the route. Each tuple consists of the coordinates of the maneuver point as well as the instruction string for the app to "speak" upon approaching the turn. While this response is returned immediately, the processing of the route (to determine landmark descriptions) is asynchronous. If a maneuver point has already been processed by the specified pipeline, its full instruction can be immediately returned in the response, but for points not yet analyzed by the pipeline, only the street name can be returned. As such, the initial route received by the application may not contain complete instructions, that is, instructions inclusive of landmark descriptions, for all maneuver points.

At this point, the application delivers a spoken instruction to the user for the

first maneuver point in the route, and the user begins driving. This begins the *check-speak-repeat* loop: at a distance of one-half mile from the proximate maneuver point, the application checks whether it received a landmark description for the maneuver point from Route Manager, as part of the initial route request. If not, it sends an HTTP GET request to Route Manager seeking an updated instruction. If the processing for the maneuver point is now complete, Route Manager responds with the updated instruction.

At a distance of one-half mile, and again at one-quarter mile from the maneuver point, the application alerts the user to the upcoming maneuver via a spoken direction of the form *"in one-quarter mile, [action] at the [landmark] onto [street]"* where `action` is a predefined description of the maneuver to be performed, such as turn right or merge, `landmark` is the landmark description computed by Torchbearer, and `street` is the name of the street onto which the maneuver will take the user. In the case where no landmark description is available, either because the pipeline did not finish processing the maneuver point in time or was unable to compute a description, the "at the [landmark]" portion of the instruction is omitted. At a distance of 25-feet the application will speak an instruction of the form "[action] at the [landmark] onto [street]". When the user passes through the maneuver point, executing the maneuver, the *check-speak-repeat* iteration for the current maneuver is complete. The relative distances at which the app speaks directions were selected in a best effort to maintain parity with the Google Maps navigation app. While we do not consider vehicle speed in timing when to deliver the "turn now" instruction, this would be a relevant area for future work.

While the *check-speak-repeat* routine is the same for alerting the user of arrival at their final destination as for intermediate maneuver points, the delivery varies slightly: immediately after completing the second-to-last maneuver (the last maneuver being

arrival at the destination) the application speaks an instruction of the form *"In [distance] your destination is the [landmark] on the [side] of the street"* where `side` is either "left" or "right".

Upon arrival at the destination, the application speaks one last instruction of the form "you have arrived at your destination. It's the [landmark] on the [side]." The arrival event completes the navigation session, and the application returns to a point from which the user can enter a new destination and begin a new session.

The mobile application is implemented using the React Native framework [16], a cross-platform, JavaScript-based library. This framework allows for a single codebase across both iOS and Android, and while it is written in JavaScript as opposed to the native Swift or Java, all visual components are rendered natively on the device. This creates a highly-responsive interface that feels like a native application as opposed to a mobile website.

Street-level Imagery

Much of Torchbearer's work, whether human-based or machine-based, relies on visual computations based on the visual scene a driver would be seeing as he approaches a maneuver point. This computation requires a source of street-level imagery, photographs, taken from a vehicle on the road. These images must be of a relatively high definition (at least 640 pixels by 640 pixels), be in color and be available at all maneuver points through which Torchbearer provides navigation services (ideally, most roadways in the United States). Additionally, images must have no distortion, either by attribute of camera setup or post-production correction. That is, each image much be rectilinear.

We use Google Street View due its high coverage of U.S. roads, high definition images, and public availability. The service can return an image for a particular

latitude, longitude and compass bearing. The service returns rectilinear-projected [2], distortion free images for a given latitude, longitude, field of view and bearing. Field of view is limited to 120 degrees, as any larger can lead to incorrect perspective near the vertical edges of the image—a side effect of rectilinear projection.

Human Input

Torchbearer makes decisions via two means—algorithms and humans. Leveraging human opinion and decision-making in a computational system presents unique challenges, which are not considerations in most computer systems. Human input provides Torchbearer with insight into the landmark description problem that may be difficult to express algorithmically: while our machine-based pipelines include well-founded heuristics for finding and describing the best landmark to use for describing a maneuver point, we hypothesize that humans may offer some unique insight into solving this problem that our heuristics do not. The subjective nature of determining the best landmark, as well as a description of what that landmark is, make human insight and opinion especially valuable.

In order to gather human input at a large scale, in real time, we require a large source of human workers. For this Torchbearer leverages Mechanical Turk (MTurk), a large-scale crowdsourcing platform. Mechanical Turk manages a pool of workers, and allows requesters (such as Torchbearer) to submit Human Intelligence Tasks (HITs) to this pool. At a high level, a HIT is simply a question to be asked of a human worker, with some form of answer specification. Torchbearer presents HITs to workers via a web page hosted on its servers, allowing for rich content to be displayed to the worker. The demographics of the Mechnaical Turk worker pool are not restricted; we allow any worker to work on Torchbearer HITs so long as they pass the qualification test detailed in Section 3.

Each HIT specifies a monetary reward, which Torchbearer pays to the worker upon successful completion of the HIT, as well as a maximum duration the worker is allowed to work on the HIT. Workers are compensated at a rate chosen to be above the average pay for similar work; this removes any concerns about inferior results due to sub-par wages. Workers are paid eight cents for selecting landmarks in an image (similar to the more general object tagging task, which is common on Mechanical Turk) and 10 cents for to describe a landmark (similar to the common image captioning task). Workers are paid three cents to verify the accuracy of a description.

Additionally, the HIT can demand that the worker has a certain qualification—a test created by Torchbearer, which the worker must pass—in order to be allowed to work on the HIT. Lastly, the HIT specifies how many workers it should be completed by, allowing for the collection of multiple answers to the same question. When a HIT is submitted to MTurk, it becomes available for workers to complete. Workers choose the HITs they work on; they are not automatically assigned. Once a HIT has been completed by the specified number of workers, the answers are sent back to the requester by adding them to a distributed queue.

While each human-based pipeline task specifies its own format for questions and answers, the general system Torchbearer uses for gathering data via MTurk is constant. When a pipeline task requests human input, Torchbearer's MTurk management service (Turk Service) submits a HIT to MTurk with the parameters specified by the pipeline task. Some questions are simple in terms of how they can be displayed to the worker. They may consist of a text-based question with text-based answers, for example. Such questions are submitted to MTurk as part of the HIT specification, and are hosted entirely by MTurk. (The Description Verification task, which we discuss in detail later on, is an example of this type of HIT.) Other questions

may require displaying rich content to the worker and accepting interactive answers, such as the drawing of boxes around landmarks in an image. These questions must be served to workers as HTML pages by Turk Service, and the HIT only specifies the URL of the given page. When a worker is ready to complete the task, MTurk requests the page from Turk Service, and displays it to the worker.

MTurk collects answers from workers as they complete each HIT. Once a HIT has been completed by the number of workers required, MTurk sends the list of answers back to Torchbearer by adding a message to a distributed queue shared between Torchbearer and MTurk. Turk Service continuously polls this queue for new lists of answers. When one arrives, Turk Service first determines the aggregated answer—the final answer based on a combination of the individual answers of each worker—by applying an aggregation function. (The aggregation function varies by HIT; we discuss the specific function for each human-based task in the Pipelines section.) Turk Service sends this aggregated answer to the Orchestrator of the pipeline, and pipeline execution continues.

Getting Meaningful Answers

The primary challenge associated with asking a question of a worker is that of trust: do we trust that the worker gave us a meaningful answer, that she took the time to give the best response, as opposed to the easiest? Additionally, do we trust that she actually understood the task?

In the simplest scenario, for a specific human input question, Torchbearer submits a single HIT to MTurk, and accepts the response from the worker who completed it as the final answer. While straightforward, this approach does not offer confidence in how meaningful the response is—it is possible the worker put minimum effort into the HIT in the name of speedy completion. To counteract this,

Torchbearer makes use of two separate methods for filtering out nonsensical human answers: sampling and majority verification. Additionally, we require that all MTurk workers complete a training exercise and pass a qualification test specific to the task they are working on prior to submitting any results. The training materials and examination are hosted by MTurk.

Worker Qualification Leveraging MTurk's Qualification system allows us to filter out workers who do not understand the goal of Torchbearer's HITs. This screening allows us to both train the worker in how to complete a task as well as ensure they have the understanding and insight needed to complete the task successfully. While qualification does not prevent a worker from providing (either intentionally or unintentionally) a bad answer, it does ensure that they are capable of providing an answer of acceptable quality.

The qualification system consists of two components, training and enforcement. The training component consists of a web-based guide to completing the given task, complete with good and bad examples, descriptions of the goal of the task and step-by-step instructions. When a worker first desires to work on a Torchbearer HIT, she is presented with this guide. After viewing it, she may take the qualification test, a short multiple-choice exam which asks the worker to pick the best answer to an example HIT. Even though some questions may be largely opinion-based, the answer set is clear as to which choice would be an acceptable answer. Other answers have a glaring inconsistency which the guide would have specifically pointed out as being undesirable–such as selecting a non-permanent object as the best landmark. An example test can be seen in Appendix **??**.

In order to be allowed to work on Torchbearer HITs, a worker must score at least 80% on the qualification test and have viewed all parts of the training guide. Until

these requirements have been met for a given type of HIT, MTurk will not allow the worker to submit answers.

Sampling In the sampling approach, we require that multiple workers complete each HIT, providing us with multiple responses. We can then determine the final answer by applying an aggregation function to the individual responses, such as taking the mean or median or mode.

We benefit in two ways from this approach: first, having a majority of meaningful responses dampens the response of a negligent worker. Consider the trivial example of a HIT asking workers to count the number of cars appearing in an image. If we asked only a single worker, we would have to take her at her word, with no means of knowing how correct or incorrect her response was. However, if we ask five workers, and three provide the correct count while two provide the incorrect count (whether by intentional neglect or honest mistake) we could still arrive at the correct answer by either taking the median or the mode. Of course, the increased cost of this approach is directly proportional to the number of workers we ask to complete our HIT.

The second benefit comes into play if there is more than one answer to the question being asked, or if the question is largely opinion-based. Consider an example where we want to know which car in an image is the nicest, or most luxurious. Obviously, this is not an objective question–but we may still be able to gain insight by looking at the most frequent answers given by our sample of workers. If four workers suggest that car A is the nicest, one suggests that car B is the nicest, and one suggest that it is in fact car C, then we have reasonable evidence that car A is considered to be the most luxurious.

The sampling approach is powerful, but works best when the answers are quantitative and can be easily aggregated. For HITs which require answers that

are difficult to aggregate and compare the majority verification approach is best.

Majority Verification Instead of requiring multiple workers to answer each HIT, the majority verification approach, inspired by Kulkarni et al. [30], requires only one answer. However, to ensure that the given answer is correct, a sample of workers (generally three) is asked to confirm that the answer is correct. This verification is treated as a majority vote: if at least two out of three workers assert that the given answer is correct, we trust that answer.

This method can be more cost effective, as asking a worker to vote on the correctness of an answer is cheaper than asking them to define the answer for a complex task. Additionally, this approach does not require an aggregation function be defined, which is convenient for answers which are difficult to quantify, such as text-based answers. One important limitation of this approach is that it will not work well with opinion-based quandaries, such as the most luxurious car in an image. The voting pool is unlikely to agree on whether an answer is correct, since they themselves have opinions which can differ from that of the worker who provided the answer. Instead, this approach is well-suited to HITs which have an obvious correct answer, such as the number of cars in an image.

Pipelines

The problem of determining the optimal landmark description for a maneuver point consists of two main tasks: determining salient landmarks within the drivers view of the maneuver point and creating lexical descriptions of those landmarks. We refer to these two broad tasks as the saliency and description tasks, respectively. We propose two methodologies for solving each task, giving a total of four pipelines.

Our approaches are based on two principal methodologies–human-based and

machine-based. To this end, we have created one pipeline which is entirely machine based, another which is entirely human based, and two others which are hybrids of machine and human computation.

Pipelines are referenced via a method-method notation, where method can be either human or machine. The left-hand method refers to the method used for selecting salient regions of the maneuver point; the right-hand method refers to that for deriving a description of a given region.

Pipelines at a High Level

While the exact manner in which a pipeline solves the landmark description problem varies from pipeline to pipeline, all pipelines share a general sequence of execution, and all take a tuple consisting of latitude, longitude and bearing as input and yield a tuple containing the best, most salient selected landmark as output.

The first step in any pipeline is to obtain street-level images of the maneuver point at the given geographic coordinate and bearing (number 1 in Figure 3.3). For each maneuver point, Torchbearer gathers street-level images from three points relative to the maneuver point: "at", "just before" and "before" the intersection, corresponding to 25, 50 and 100 feet, respectively. (See Figure 3.4.) When directions are spoken to the end user, these positions are inverted, into "at", "just after" and "after", describing the position of the maneuver point relative to the image the selected landmark was found in.

We use imagery from these three positions in order to obtain a "view" of the maneuver point that captures landmarks of different scales—from signs right at the intersection to buildings which may only be visible from farther back from the intersection. The closest (25-foot) distance was selected as it is the closest distance at which a stop sign generally becomes visible in a Google Streetview image; the farthest

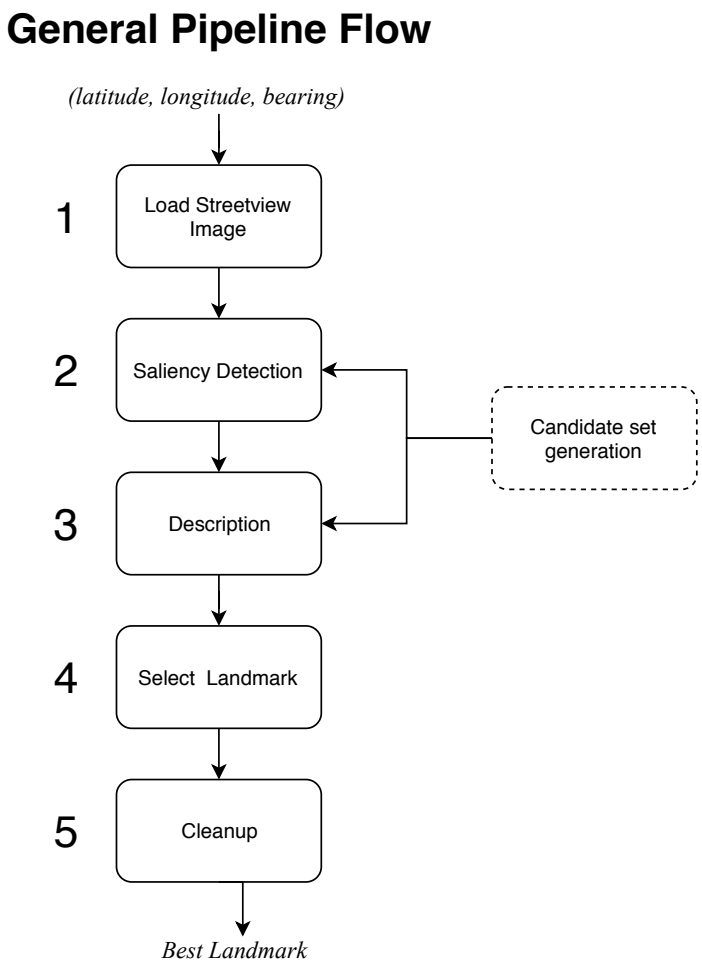


Figure 3.3: The general structure of a Torchbearer pipeline.

distance was selected as buildings on the side of the road near the intersection became visible.

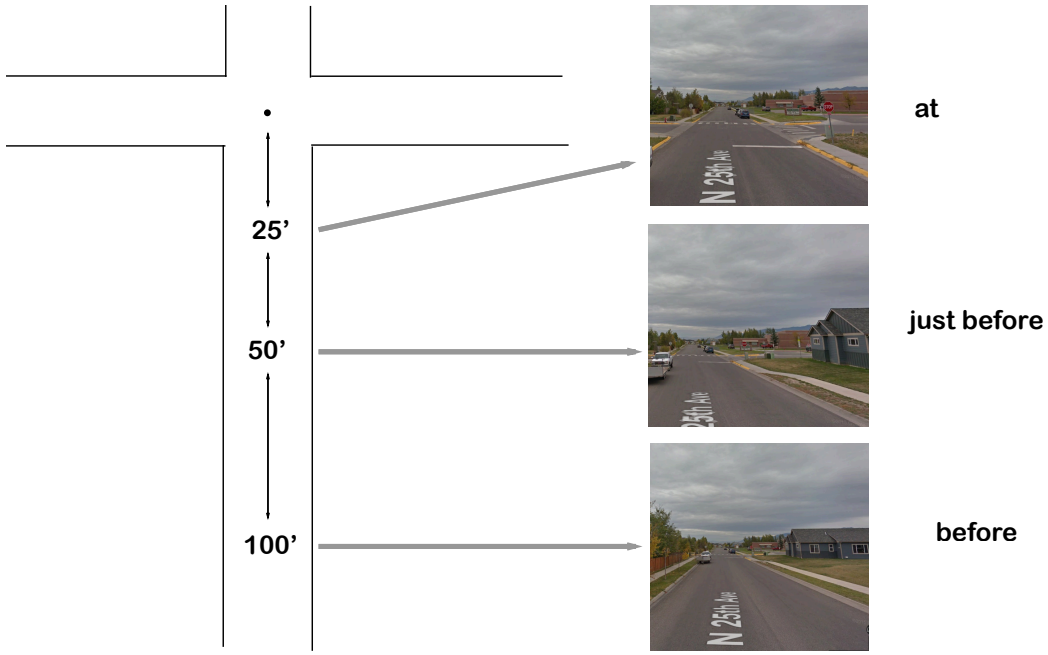


Figure 3.4: The positions of street-level images relative to a maneuver point.

No matter the exact approach the pipeline takes to obtaining a landmark description, it will need these images to perform its determination. Next, the pipeline must generate a set of candidate landmarks, C. A candidate landmark is simply an object at the maneuver point that could be used as the basis for a landmark-based instruction; we know nothing about how salient the landmark is, however.

Generation of the candidate landmark set is performed implicitly by either the saliency or description step, depending on the specific pipeline. Pipelines which leverage human-based description rely on the saliency step (whether human or machine-based) to generate a candidate set (step 2 in Figure 3.3). Pipelines which use machine-based landmark description generate a candidate landmark set as part of the description step (step 3 in Figure 3.3).

After the saliency of each candidate landmark has been determined (step 2 in Figure 3.3), and each candidate has received a lexical description (step 3 in Figure 3.3), the pipeline must decide which landmark is best, or most salient (step 4 in Figure 3.3). The formula varies by pipeline, and depends on which components of saliency were measured.

Saliency

The saliency step of a pipeline is responsible for quantifying the saliency of candidate landmarks. While saliency consists of three components—visual, semantic and structural, not all three components are considered individually in each pipeline. Human-based saliency is based on only a single overall score, generated by human opinion, that represents humans' ability to distinguish good landmarks. Machine-based pipelines consider both semantic and visual saliency. In all pipelines, structural saliency is *enforced* rather than *evaluated*: in accordance with the literature, we consider only candidate landmarks that are located at or very near to the maneuver point.

The Human Approach Humans are accustomed to picking out landmarks from their surroundings in day-to-day life, be it for giving a friend directions or for their own internalization of a route or location. We can take advantage of this innate ability by asking a human MTurk worker to select what they believe is the most salient, most standout landmark at a given maneuver point. Unlike algorithmic saliency detection, here we do not separate the concept of saliency into its visual and semantic subparts. Rather, we hypothesize that, because human workers have an elemental understanding of what makes a landmark salient, the decisions they make regarding the best landmark at a given point implicitly incorporate these saliency concepts.

We gather human saliency detection input via an MTurk HIT, denoted a "Saliency HIT". The Saliency HIT must be completed by five workers, and consists of the following task: after the worker elects to work on the HIT, he is shown a high-resolution image of the maneuver point in question from three distances (at, just before, and before the point, corresponding to 25, 50, and 100 feet, respectively). Note that all three of these images are of equal dimensions. The HIT instructs the worker to use his mouse to draw a bounding box tightly around the object that he believes is the best landmark—the one he would use if he were telling a driver to perform the given maneuver right at that point. The worker can choose an object in any of the three images, but can only select one object.

We offer the worker three images from three distances so that landmarks of different scales can be captured: a stop sign, for example, is hard to detect in an image from far away, but is prevalent in an image from right near the maneuver point. Likewise, a large building may be an excellent landmark, but might only be visible from some distance away from the maneuver point. In essence, we are showing the worker the approach to, or the path leading up to, the landmark, and allowing them to see what the driver would see at three points along this path. In the final instruction spoken to the driver, we take into account the position of the best landmark—that is, if the best landmark is one which was selected from the just before image, the spoken instruction will tell the driver to turn just after the specified landmark.

After the worker makes his selection, the Torchbearer-hosted webpage submits the coordinates of the drawn bounding box, along with the position corresponding to the image the box was drawn in, to MTurk. After five workers complete the task, MTurk sends the set of five bounding boxes back to Turk Service, via a distributed queue. Torchbearer must now aggregate these answers: this particular human task

leverages the sampling and aggregation approach to human input described in a previous section. That is, because bounding box coordinates are quantitative, we can combine them together in a manner which rewards the agreement among workers, if there is any, and culls answers which are in the severe minority and likely to be meaningless.

Turk Service performs aggregation by creating a matrix called a *saliency map* for each of three maneuver point images; this matrix represents the number of workers who included each pixel in the bounding box they drew. Algorithm 3.1 creates this matrix. The result of this operation is a matrix of size equal to the maneuver point images shown to the worker, where an element corresponds to a pixel in the original maneuver point image and where the value of each element is an integer between 0 and n, where n is equal to the number of workers. While the saliency map does not incorporate any decision about which regions are or are not salient landmarks, it encodes the relative saliency of each pixel in the image. To make this matrix easier to work with in subsequent pipeline steps, we normalize all values between 0 and 255, where a value of 255 indicates maximal saliency. A subsequent task in a pipeline can use this saliency map to either find the most salient regions or to query the total saliency of a target region.

The Machine Approach The algorithmic approach to determining salient landmarks consists of separate components for visual and semantic saliency. However, the machine-based saliency step deals only with visual saliency; the machine-based description step provides semantic saliency scores.

Visual saliency refers to the perceptive quality of a region of the drivers view which causes that region to stand out from its neighbors–that is, the degree to which a region grabs a drivers visual attention. Street-level imagery of a maneuver point

Input: B, a set of tuples (x_1, y_1, x_2, y_2) representing bounding boxes; m, the width of maneuver point image; n, the height of maneuver point image;

Output: S, a matrix of dimension m by n

1: $S \leftarrow 0_{m,n}$

2: **for** $b \in B$ **do**

3: $S[b_{y1} : b_{y2}, b_{x1} : b_{x2}]$ += 1

4: **end for**

5: **return** B

Algorithm 3.1: Creating a saliency map from human input

serves as input; the goal is to quantify each pixel of a maneuver point image in terms of its relative visual saliency. Specifically, given an m x n input image of a maneuver point, we output an m x n *saliency map*, where each element in the matrix is an integer between 0 and 255 corresponding to how visually salient that pixel is. A value of 0 indicates no saliency, while a value of 255 indicates maximal saliency.

Torchbearer leverages a state-of-the art, deep-learning based algorithm called SalNet [44] to estimate the pixel-level visual saliency across an image. Rather than seeking to identify specific neuroscience-inspired image features, which identify saliency, as many previous approaches do, SalNet takes a completely data-driven approach, using a deep convolutional neural network to learn where the human gaze tends to fixate in different images. Training data consists of a large dataset of ImageNet [13] images, each with a corresponding ground truth *saliency map*. This dataset was created by tracking subjects' gaze as they were shown each image and recording the time gaze was focused on each pixel. These gaze times were then normalized to between 0 and 255, inclusive.

SalNet uses a deep neural network architecture to predict the saliency map

for an input image. The first three layers of this network consist of pretrained layers from a Visual Geometry Group image classification network, VGG16 [57]; the authors recognize that the low-level features learned by these layers offer valuable input to the saliency problem. VGG16 was trained on an extremely large dataset, and by using transfer learning, SalNet can benefit from this extensive training without needing to train on so many images itself.

After the pretrained VGG network, SalNet incorporates a series of convolutional and pooling layers, and finally a deconvolutional layer, which will cast the output back into a matrix of the same size as the input. Training of the neural network consists of minimizing the Euclidean distance between the saliency map output by the network and the ground truth saliency map provided by the training dataset. During training, the weights of the first three layers are fixed at the pretrained weights from the VGG16 network; only the additional, saliency-specific layers unique to SalNet are actively trained.

It is important to note that SalNet is trained on a wide range of ImageNet images from across a broad range of topics; it does not incorporate any knowledge specific to the navigation domain. At the time of writing, no dataset containing ground truth saliency maps for street-level imagery of sufficient size for training a neural network was available. Training the SalNet architecture with domain-specific data would certainly be worthwhile future work. However, the general principles of visual saliency are not specific to any single domain, and the generalized training of SalNet allows it to perform well on an evaluation set of images from across the ImageNet corpora. We have hypothesized that it can adequately generalize to the navigation domain.

Figure 3.5: Left: a maneuver point image. Right: a corresponding saliency map generated by SalNet

Description

The second half of the landmark selection problem consists of deriving a lexical description of a candidate landmark, although the machine approach to description is also responsible for generating candidate landmarks as well as providing semantic saliency scores. This description should be specific enough so as to allow a driver to easily distinguish that given landmark from its surroundings.

The Human Approach To gather human descriptions for a given landmark, we again leverage Mechanical Turk. However, instead of using a sampling approach as we did with saliency crowdsourcing, we use a verification approach. First, for a candidate landmark c, we annotate the street-level image of the maneuver point for which this landmark is a candidate to include a bounding box drawn around the landmark. We create an MTurk HIT is with only a single assignment; the worker is shown this image and asked to describe the object enclosed in the bounding box. The exact format of the question is: *"Provide a specific description of the main object in the box. Describe PERMANENT, man-made things–NOT cars, people or things that could move. Pretend you were using that object as a landmark when giving someone directions."* The Torchbearer-hosted webpage presents the worker with a text box into which to type their answer.

After the worker has submitted the description, we create a verification HIT on MTurk, with three assignments. The annotated maneuver point image, along with the candidate description, is shown to each worker. The worker is asked to decide whether the description is accurate and meets the criteria of describing permanent, man-made things–not cars, people or things that could move. Three radio buttons are displayed–"Description is accurate" and "Description is inaccurate and the landmark is valid", and "not a valid landmark".

If at least two of the three workers indicate that the description is accurate, the description is accepted, and pipeline execution can continue. If at least two of the three workers indicate the landmark is invalid, pipeline execution continues, with this landmark removed from the set of candidates. If the majority of workers indicate that the description is incorrect, or if there is no majority opinion, the description process repeats, with the creation of a new description HIT and subsequent verification HITs. Torchbearer will retry this process up to three times—if no description could be derived, the landmark is removed from the candidate set and pipeline execution continues.

The Machine Approach Torchbearer leverages two approaches for finding semantically salient landmarks and quantifying their salience: a data driven approach, which uses a geosocial datasource to estimate the local significance of business and points of interest, and a deep learning-based object detection algorithm, which searches for known types of semantically salient features in maneuver point images.

Data-driven Approach Torchbearer estimates the semantic saliency of a landmark via an estimate of the number of people, who have recently visited the landmark counted by the social networking application FourSquare. Previous work has shown the efficacy of using geosocial streams as a proxy for the local importance of a

landmark; the intuition being that the more people who have checked in to a given location, the more well-known, or prominent, it is [47]. FourSquare incorporates businesses, points of interest and publicly accessible places into its ecosystem; these are referred to as venues. User location is recorded transparently, without the need for the user to explicitly tap a "check in" button Torchbearer leverages FourSquares venue data both to find candidate landmarks and determine their semantic saliency.

To find candidate landmarks for a given maneuver point, Torchbearer queries FourSquare for venues which are within a given radius of the maneuver point. By default, we use a small 100-foot radius, in the aim of ensuring any returned venue will be on or near the road upon which the maneuver point is located. FourSquare returns a list of tuples consisting of the venue name, the type of venue (such as restaurant, gas station, etc.) the geographic coordinates and the number of FourSquare users who have checked in to that venue.

We compute the relative bearing between the venue and the approach bearing of the user, and discard venues which are not within 45-degrees of either side of the user, as the field-of-view of our street-level imagery is 90-degrees. We convert each of these venues to a Landmark: the landmark's description is the name of the FourSquare venue, concatenated with its category. For example, the description for a landmark corresponding to a venue with the name "Starbucks" and category "Coffee Shop" would be "Starbucks Coffee Shop". The landmarks semantic saliency score, S_s is a function of the number of checkins in the last six months c and the number of locations l, if the venue is a chain:

$$S_s = c + l \tag{3.4}$$

This measure captures both the local significance and wide-area ubiquity of the

landmark. Note that all saliency scores are relative, and are meant to be compared against other candidate landmarks at a maneuver point.

We determine the position of the landmark relative to the maneuver point based on its proximity to the maneuver point: if within 50 feet, the position is "at". If not, the position is "after". These positions are inverted (into "at" and "before", respectively) if the landmark is selected for inclusion in a spoken instruction to an end user. Figure 3.6 shows this determination.

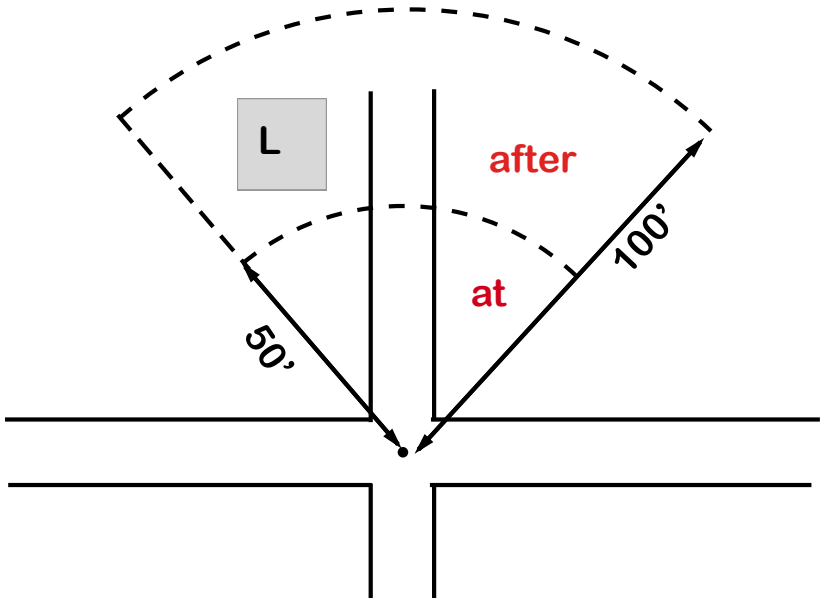


Figure 3.6: Determining landmark position for data-driven description approach. We consider landmarks within the 50-foot inner radius to have a position of "at", and those within the 100-foot outer radius to have a position of ""after". For example, landmark L in this diagram would have a position of "after".

Object Detection Approach Some landmarks are ubiquitous and proven to be highly semantically salient, independent of the maneuver point's geographic location. Road infrastructure, such as stop signs and traffic lights, is a prime example: these landmarks are universally recognizable among drivers, and have been shown to serve as excellent landmarks for use in navigation instructions [38]. Unfortunately, we found

no dataset of street signage or traffic lights with coverage beyond a specific locality. Instead, we leverage a state-of-the-art object detection algorithm, Faster-RCNN [50], to detect stop signs and traffic lights at maneuver points. Note that as a direction for future research, extending the object detection model to include other types of landmarks is both feasible and potentially beneficial.

Faster-RCNN (FRCNN) is a deep, region-based, convolutional neural network which takes an image as input and yields a set of bounding boxes, class labels (a string denoting which object the region was classified as) and confidence scores for objects of interest detected within the image [50]. It is currently one of the highest performing classifiers in terms of both speed and accuracy [53], [50].

FRCNN leverages an existing image classification network, ResNet, to compute feature maps for an image, and then uses the output of an intermediate convolutional layer in that base network as input to its own FRCNN-specific layers. This inclusion of a network trained for large-scale classification is known as transfer learning, and allows an FRCNN model to take advantage of the extensive training across millions of ImageNet images encoded within ResNet. The output of this intermediate convolutional layer, although trained on ImageNet data, outputs high-level image features as opposed to specific classes probabilities. Using these high-level feature maps as input, FRCNN trains its own final (fully-connected) layers to output class probabilities specific to our data.

FRCNN consists of two sub-networks: a Region Proposal Network (RPN), trained to output a set of possible bounding boxes, and the CNN network itself, which performs classification and final bounding box adjustment (based on the predicted class). To predict likely bounding boxes, the RPN considers a pre-generated set of *anchor boxes*. Each anchor box is a fixed set of 9 candidate bounding boxes, of different sizes and aspect ratios, anchored at every point in the image. For example,

if the input image is of dimensions n x n, there are $9n^2$ anchor boxes for the RPN to consider. For each anchor box, the RPN learns to output (through the training of three convolutional layers) a probability corresponding to the likelihood of the box containing an object of interest, as well as a tuple of four doubles indicating the amount by which to adjust each coordinate of the predefined anchor box. Boxes with a probability of objectiveness below a certain threshold are discarded, the rest are passed on to the classification sub-network.

Given a set of possible bounding boxes generated by the RPN, the CNN first uses Region of Interest Pooling (ROI) to generate fixed-size convolutional feature maps corresponding to the region of the input feature map contained within each bounding box. ROI consists of splitting the box into k evenly sized regions and selecting the maximal value from each region, yielding a feature map of size k, where k is a small integer, often 7. This pooled feature map is then input to two successive 4096-neuron fully-connected layers–these two layers learn the actual classification function.

The output of the second fully-connected layer is passed through a softmax layer of size equal to $c + 1$, where c is the number of classes we are trying to predict. (The extra output is for the "background" class–a bounding box that did not contain an object.) The softmax layer gives a floating-point number for each output, subject to the following constraint: let Y be the set of outputs, then

$$\sum_{y \in Y} y = 1 \tag{3.5}$$

This gives a probability distribution over the set of possible classes for the likelihood of an object being a particular class (or background).

In addition to the softmax output corresponding to class predictions, the network outputs a tuple of bounding box adjustments corresponding to each class. (These

are output via a single fully-connected layer of size $4c$.) These adjustments capture information about how to transform a pre-generated anchor box into the correct shape for a class; for example, it will learn that a stop sign is square.

Using images from Google Streetview, we constructed a dataset of 800 street-level images and ground-truth bounding boxes. Ground truth labels were created by hand using the Visual Object Tagging Tool [12]. Each image contained traffic lights, stopsigns or both. We generated an addition 75 negative examples—images containing neither a stoplight nor a stop sign. This dataset was divided into training and test sets, with a split of 85% train and 15% test. We trained an FRCNN network for 20 epochs–that is, 20 complete passes through our training set. At the completion of training, we achieved a mean average precision on our test dataset of 0.71 for stop lights and 0.75 for stop signs.

Finding Landmarks in Saliency Maps

Given a saliency map, it is often important to locate candidate landmarks based on hot spots, or highly salient regions, in the map. The significance of this is different for human-based saliency detection than for machine-based saliency detection. As an example, consider the street-level image and corresponding saliency map shown in Figure 3.7.

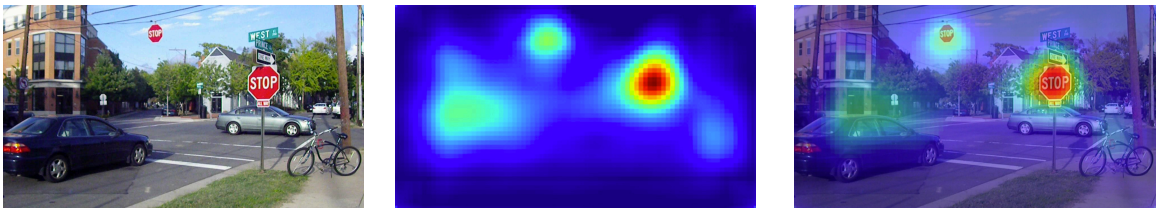


Figure 3.7: Left: a street-level image, with two stop signs and a building as potentially salient landmarks. Center: the corresponding saliency map, generated by SalNet. Right: the saliency map overlaid atop the street-level image.

With human-based saliency detection, the goal is to reduce the set of returned bounding boxes into a reduced set of distinct landmarks, by combining overlapping bounding boxes into a single area. For example, of the five answers it might be that three bounding boxes mostly overlap, indicating that that those workers intended to select the same landmark, while the other two answers overlap a separate landmark. Rather than treat all five bounding boxes as separate landmarks, it is beneficial to instead consider only the two distinct landmarks. First, this reduces the scale of future pipeline operations—those steps do not need to perform (redundant) calculations on as many candidate landmarks. This reduction saves time and compute cycles and, in the case of human-based tasks, fees paid to workers. Second, by reducing bounding boxes into aggregated areas, we can assign a saliency score to the candidate landmark based on how many answers included it in their bounding box. This can be used at the end of the pipeline as part of the decision process for choosing the best landmark. It is this score that acts as proxy for human intuition into what makes the best landmark: the more workers who select the pixels containing a landmark, the more salient the landmark.

In the case of machine-based candidate landmark generation, we need to correlate the set of candidate landmarks generated by the machine description (FourSquare-based) step with an area of the visual saliency map. Only the latitude, longitude and relative bearing between street-level image and landmark are known. We need to locate potential salient regions in the saliency map, so that we can determine if the candidate landmark aligns with one of those regions. Given a saliency map, a matrix of values ranging from 0 to 255, the goal is to label each pixel as belonging to a specific salient region or being non-salient.

Non-maximal suppression (NMS) is a state-of-the-art method for reducing a set of bounding boxes to only the significant ones, discarding bounding boxes which

enclose the same region using greedy clustering and a fixed distance threshold [40]. If our saliency map were composed of entirely rectangular regions of different saliency values (as is actually the case with human-based saliency detection) this method would be sufficient. However, the saliency map returned by our computer-vision based saliency algorithm estimates saliency at the pixel level and, as a result, makes no guarantee about the shape of salient regions.

The Watershed Algorithm is an image segmentation approach, designed to single out distinct regions in the image by separating foreground elements from background elements [3]. In classic image processing, these regions might be objects one wishes to separate from one another. In our case, we wish to separate regions of relatively high saliency (foreground) from their low-saliency surroundings (background). The algorithm works by considering our saliency map as a topological surface, where the value of a pixel denotes its height–pixels with a value of 0 (no saliency) are valleys and pixels with a value of 255 (highest saliency) are peaks. For each valley, or minima, in the map, the algorithm simulates filling the topology with different-colored water–that is, it labels pixels as belonging to a given segment. As simulated the water level rises, water from different valleys will begin to converge. To prevent this, the algorithm constructs infinitely tall barriers, or segmentation lines, between the two valleys. The algorithm continues this process until even the tallest peak is submerged, leaving only the barriers above water. These barriers now encapsulate different objects, or salient regions, within the map.

To make this algorithm more impervious to over-segmentation and noise–small regions of high salience within a low-salience area or vise versa–we leverage the marker-controlled watershed algorithm [51]. Here, we dictate to the algorithm which pixels we know to be independent, salient regions, which ones we know to be non-salient, background pixels and which ones we are unsure about (the border area

between known salient regions and non-salient background). Now, rather than flooding starting at the minima, the algorithm begins flooding from each foreground region we specified and the background region; it is now simply finding where the segmentation line will be placed within the unknown border area.

In order to apply the watershed algorithm, several preprocessing morphological steps must be taken to clean up the saliency map, and each pixel must be labeled according to its status as known background, known foreground, or unknown. We adapt a procedure outlined in [1]. The following steps outline this process, given a saliency map S:

1. Perform binary segmentation on S, rendering each pixel as salient (255) or non-salient (0). (This segmentation yields a "black and white" image.) We first compute a threshold t, at or above which a pixel is considered salient and below which a pixel is considered non-salient. We select t via Otsu Thresholding [42], which works by iterating through all possible threshold values in $[0, 255]$ and selecting the one which minimizes the sum of the weighted variances within the salient and non-salient classes. That is,

$$threshold = argmin_t(\frac{\mid n \mid}{\mid n \mid + \mid s \mid}\sigma_n^t + \frac{\mid s \mid}{\mid n \mid + \mid s \mid}\sigma_n^t) \tag{3.6}$$

   where t is the candidate threshold, s is the set of salient pixels, n is the set of non-salient pixels and σ is the variance within the given set of pixels when a given t is used as the threshold value. Figure 3.8 shows the saliency map after Otsu Thresholding.

2. Remove small, insignificant salient areas (white noise) by performing morphological opening on the binary segmentation. Figure 3.9 shows the saliency map

Figure 3.8: The result of applying Otsu Thresholding to the saliency map. White areas (having a value of 255) represent areas of saliency.

after applying morphological opening. While difficult to see at a small scale, several spots of white noise were removed.

Figure 3.9: The saliency map after applying both Otsu Thresholding and morphological opening. While difficult to see at a small scale, several spots of white noise were removed.

3. Remove small, insignificant non-salient areas (holes) by performing morphological closing on the binary segmentation. Figure 3.10 shows the results of this step; as this particular saliency map does not have any non-salient holes within a salient region the process had no visible effect.

4. Determine which pixels are known to be non-salient by dilating the binary segmentation, falsely enlarging the salient regions. Dilation consists of scanning a square kernel K over the binary segmentation and, at each point, replacing

Figure 3.10: The results of the morphological closing step; as the particular saliency map does not have any non-salient holes within a salient region the process had no visible effect.

the binary segmentation pixel underneath the anchor point (center) of K with the maximal value overlapped by K. Denote this dilation, shown in Figure 3.11, as M_n.

Figure 3.11: Dilation M_n: the parts of the image known to be non-salient are in black (values of 0). Notice that the salient (white) regions are slightly enlarged compared to the results of the previous step.

5. Apply a distance transformation to the binary segmentation, resulting in the value of each pixel being equal to the Euclidean distance between that pixel and a pixel with value 0 (non-salient background). This operation is essentially finding salient peaks, or the centers of salient regions, as the pixels which are farthest from a non-salient pixel are the ones in the center of a large salient

region. Denote this distance transform as D (shown in Figure 3.12).

Figure 3.12: Distance transformation D: the center points of the salient regions are exactly white (255), as they are the farthest from a non-salient (black) pixel.

6. Determine the set of pixels which are likely to be salient by applying a binary threshold to the distance transform, where t, the threshold, is set to $c*max(D)$, where c is a constant factor which we set to 0.7. The goal is to isolate those pixels which are far from any non-salient pixels, as we can be confident that these are salient pixels. Denote this threshold M_s, shown in Figure 3.13.

Figure 3.13: Threshold M_s, the white areas (values of 255) represent the areas of the saliency map we have high confidence are salient.

7. Pixels which are not known to be either salient or non-salient can be found by $M_u = M_s - M_n$. This subtraction is shown in Figure 3.14.

Figure 3.14: M_u, the result of subtracting the matrix of known background areas from the matrix of known foreground areas. the white areas (values of 255) represent the unknown areas between salient and non-salient (background) regions.

8. Each distinct (disconnected) region of salient pixels in M_s needs to be labeled from $2...n+1$, where n is the number of distinct regions. The background, or non-salient-pixels, must be labeled as 1. This is accomplished by performing a connected component analysis on M_s with 8-connectivity, yielding $M_{labeled}$, a matrix with consecutively labeled connected components. (This matrix is shown in Figure 3.14.) Label the unknown region with 0; this is the region in which watershed will draw a segmentation line to determine the final boundary around the salient regions. Specifically, $\forall p_{ij} \in M_u \mid p = 0, M_l[i,j] = 0$.

Figure 3.15: $M_{labeled}$, where dark blue is known non-salient background, purple is unknown, and yellow, green and turquoise are each a specific known salient region.

9. Run the watershed algorithm on S, using $M_{labeled}$ as markers. The returned

matrix M_w will have labeled all pixels as non-salient (1) or as belonging to a salient region (2...n+1). The result is shown in Figure 3.16.

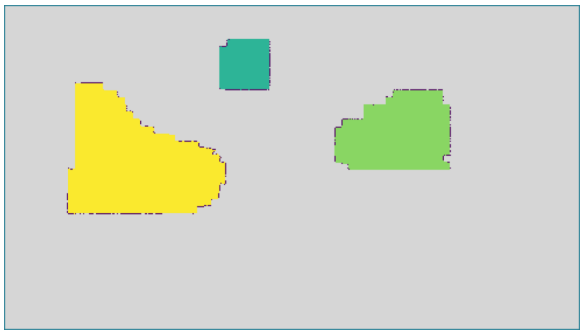


Figure 3.16: M_w, the result of the watershed algorithm. The grey region is non-salient background, and each of the colored regions is a distinct salient region.

10. Calculate the bounding box around each salient region in M_w; these are the saliency map's salient regions. The final bounded salient regions are show in Figure 3.17.

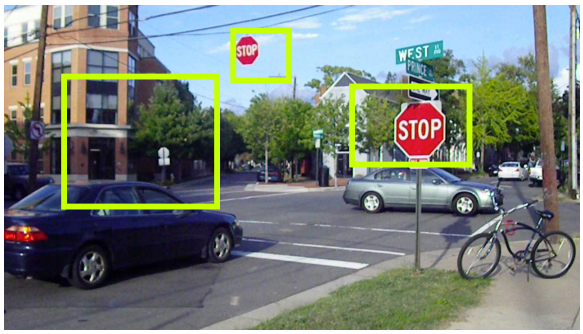


Figure 3.17: The final salient bounding boxes.

Quantifying Landmark Uniqueness

The semantic uniqueness of a landmark is an important factor in its saliency [10]. Even for pipelines that leverage human-based saliency detection, and therefore do not componentize the saliency score, uniqueness is still used for tie breaking purposes.

We use the lexical description of a landmark to derive its uniqueness as compared to the rest of the candidate landmark set. Our approach is based on word embeddings, where a word is represented as a high-dimensional vector in vector space [20]. The value in such a framework stems from the Distributional Hypothesis, which contends that words which are semantically similar will be distributionally similar as well, appearing together in the same written contexts [26]. The goal in creating vectorizations of a set of words is to represent semantically similar words with similar, i.e. close, points in high-dimensional space. This approach allows us to determine the similarity of words by comparing the Euclidean distance between the points or cosine similarity between the corresponding (normalized) vectors.

Word2Vec Predictive modeling is a common method for generating the vector representations of a set of words, wherein a machine learning algorithm learns to accurately predict a word's context, or words that are likely to appear around it, given only the word [4]. One such model, Word2Vec, is trained to predict a nearby word given another word, effectively internalizing a representation of which words appear in the same contexts [39]. The algorithm uses a neural network with a single hidden layer of size equal to the desired dimensionality of the word embedding (often 300). Using a large corpus of text, and a selected vocabulary of important words therein, the network is trained to accurately predict the probability of each word in the vocabulary occurring within a small window of other words in the vocabulary, within the text of the corpus. In doing so, the algorithm generates a v x d weight matrix (from the hidden layer to the output layer) which acts as a function from $word : P$, where v is the size of the vocabulary, d is the dimensionality of embeddings and P is a vector of probabilities for each word in the vocabulary. After training, each row in this matrix represents the embedding for a word in the vocabulary. Intuitively, if two words

are similar, they are likely to be surrounded by similar words, per the Distribution Hypothesis. Thus, they will have learned similar weights, so as to generate similar probability distributions over the vocabulary.

We use a pretrained word2vec model [22] with 300-dimensional word embeddings, trained on the Google News corpus and containing 300 million vocabulary words. We use cosine similarity as a measure of similarity between word embeddings, meaning that our similarity measure is bound between [-1, 1], with 1 indicating complete similarity and -1 indicting complete lack of similarity.

To find the similarity between two landmark description phrases, we compute a description vector, which is a sum of the vectors of each word in a description. We then calculate the similarity between the two description vectors. Given two candidate landmarks c_1 and c_2, the similarity s between these landmarks is defined as

$$pairSimilarity = (c_1, c_2) \longrightarrow c(\sum_{w \in c_1.description} embedding(w), \sum_{w \in c_2.description} embedding(w)) \tag{3.7}$$

where *embedding* is the word2vec vector for the given word and c is the cosine similarity function of two vectors v_1 and v_2:

$$c = (v_1, v_2) \longrightarrow cos(\theta) \tag{3.8}$$

$$= (v_1, v_2) \longrightarrow \frac{A \cdot B}{\parallel A \parallel \parallel B \parallel} \tag{3.9}$$

where θ is the angle between the two vectors.

To find the similarity of a landmark c as compared to all other landmarks in a

set of candidate landmarks C:

$$totalSimilarity = \sum_{k \in C} pairSimilarity(k, c) \tag{3.10}$$

Pipeline Specifics

Machine-Machine

Input: $X_0 = (latitude, longitude, bearing)$

Output: The most salient landmark, including a description for use in navigation instructions

Candidate selection: Description

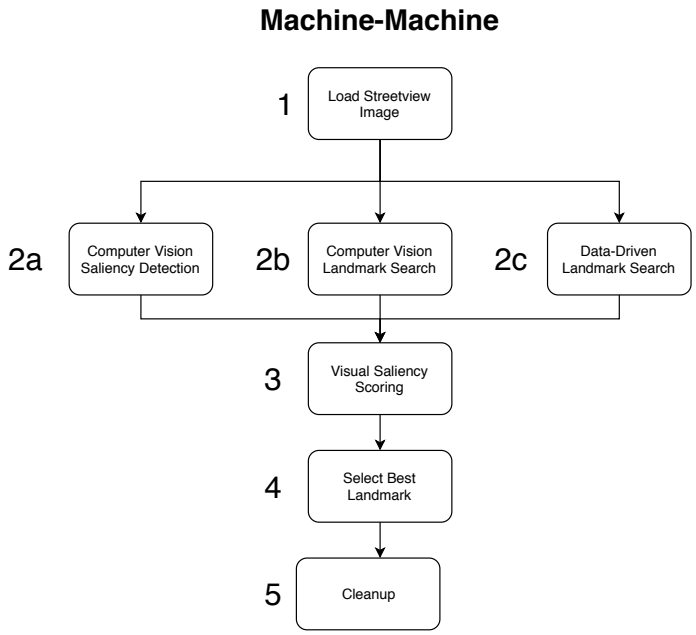


Figure 3.18: The pipeline structure of the Machine-Machine pipeline.

Step 1: Load Streetview Image

Input: $X_1 = (latitude, longitude, bearing)$

Output: $Y_1 = (latitude, longitude, bearing, [image\_urls])$

This step consists of querying the Google Streetview API for street-level imagery at distances of 25, 50 and 100 feet from the given coordinate at an angle opposite of the bearing, as shown in Figure 3.4. (We refer to these distances relatively as "at", "just before" and "before".) We store returned images on Amazon Simple Storage Service (S3), and include the S3 URL of each in a tuple which is included in the output of this step.

Step 2a: Computer Vision Saliency Detection

Input: $X_{2a} = Y_1 = (latitude, longitude, bearing, [image\_urls])$

Output: $Y_{2a} = (latitude, longitude, bearing, [image\_urls], [saliency\_maps])$

Implementing the machine approach methodology outlined in the Saliency section, this step uses the SalNet deep learning architecture to compute a saliency map for each image in the tuple of images in X_{2a}. The processing of the images happens in parallel and consists of feeding the the street-level image through SalNet.

For each image, this step yields a one-dimensional matrix of the same shape as the input image, with values ranging between 0 and 255, inclusive. We add each matrix to the output tuple provided to subsequent pipeline steps.

Step 2b: Computer Vision Landmark Search

Input: $X_{2b} = Y_1 = (latitude, longitude, bearing, [image\_urls])$

Output: $Y_{2b} = (latitude, longitude, bearing, [image\_urls], [candidate\_landmarks])$

This step uses the Faster RCNN-based object recognition algorithm, described in the Saliency section, to detect candidate landmarks in each maneuver point image. The network has been trained to detect stop signs and stop lights; it returns, for each object it detects, a tuple consisting of the coordinates of the objects bounding box within the image, a confidence score between 0 and 1 and a description (label) for the

object. We discard any objects with a confidence score less than 0.8 to avoid false detections, based on the notion that it is better from a usability standpoint to not provide a landmark description in an instruction than it is to provide a description of a nonexistent landmark. The remaining objects are converted into candidate landmark tuples, with a semantic saliency score of 1.0. (We assume that all users are fully aware of what a stop sign or stoplight looks like, thus no other landmark can be more semantically salient than a landmark detected by this step.) These landmarks are included in the output of this step.

Step 2c: Data-driven Landmark Search

Input: $X_{2c} = Y_1 = (latitude, longitude, bearing, [image_urls])$

Output: $Y_{2c} = (latitude, longitude, bearing,$
$[image\_urls], [candidate\_landmarks], [saliency\_maps])$

This step uses FourSquare, described in Section 3, to find candidate landmarks by searching for venues within a 100-foot radius of the maneuver point, as detailed in the Saliency section. Candidate landmarks are included in the output tuple.

Step 3: Visual Saliency Scoring

Input: $X_3 = Y_{2a} \cup Y_{2b} \cup Y_{2c} = (latitude, longitude, bearing, [image\_urls],$
$[saliency\_maps], [candidate\_landmarks])$

Output: $Y_3 = (latitude, longitude, bearing, [image\_urls], [candidate\_landmarks])$

This step assigns a quantitative score to each candidate landmark to designate its visual saliency in the context of the maneuver point image. The computer-vision based saliency detection approach (Step 2a) is not landmark-aware; that is, it determines relative saliency at the pixel-level. This step aggregates these pixel-level values into a score for the entire landmark. Given the bounding box coordinates x_1, x_2, y_1, y_2 of a candidate landmark and the saliency map S of the maneuver point,

the visual saliency score of that candidate is calculated as

$$score = \frac{\sum_{i=x_1}^{x_2} \sum_{j=y_1}^{y_2} S_{ij}}{\sum S} \tag{3.11}$$

That is, the visual saliency score is the sum of the submatrix contained within the bounding box divided by the sum of the entire saliency map. This gives two desirable properties: first, the larger a landmark is, the higher its score. Second, the more high-saliency pixels contained within a landmark bounding box, the higher its score.

While candidate landmarks detected by the object detection (Step 2b) include bounding boxes, and can therefore be correlated directly with a region in the saliency map, those returned by the data-driven approach (Step 2c) do not. For these candidates, only the relative bearing between the maneuver point and landmark is known. In order to estimate which rectangular region of the saliency map corresponds to these landmarks we must first locate salient regions within the saliency map and then determine if one of those regions lies on the given bearing.

To locate salient regions, we use the watershed-based approach described previously. This lends us a set of bounding boxes, each containing a salient region within the saliency map. To determine if one of these salient regions represents our candidate landmark, we consider two points about the street-level image off of which the saliency map was created: first, the pitch of the image is zero degrees, meaning that the horizon line, where a venue would be, is roughly in the vertical center of the image. Second, the field of view of the image is 90-degrees, and is not distorted or warped. Given the relative bearing between the maneuver point and the landmark, we check if there exists a salient region at this bearing in the vertical middle of the saliency map. (See Figure 3.19.) If there is, we use this region as the bounding box

for the candidate, and calculate the visual saliency score as above. If not, we assign a score of 0, as we have no evidence as to the visual saliency of this landmark.

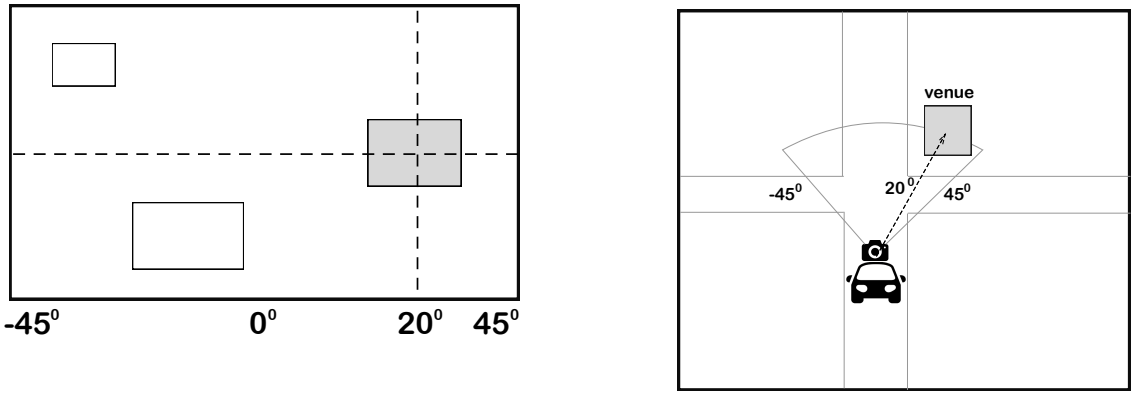


Figure 3.19: Left: a landmark saliency map, with bounding boxes of salient regions. The intersection between the relative bearing parallel and vertical middle is within a salient region (shaded), and identifies the landmark within the saliency matrix. Right: A bird's eye view of an intersection. Our street-level images are a rectilinear projection of a spherical image covering a 90 degree field of view.

Step 4: Select Most Salient Landmark

Input: $X_4 = Y_3 = (latitude, longitude, bearing, [image\_urls],$

$[saliency\_maps], [candidate\_landmarks])$

Output: $Y_4 = (latitude, longitude, bearing, best\_landmark)$

At this point in the pipeline, we have a set of candidate landmarks, each complete with both a visual and semantic saliency score. In order to determine the best, most salient landmark, we must first determine the uniqueness saliency score for each candidate, calculated via the method described in Section 3. Next, we normalize each of the three saliency scores to a value between 0 and 1. Given a set of candidate landmarks C, the normalized score for a given saliency component (visual, semantic

or structural) for a given landmark c can be found by

$$score_{component} = \frac{c_{score}}{max_{i \in C}(i_{score})} \tag{3.12}$$

where i_{score} is the score for landmark i for a given component.

The total saliency score for a candidate is then the sum of the three normalized scores.

$$S = S_v + S_s + S_u \tag{3.13}$$

where S_v is the visual saliency score, S_s the semantic saliency score, and S_u the uniqueness score, .

The candidate landmark with the highest summed scores is the best, most salient landmark, and is the output of this step. The description of this landmark will be included in navigation instructions spoken to the user.

Step 5: Cleanup

Input: $X_5 = Y_4 = (latitude, longitude, bearing, best\_landmark)$

Output: $Y_5 = (best\_landmark)$

This final step consists of system cleanup tasks. All intermediate images—namely, street level imagery—stored on S3 are removed. The best landmark is stored in a database, associated with the maneuver point and pipeline identifier for future retrieval.

Human-Machine

Input: $X_0 = (latitude, longitude, bearing)$

Output: The most salient landmark, including a description for use in navigation instructions

Candidate selection: Description

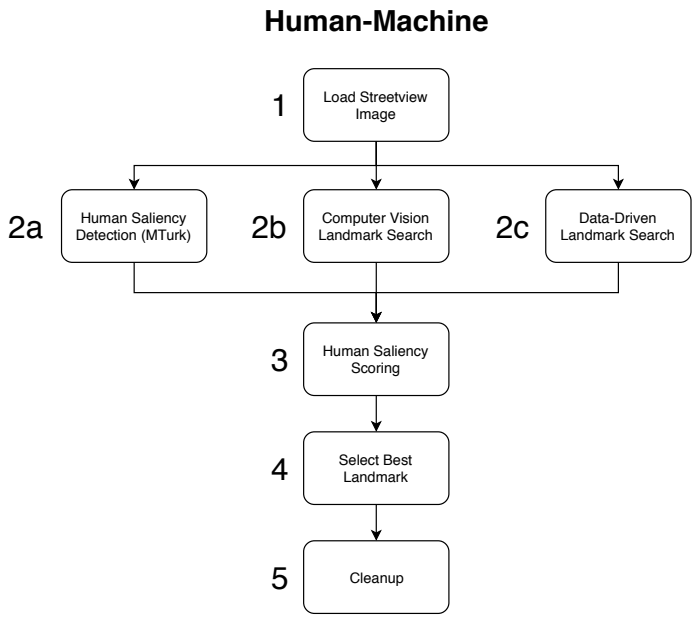


Figure 3.20: The pipeline structure of the Human-Machine pipeline.

Step 1: Load Streetview Image

Input: $X_1 = X_0 = (latitude, longitude, bearing)$

Output: $Y_1 = (latitude, longitude, bearing, [image\_urls])$

This step is implemented in the same manner as Step 1 of the Machine-Machine pipeline.

Step 2a: Human Saliency Detection

Input: $X_{2a} = Y_1 = (latitude, longitude, bearing, [image\_urls])$

Output: $Y_{2a} = (latitude, longitude, bearing, [image\_urls], [saliency\_maps])$

This step generates saliency matrices for the maneuver point, one for each of the street-level images found in Step 1. This implementation uses the crowdsourcing approach described in Section 3, and leverages human intuition about what constitutes a good landmark. The generated saliency map is therefore not specific to a single component of landmark saliency (visual, semantic or structural) but comprises the entire saliency metric.

The output of this step is a matrix of the same dimensions as the input maneuver point image; each element is a value between 0 and 255 indicating the relative saliency at that point in the image.

Step 2b: Computer Vision Landmark Search

Input: $X_{2b} = Y_1 = (latitude, longitude, bearing, [image\_urls])$

Output: $Y_{2b} = (latitude, longitude, bearing, [image\_urls], [candidate\_landmarks])$

This step is implemented in the same manner as Step 2b of the Machine-Machine pipeline.

Step 2c: Data-driven Landmark Search

Input: $X_{2c} = Y_1 = (latitude, longitude, bearing, [image\_urls])$

Output: $Y_{2c} = (latitude, longitude, bearing, [image\_urls],$
$[candidate\_landmarks], [saliency\_maps])$

This step is implemented in the same manner as Step 2c of the Machine-Machine pipeline, except that the semantic saliency gleaned from the geosocial database is not used. (In this pipeline, the human-based saliency detection serves as the entire basis

of saliency.) Rather, this step is used to generate candidate landmarks, which are correlated with the human-created saliency map in Step 3.

Step 3: Human Saliency Scoring

Input: $X_3 = Y_{2a} \cup Y_{2b} \cup Y_{2c} = (latitude, longitude, bearing, [image\_urls], [saliency\_maps], [candidate\_landmarks])$

Output: $Y_3 = (latitude, longitude, bearing, [image\_urls], [candidate\_landmarks])$

This step is implemented in the same manner as Step 3 of the Machine-Machine pipeline.

Step 4: Select Most Salient Landmark

Input: $X_4 = Y_3 = (latitude, longitude, bearing, [image\_urls], [saliency\_maps], [candidate\_landmarks])$

Output: $Y_4 = (latitude, longitude, bearing, best\_landmark)$

The candidate landmark with the highest human saliency score is the best, most salient landmark, and is the output of this step. If a tie exists, uniqueness, calculated as described in Section 3, is used as a tie-breaker. The description of this landmark will be included in navigation instructions spoken to the user.

Step 5: Cleanup

Input: $X_5 = Y_4 = (latitude, longitude, bearing, best_landmark)$

Output: $Y_5 = (best\_landmark)$

This step is implemented in the same manner as Step 5 of the Machine-Machine pipeline.

Machine-Human

Input: $X_0 = (latitude, longitude, bearing)$

Output: The most salient landmark, including a description for use in navigation instructions

Candidate selection: Saliency

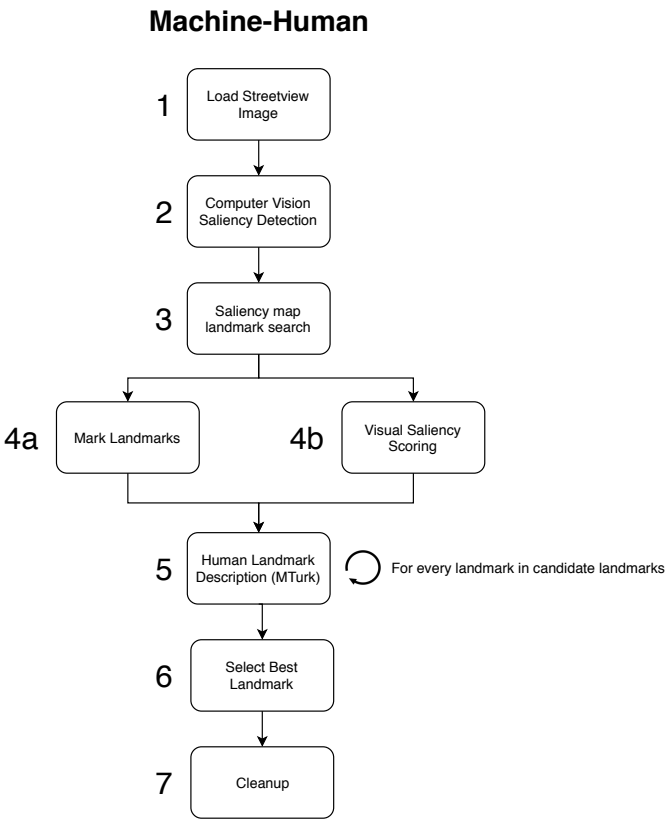


Figure 3.21: The pipeline structure of the Machine-Human pipeline.

Step 1: Load Streetview Image

Input: $X_1 = X_0 = (latitude, longitude, bearing)$

Output: $Y_1 = (latitude, longitude, bearing, [image\_urls])$

This step is implemented in the same manner as Step 1 of the Machine-Machine pipeline.

Step 2: Computer Vision Saliency Detection

Input: $X_2 = Y_0 = (latitude, longitude, bearing, [image\_urls])$

Output: $Y_2 = (latitude, longitude, bearing, [image\_urls], [saliency\_maps])$

This step is implemented in the same manner as Step 2a of the Machine-Machine pipeline.

Step 3: Find candidate landmarks within saliency map

Input: $X_{3a} = Y_2 = (latitude, longitude, bearing, [image\_urls], [saliency\_maps])$

Output: $Y_{3a} = (latitude, longitude, bearing, [image\_urls],$
$[saliency\_maps], [candidate\_landmarks])$

Using the watershed algorithm described in Section 3, we search the machine-generated saliency maps from Step 2 for salient regions, which compose the candidate landmark set.

Step 4a: Create annotated maneuver point images

Input: $X_{3a} = Y_2 = (latitude, longitude, bearing, [image\_urls],$
$[saliency\_maps], [candidate\_landmarks])$

Output: $Y_{3a} = (latitude, longitude, bearing, [image\_urls],$
$[saliency\_maps], [candidate\_landmarks], [annotated\_image\_urls])$

In order for human workers to provide written descriptions for candidate landmarks, they need to see an image of the maneuver point with the candidate landmark outlined. We choose to show workers an annotated image of the entire maneuver point, as opposed to a cropped image containing only the candidate landmark, so that workers can incorporate context into their descriptions. For example, we have observed descriptions which incorporate the landmark's surroundings, such as "one story blue house next to the oak tree" and "stop sign near the crosswalk".

For each candidate c in the set of candidate landmarks C, we generate an image which contains a 3-pixel thick red border drawn around the bounding box of c. We store these images on S3, and include the relevant URLs in the output of this step.

Step 4b: Visual Saliency Scoring

Input: $X_{3b} = Y_2 = (latitude, longitude, bearing, [image\_urls],$
$[saliency\_maps], [candidate\_landmarks], [annotated\_image\_urls])$
Output: $Y_{3b} = (latitude, longitude, bearing, [image\_urls],$
$[saliency\_maps], [candidate\_landmarks], [annotated\_image\_urls]))$

This step is implemented in the same manner as Step 3 of the Machine-Machine pipeline, except that the landmark search (watershed) component is not needed as all candidate landmarks include bounding boxes.

Step 5: Human-based Landmark Description

Input: $X_4 = Y_{3a} \cup Y_{3b} = (latitude, longitude, bearing,$
$[image\_urls],$
$[saliency\_maps], [candidate\_landmarks], [annotated\_image\_urls])$
Output: $Y_4 = (latitude, longitude, bearing, [image\_urls],$
$[saliency\_maps], [candidate\_landmarks], [annotated\_image\_urls]$

For each landmark c in the set of candidate landmarks C, we utilize the human-based description method described in Section 3 to obtain a lexical description of c. These descriptions are included in the given candidate landmark tuple in the output of this step. This step does not complete until all candidate landmarks have been processed through MTurk.

Note that it is possible for the description of a candidate landmark to fail, if workers are unable to agree upon the accuracy of a description within three attempts, or if workers agree that the landmark is invalid due to being temporary or irrelevant.

(This process of description and verification is described in Section 3.) If description fails for a candidate, it is removed from the candidate set.

Step 6: Select Most Salient Landmark

Input: $X_5 = Y_4 = (latitude, longitude, bearing, [image\_urls],$
$[saliency\_maps], [candidate\_landmarks], [annotated\_image\_urls])$

Output: $Y_5 = (latitude, longitude, bearing, best\_landmark)$

This step is implemented in the same manner as Step 4 of the Machine-Machine pipeline.

Step 7: Cleanup

Input: $X_6 = Y_5 = (latitude, longitude, bearing, best\_landmark)$

Output: $Y_6 = (best\_landmark)$

This step is implemented in the same manner as Step 5 of the Machine-Machine pipeline.

Human-Human

Input: $X_0 = (latitude, longitude, bearing)$

Output: The most salient landmark, including a description for use in navigation instructions

Candidate selection: Description

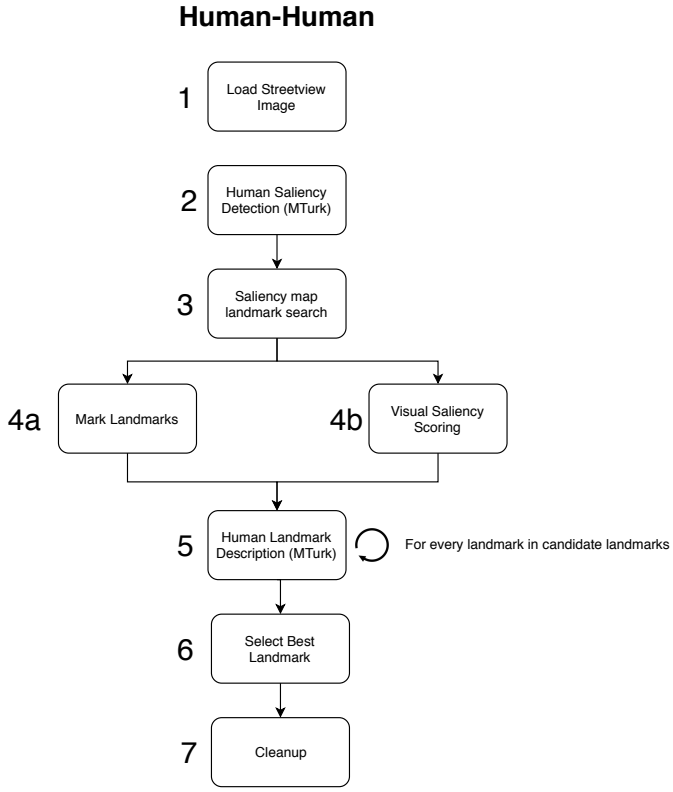


Figure 3.22: The pipeline structure of the Human-Human pipeline.

Step 1: Load Streetview Image

Input: $X_1 = X_0 = (latitude, longitude, bearing)$

Output: $Y_1 = (latitude, longitude, bearing, [image\_urls])$

This step is implemented in the same manner as Step 1 of the Machine-Machine pipeline.

Step 2: Human Saliency Detection

Input: $X_2 = Y_1 = (latitude, longitude, bearing, [image\_urls])$

Output: $Y_2 = (latitude, longitude, bearing, [image\_urls], [saliency\_maps])$

This step is implemented in the same manner as Step 2a of the Human-Machine pipeline.

Step 3: Find candidate landmarks within saliency map

Input: $X_{3a} = Y_2 = (latitude, longitude, bearing, [image\_urls], [saliency\_maps])$

Output: $Y_{3a} = (latitude, longitude, bearing, [image\_urls],$
$[saliency\_maps], [candidate\_landmarks])$

Using the watershed algorithm described in Section 3, we search the machine-generated saliency maps from Step 2 for salient regions, which compose the candidate landmark set.

Step 4a: Create annotated maneuver point images

Input: $X_{3a} = Y_2 = (latitude, longitude, bearing, [image\_urls],$
$[saliency\_maps], [candidate\_landmarks])$

Output: $Y_{3a} = (latitude, longitude, bearing, [image\_urls],$
$[saliency\_maps], [candidate\_landmarks], [annotated\_image\_urls])$

This step is implemented in the same manner as Step 3a of the Machine-Human pipeline.

Step 4b: Visual Saliency Scoring

Input: $X_{3b} = Y_2 = (latitude, longitude, bearing,$
$[image\_urls], [saliency\_maps], [candidate\_landmarks])$

Output: $Y_{3b} = (latitude, longitude, bearing, [image\_urls], [candidate\_landmarks])$

This step is implemented in the same manner as Step 3 of the Machine-Machine pipeline, except that the landmark search (watershed) component is not needed as all candidate landmarks include bounding boxes.

Step 5: Human-based Landmark Description

Input: $X_4 = Y_{3a} \cup Y_{3b} = (latitude, longitude, bearing, [image\_urls],$
$[saliency\_maps], [candidate\_landmarks], [annotated\_image\_urls])$

Output: $Y_4 = (latitude, longitude, bearing, [image\_urls],$
$[saliency\_maps], [candidate\_landmarks], [annotated\_image\_urls])$

This step is implemented in the same manner as Step 4 of the Machine-Human pipeline.

Step 6: Select Most Salient Landmark

Input: $X_5 = Y_4 = (latitude, longitude, bearing, [image\_urls],$
$[salienc_maps], [candidate\_landmarks], [annotated\_image\_urls])$

Output: $Y_5 = (latitude, longitude, bearing, best\_landmark)$

This step is implemented in the same manner as Step 4 of the Human-Machine pipeline.

Step 7: Cleanup

Input: $X_6 = Y_5 = (latitude, longitude, bearing, best\_landmark)$

Output: $Y_6 = (best\_landmark)$

This step is implemented in the same manner as Step 5 of the Machine-Machine pipeline.

RESULTS

Our aim with these analyses is to understand the effectiveness of human versus machine methodologies for landmark selection and to determine the efficacy of the overall system for improving drivers' cognitive load and performance during navigation. We analyze the Torchbearer system on two fronts: first, we examine the differences between pipelines on a performance and efficiency level, comparing execution cost, execution time and similarity between results. Second, we perform a field study with real drivers using the Torchbearer system to navigate along a route unknown to them, comparing cognitive load, driving performance and perceived task difficulty between all four pipelines and a control.

We leverage ANOVA-based analyses throughout this section to determine if pipeline has a significant effect on the variable of interest. Note that in all statistical analyses used throughout this section, requirements for normal distribution are tested by visual analysis of the Q-Q plot. Homogeneity of variance is tested via Levene's Test at a significance level of 0.05. If either of these assumptions fail, we utilize the Kruskal-Wallis analysis in place of ANOVA.

Pipeline Comparison

To evaluate the differences in efficiency, cost and solution overlap we created a test set of 400 maneuver points in San Francisco, California, using an existing dataset of geographic coordinates for all intersections in the city [41]. Maneuver points were created at random by selecting an intersection and a route leading into it; the bearing for the maneuver point was computed by measuring the angle between the two points closest to the intersection in a poly line representation of the route (see Figure 4.1).

Each maneuver point was processed through each of the four Torchbearer

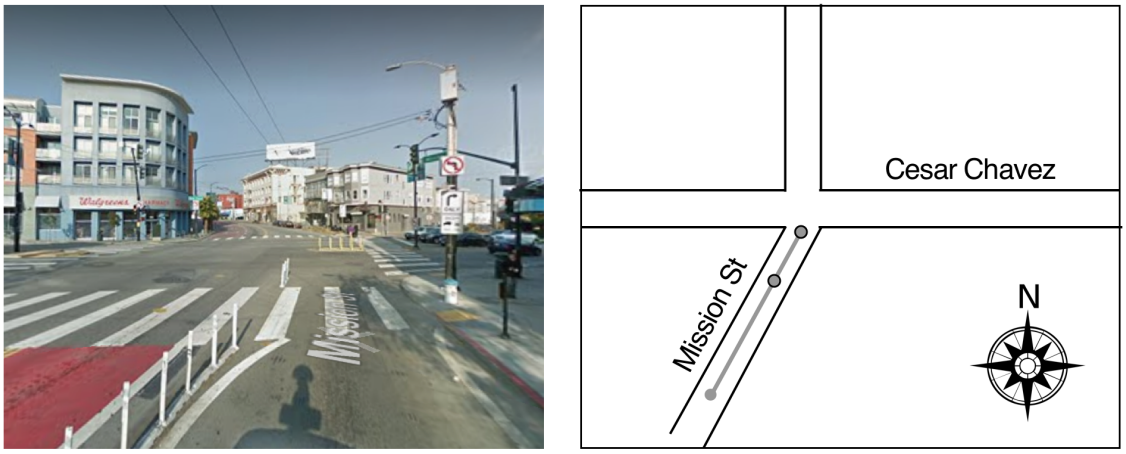


Figure 4.1: Left: The Google Streetview image of the intersection of Mission and Cesar Chavez in San Francisco, part of the SF test set. Right: A map view of this intersection. The grey line is a polyline representative of the selected route leading into the intersection. To find the bearing value for the Torchbearer maneuver point we calculate the angle w.r.t. due north between the two points outlined in black.

pipelines, resulting in a balanced result set of 1,600 pipeline executions.

Marginal Cost

Torchbearer pipelines incur monetary cost when they use MTurk to gather human input. In an effort to compare the drawbacks and benefits of each pipeline, it is important to have an understanding of the differences in expenditure.

Task workers record the cost incurred for the processing of each maneuver point as the pipeline executes in the Torchbearer database; anytime a HIT is submitted to MTurk the cost is increased by nc where n is the number of workers who will complete the HIT and c is the amount to be paid to each worker. For this experiment we paid workers \$0.05 for a saliency selection HIT, \$0.05 for a landmark description HIT and \$0.03 for a landmark verification HIT. These amounts were selected based on observational analysis of Mechanical Turk pricing for similar object-detection-related tasks; we aimed to offer above average pay for each type of HIT to avoid low pay

as a confounding variable in work quality. The marginal pipeline costs (the cost of processing an additional maneuver point) are shown in Figure 4.2

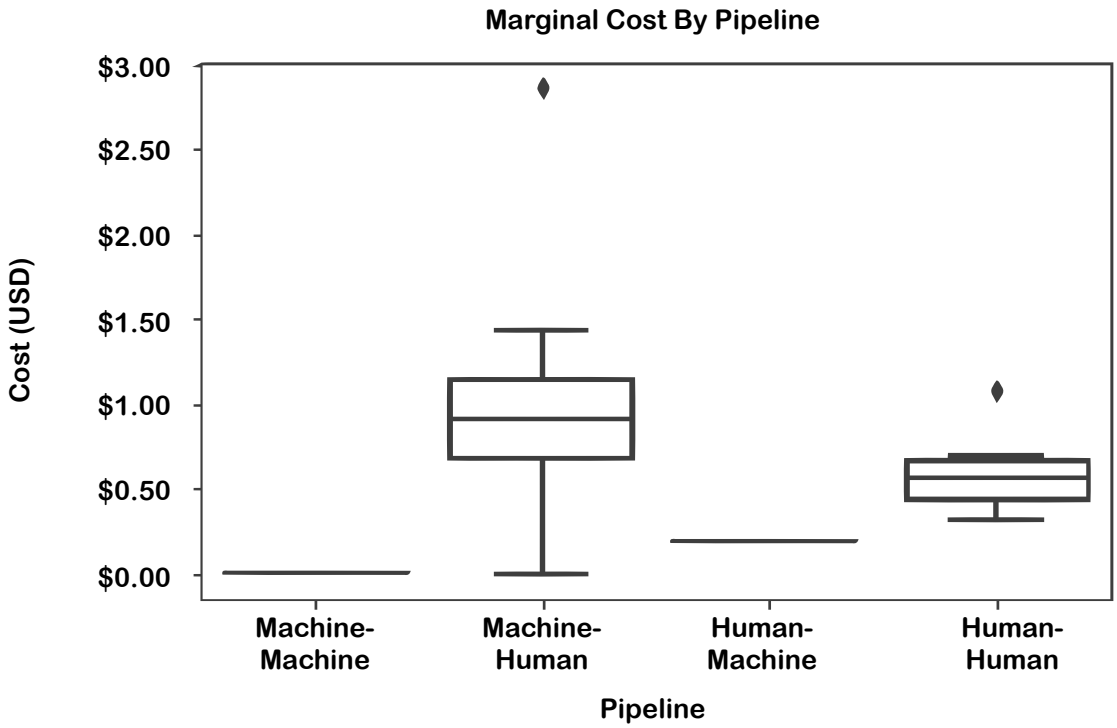


Figure 4.2: Marginal cost by pipeline

Based on the results of a one-way Analysis of Variance (ANOVA) test, we find that the mean marginal cost differs significantly by pipeline $F(3, 396) = 154.59$, $p < 0.001$. A post-hoc analysis using Tukey Honest Significant Differences (HSD) reveals that, at $p < 0.05$, the marginal cost differs significantly between all pipelines except for the Machine-Machine and Human-Machine pipelines.

The marginal cost of the Machine-Machine pipeline is extremely low—it has a mean cost per maneuver point of \$0.00004. The Machine-Machine pipeline requires no human input, therefore the cost is entirely a result of computational resource usage: this pipeline takes on average of six seconds to execute form end-to-end, as will be seen in the following section, and the price of the AWS node upon which Torchbearer

runs is $0.02 per hour.

The results for the Human-Machine pipeline are similarly deterministic—this pipeline requests a single saliency detection HIT with a fixed number of worker assignments (5 in our experiment). The Machine-Human pipeline exhibits not only the highest average cost, but also the highest variance. Both of these traits are due to the description verification component, which has the potential to repeat the entire description step, introducing non-determinism and increasing the cost of a execution significantly.

This non-determinism due to verification is also a likely explanation of the variance observed in the Human-Machine pipeline. However, variance is less than the Machine-Human pipeline, which we attribute to humans' seeming ability to select more meaningful landmarks during the saliency step than the SalNet-based saliency approach. In other words, it is possible that the machine approach to saliency sometimes selects salient regions, which do not contain an object that can be easily described, and contention is created among and between the describing workers and verification workers. This leads to more "loops" of the description step when workers don't agree, and therefore a higher execution cost.

The relatively high costs of the human-based pipeline may not render them impractical, however. Since the street-level imagery used by Torchbearer is not (currently) realtime, updated on a scale of years, a given maneuver point only needs to be processed by Torchbearer on a relatively rare frequency. Thus, if a particular pipeline proves to be expensive, but highly useful for drivers, it might be worth bearing that cost on an n-year cycle. Of course if realtime imagery is used, the Machine-Machine pipeline may be the only economically viable option.

Torchbearer is able to amortize costs by storing landmark descriptions for every maneuver point it processes. Thus, only the first request for a given maneuver

point/pipeline combination will require processing by the pipeline. Costs are amortized by the number of requests received between updates of the street-level imagery source.

Execution Time

Along with monetary cost, execution time is a cost to using a given pipeline. Using our San Francisco test set, we measure both end-to-end processing time and task-wise execution time.

End-to-End Execution Time We record the start and end timestamps for each execution; the difference between these timestamps are shown in Figure 4.3

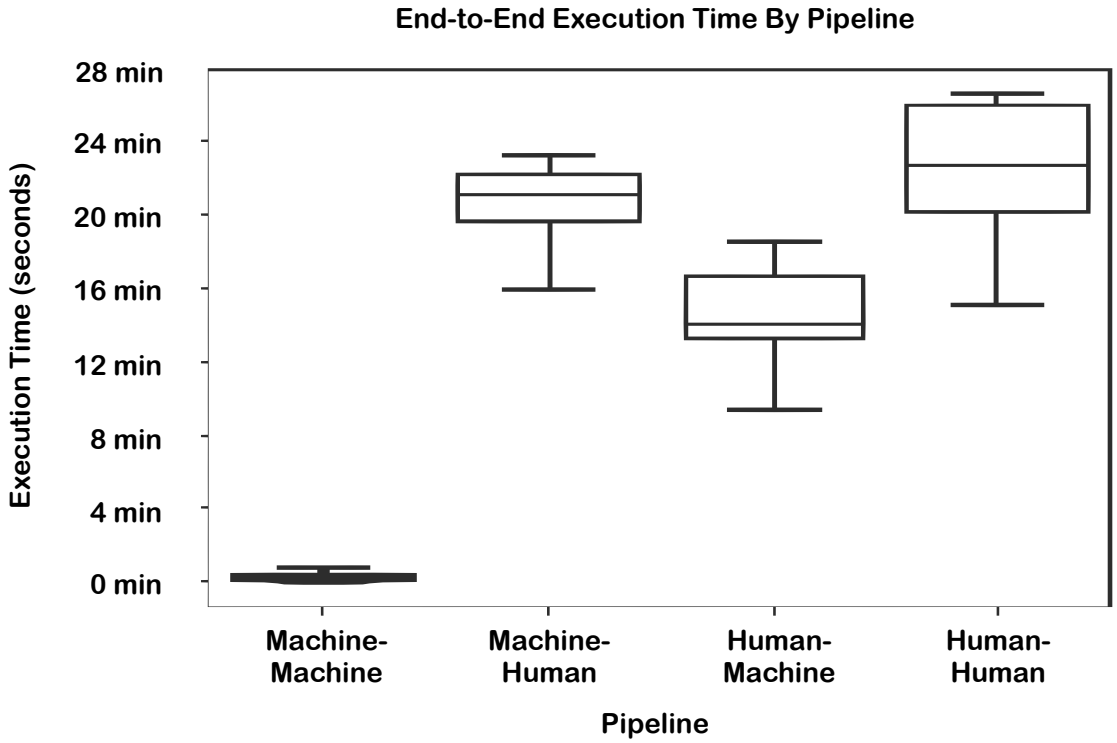


Figure 4.3: End-to-end execution time by pipeline

Based on the results of an ANOVA test, we find that the mean end-to-end differs significantly by pipeline $F(3, 396) = 117.24$, $p < 0.001$. A post-hoc analysis

using Tukey HSD reveals that, at $p < 0.05$, the end-to-end execution time differs significantly between all pipelines.

The Machine-Machine pipeline exhibits the lowest mean end-to-end execution time by an extreme margin, with very low variance. The pipelines which incorporate human input are, unsurprisingly, slower on the order of tens of minutes. These pipelines also exhibit significant variance, which is expected given the relative unpredictability of the human pipeline tasks. Likely for the same reasons we observe a higher marginal cost, we see a longer mean execution time for the Machine-Human pipeline than we do for the Human-Machine pipeline. The mean execution time for the Human-Human and Machine-Human pipelines are similar, again implying that the machine approach to saliency results in more looping, or contention, at the human description step. The Human-Human pipeline has the largest variance, due to the most reliance on human work, and also the highest time.

Based on these results, it is likely that the only pipeline capable of executing at realtime speeds is the Machine-Machine pipeline. However, there a couple of nuances to consider: first, Mechanical Turk has the potential to become faster as Torchbearer continues to build up a pool of workers. (The more workers, the more likely an already-qualified worker will be at the ready when a Torchbearer HIT is submitted.) During the SF test set simulation, 47 workers completed saliency HITs, 37 completed description HITs and 64 completed verification HITs. Over time, as the reputation of Torchbearer as a fair, well-paying requester grew, and more workers completed the qualification process, more parallelization could occur at the human worker level.

Second, as noted in regards to pipeline cost, the benefits of a realtime pipeline will not be realized unless the street-level imagery source is also realtime, which Google Streetview is not. Indeed, most landmarks are permanent fixtures in the environment, and thus a Torchbearer pipeline only needs to process a given maneuver

point whenever the street-level imagery is updated. This means that processing could be batched—an entire city's landmarks could be reevaluated at one time, and the per-maneuver point execution time of a pipeline is not relevant.

Third, for long trips it may be that a 20 to 30 minute processing time is acceptable for some landmarks. If a route consists of an hour of freeway driving, followed by several maneuver points in the destination city, the latter maneuver points can be processed while the user is on the freeway.

Execution Time By Task

For each pipeline, we evaluate the mean time required to complete each task. This gives insight into any bottlenecks that might exist in a pipeline, as well as the effectiveness of any task parallelization that was implemented. Each plot below represents an "average timeline" of execution. The length of the horizontal bar shows the average execution time for the given task, laid out in the order of execution. The plot is arranged such that tasks which execute in parallel are shown with the same start time.

Machine-Machine Figure 4.4 shows that the lion's share of processing time in the Machine-Machine pipeline is devoted to the computer vision saliency (SalNet) task. This is unsurprising, as SalNet is a computationally-intensive algorithm, consisting of convolutional filters being applied across the street-level image many times. We also observe noticeable time reductions by parallelizing the saliency detection, computer vision search and data-driven search tasks.

Machine-Human Figure 4.5 makes clear that the human landmark description task is the bottleneck in the Machine-Human pipeline, accounting for approximately 98% of total execution time. Parallelization does not provide significant benefits in

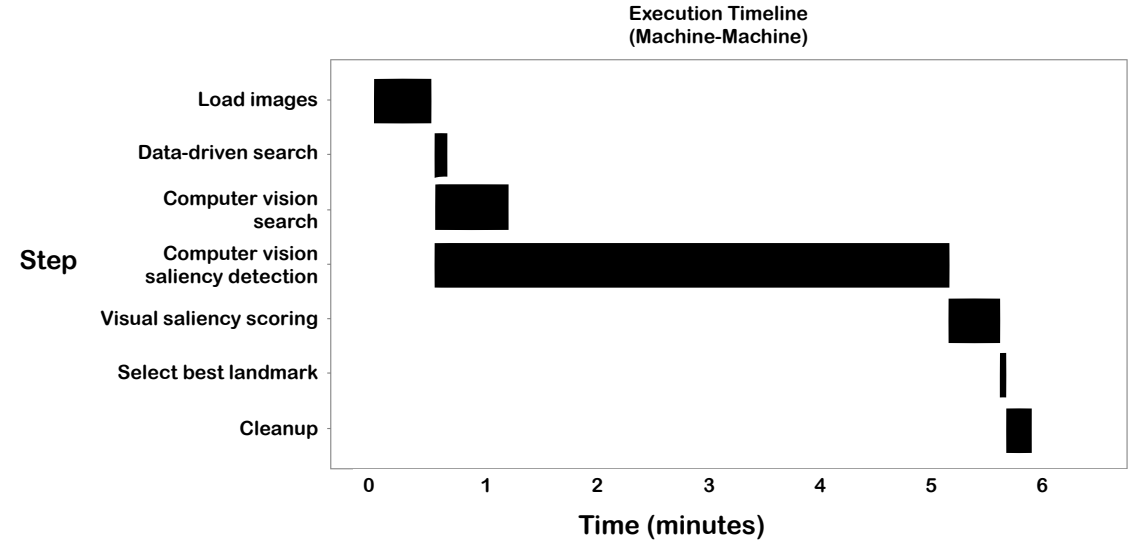


Figure 4.4: Execution time by task (Machine-Machine pipeline)

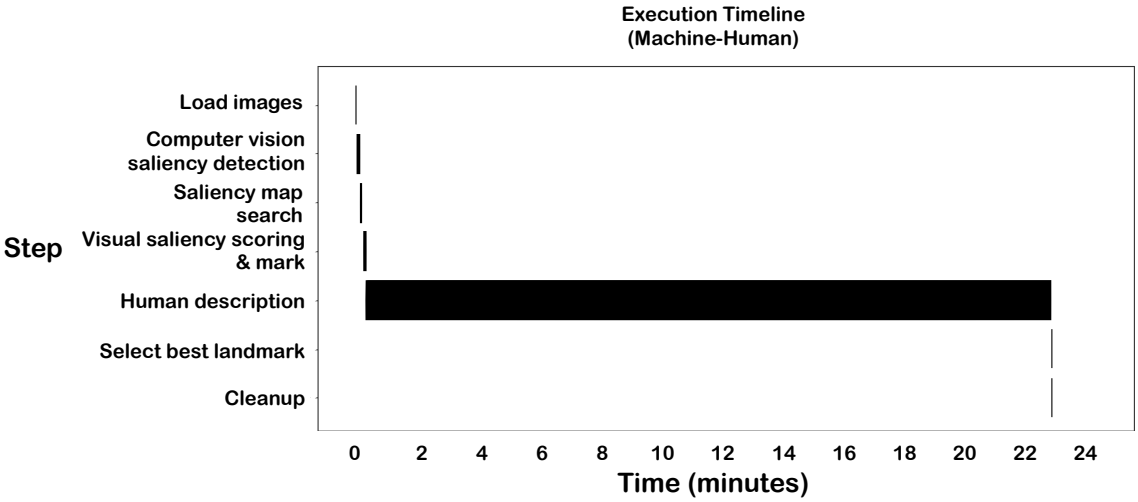


Figure 4.5: Execution time by task (Machine-Human pipeline)

terms of end-to-end execution time.

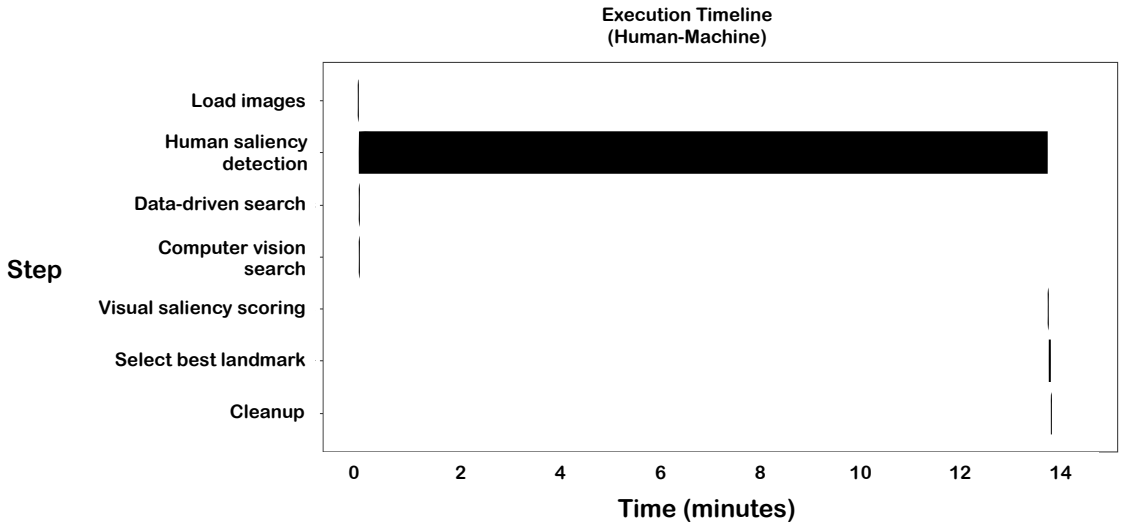


Figure 4.6: Execution time by task (Human-Machine pipeline)

Human-Machine Figure 4.6 shows that the Human-Machine pipeline suffers from a single bottleneck in the form of the human saliency task, which is to be expected as all other tasks required no human input. While some tasks are parallelized, the effect of this on the overall execution time is negligible.

Human-Human In Figure 4.7, it is clear that the two human-based tasks comprise the majority of execution time. The duration of the saliency task is somewhat longer than the description task, which we attribute to the number of workers required, as well as the difficulty of each task: the saliency task requires a sample of five workers, each of who had to make a somewhat involved decision about where to draw a box. The description task, on the other hand, requires a single worker to write a description, and three more to simply approve of what she wrote. While the description task does have the potential to "loop" if the description is rejected,

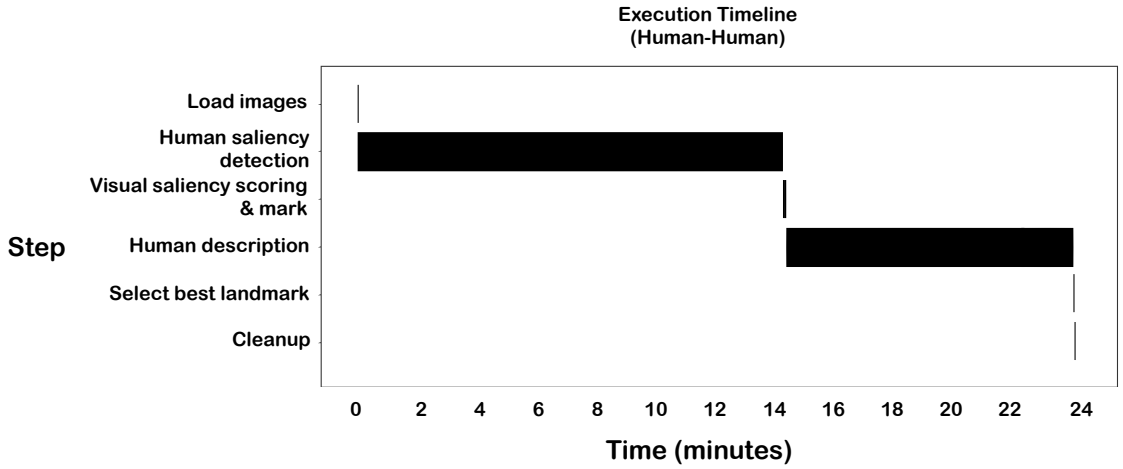


Figure 4.7: Execution time by task (Human-Human pipeline)

in the single-iteration case this task requires less workers, performing an easier task, than the saliency task does.

Selected Landmark Overlap

Every pipeline eventually selects a landmark, inclusive of a bounding box within the street-level image outlining its location. By comparing the intersection-over-union (IoU) between two landmark bounding boxes we can see to what degree the bounding boxes are selecting the same area. IoU is the ratio of area overlapped by both bounding boxes to the area encompassed by both bounding boxes; thus an IoU of 1 signifies complete agreement, or overlap, and an IoU of 0 indicates no overlap. IoU is expressed as

$$IoU = \frac{area(intersection(b_1, b_2))}{area(union(b_1, b_2))} \tag{4.1}$$

where b_1 and b_2 are the bounding boxes of two selected landmarks. Figure 4.8 shows the intersection and the union of two hypothetical bounding boxes.

For each maneuver point in the SF test set, we compute the IoU between the selected landmark returned from each pipeline. Table 4.1 shows the mean IoU between

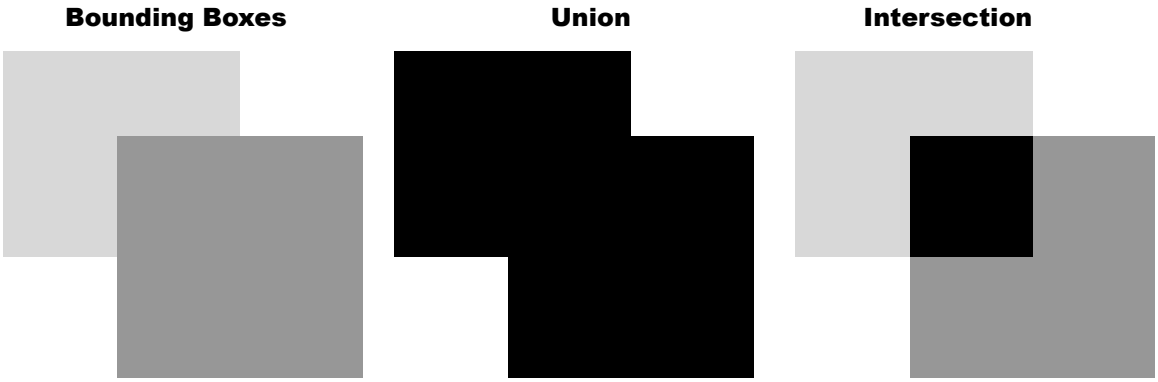


Figure 4.8: The intersection (right) and union (center) of a pair of hypothetical bounding boxes (left). The black area selection represents the area of the given metric.

Table 4.1: Mean Intersection Over Union of Selected Landmark

	Machine-Human	Human-Machine	Human-Human
Machine-Machine	0.35	0.65	0.08
Machine-Human		0.07	0.09
Human-Machine			0.05

each pair of pipelines across all maneuver points. This is essentially a measure of how likely two pipelines were to select the same landmark, or, looked at another way, the agreement between two pipelines in terms of landmark saliency.

The mean IoU between landmarks selected by the Machine-Machine and Human-Machine pipelines is the highest, at 0.65; which we largely attribute to the pipelines' identical method of selecting candidate landmarks—object detection via Faster-RCNN and FourSquare venue search. Interestingly, the methods used for determining saliency, and selecting the best landmark, vary: while Human-Machine considers only saliency as determined by human workers, Machine-Machine considers a componentized saliency score with input from SalNet and semantic saliency based on checkins and ubiquity. This implies that, at least to some degree, humans agree with our componentized saliency method in regards to what makes the best landmark.

A high IoU also exists between the Machine-Machine and Machine-Human pipelines, suggesting agreement between the salient regions generated by SalNet, which define the candidate landmark set for the Machine-Human pipeline, and the object-detection algorithm and/or the FourSquare venue search, which together build the candidate set for the Machine-Machine pipeline.

For other pipeline combinations, the mean IoU is low enough that is unlikely to be more significant than random chance. However, even though different pipelines identify different landmarks, they could still provide utility to drivers. This is examined in the following section.

Field Experiments

To evaluate the efficacy of each of our approaches for reducing driver cognitive load and improving driving performance, we conduct a Institutional Review Board approved instrumented-vehicle driving study (real driver, real vehicles, real roads)

in which subjects navigate along a route unknown to them using the Torchbearer system. (The Human Subjects Consent Form for these experiments can be found in Appendix B.) It must be noted up front that, due to constraints on time and resources, a full-scale human factors study is out of the scope of this work. While the experimental design we discuss could be applied to a larger sample and potentially yield significant results, here we use a sample size of five human subjects. Along with contributing a experimental design for future work, this small-scale study provides exploratory evidence as to the effectiveness of the Torchbearer system.

Experimental Design

We evaluate each of Torchbearer's four pipelines against a control pipeline which delivers instructions containing no landmarks. The control pipeline is comparable to a mainstream navigation application, such as Google Maps, which provides only street names and distances in its instructions.

Using a within-subjects design, five subjects drove an identical route through downtown Bozeman, Montana, using only the Torchbearer app for navigation (shown in Figure 4.9). The route was selected due to its grid (city block) layout, offering many locations for turns and a wide variety of landmarks (residential, business, and street infrastructure.) It allowed for incorporating a large number of maneuvers into the allocated 60-minute experiment time frame. This route was divided into five legs, with a different pipeline being used for navigation of each leg. After the completion of each leg, the subject was given asked to complete the NASA-TLX survey, to measure perceived task load for that leg and pipeline. A sample of landmarks used for each pipeline and route leg can be seen in Appendix A.

The subject was given no information about the route prior to the start of driving; the only information they were given throughout the drive was spoken by

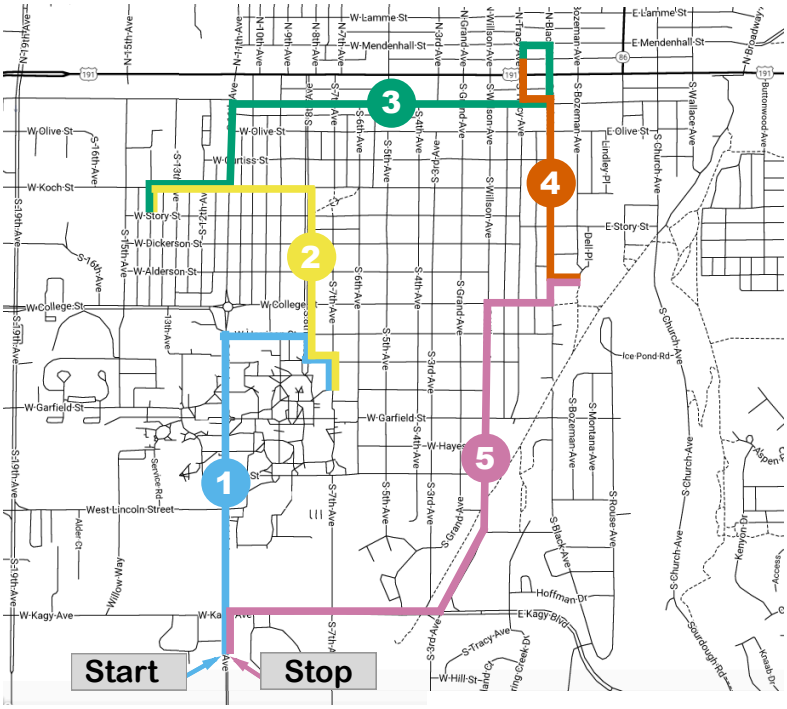


Figure 4.9: The route driven by subjects through Bozeman, Montana. Each color represents a different leg. Each leg is navigated using a different pipeline.

the Torchbearer app. Subjects were all white; two were male and three were female. All indicated they had at least some familiarity with the area of Bozeman in which the test route was located. Subjects were not compensated.

Our experiment has two sources of nuisance variability, or blocking factors: the route leg and the subject (driver). Each leg of the route is likely to have differences in road type, normal traffic levels and availability of good landmarks. Subjects vary in their driving abilities, driving style (tendency to brake hard, turn quickly, etc.), preexisting knowledge of the area in which the experiment is conducted, as well as global factors such as the time of day, weather, or traffic levels at the time the subject completed the trials. All of these characteristics can have an undesired effect on the variable of interest.

To control for these two blocking factors, we use a Latin squares design, which allows for controlling two sources of variation—subject and route leg—and isolates the treatment effect (pipeline). This is accomplished by requiring that each pipeline be analyzed on all route legs an equal number of times, and also that each subject be treated with each pipeline an equal number of times. A Latin square can be thought of as an n by n matrix, where rows represent a subject and columns represent a route leg, and n is equal to the number of pipelines, subjects and route legs (five).

The standard Latin squares design does not control for the effects of treatment order—the carryover effect of subjects always being treated with pipeline x after pipeline y—so we use a counterbalanced Latin square, which carries the additional stipulation that each pipeline must be preceded by and followed by every other pipeline an equal number of times. That is, if p_y is *preceded* by p_x for one subject, p_y must be *followed* by p_x for exactly one subject.

Because we have an odd number of treatments (four Torchbearer pipelines and one control pipeline) it is not possible to achieve the counterbalancing stipulation

within an n by n Latin square. Instead, two n x n Latin squares, with the second being a vertical reflection of the first, are required. This results in a $2n$ by n matrix, still with n route legs and n pipelines, but now requiring $2n = 10$ subjects. Because the scope of our study is limited to 5 subjects, we counterbalance the 5 by 5 to the greatest extent possible, but still have some immediate orderings which do not have the reverse represented in the square. This is a weakness of our study, and an argument in favor of future work with a larger pool of subjects, but we argue it will not threaten validity to a greater extent than the small sample size. Our Latin square design is displayed in Table 4.2.

Using the Latin square design, we arrive at the following statistical model:

$$Y_{ijk} = \overline{Y} + P_i + R_j + S_k + e_{ijk} \tag{4.2}$$

where $\overline{Y}$ is the grand mean, P_i is the pipeline (treatment) effect for a particular pipeline i, R_j is the route leg effect for a particular route leg j, S_k is the subject effect for a particular subject k, e is the error term and Y_{ijk} is an observation for a particular subject, route leg and pipeline.

Peripheral Detection Task To measure the effect of pipelines' landmark descriptions on cognitive load, we use a peripheral detection task (PDT). This secondary task consists of subjects wearing a headset, which positions an LED light approximately 15 degrees to the left of the center of vision and 2 degrees above the horizon. This light blinks at a uniform random interval between 3 and 5 seconds, for a duration of between 200 and 1000 milliseconds [35]. A button is attached to the subject's finger, which can be pressed against the steering wheel. The subject is asked to depress the button as quickly as possible whenever they see the light blink. The average delay between light blink and button depression is recorded, along with a miss rate—if the

Table 4.2: Counterbalanced Latin Squares Design

	Leg 1	Leg 2	Leg 3	Leg 4	Leg 5
Subject 1	No landmarks	Human-Human	Machine-Machine	Human-Machine	Machine-Human
Subject 2	Human-Human	Human-Machine	No landmarks	Machine-Human	Machine-Machine
Subject 3	Human-Machine	Machine-Human	Human-Human	Machine-Machine	No landmarks
Subject 4	Machine-Human	Machine-Machine	Human-Machine	No landmarks	Human-Human
Subject 5	Machine-Machine	No landmarks	Machine-Human	Human-Human	Human-Machine

subject fails to press the button within 2 seconds of a light blink, it counts as a miss. Intuitively, the more cognitive effort the subject must expend on the primary task of navigation and vehicle operation, the less effort they can put towards the PDT. Thus, a more cognitively-intensive primary task will result in a higher miss rate and longer button press delay.

The PDT must be evaluated on two levels—first the miss rate, the probability of a subject never pressing the button within 2 seconds of the LED blinking, and second the mean response time for non-missed blinks.

Using a Kruskal-Wallis evaluation based on the linear model in Equation 4.2, where Y is the PDT response time, we found no evidence supporting a difference in mean PDT response time between pipelines ($F(4, 12) = 1.38$, $p = 0.29$). We use Kruskal-Wallis in place of ANOVA because the normality assumption is violated (by visual analysis of the Q-Q plot). Figure 4.10 shows that differences in the mean are small relative to the large interquartile range.

We also found no evidence of pipeline affecting PDT miss rate ($F(4, 12) = 0.46$,

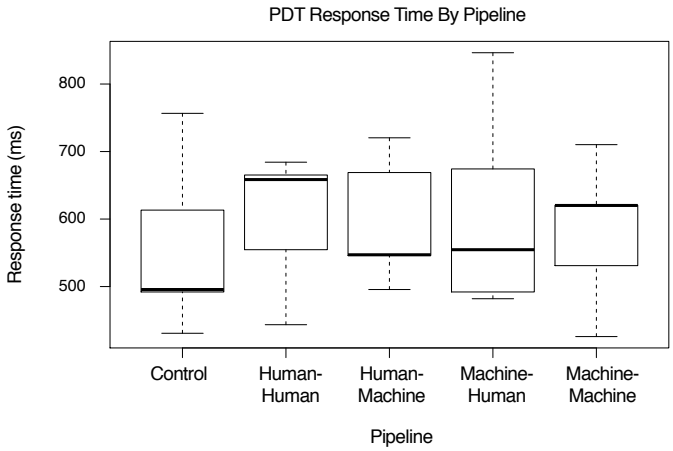


Figure 4.10: PDT response time by pipeline

$p = 0.76$), using the same analysis as for response time. (The normality assumption was violated for this data as well.) Figure 4.11 shows the distribution of miss rate by pipeline.

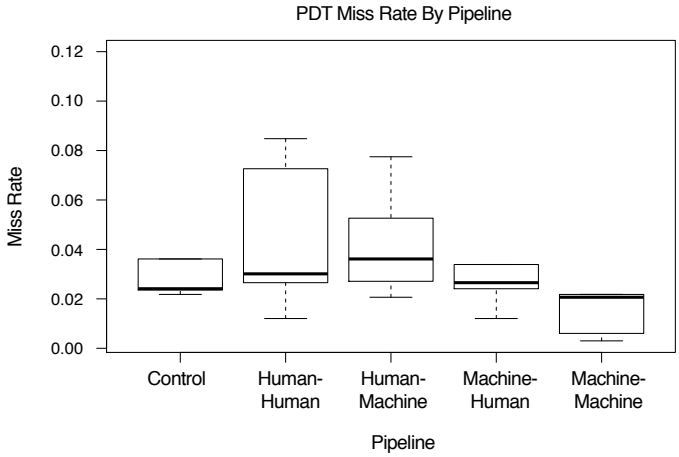


Figure 4.11: PDT miss rate by pipeline

Gravitational Force Events We also monitor for erratic, harsh, potentially dangerous driving patterns, by counting instances of high lateral (X) and longitudinal (Y) gravitational forces (G-forces). These G-force spikes, which we call *excessive force*

events can signify harsh breaking, rapid acceleration or swerving. Specifically, we count the number of times during a route leg that the vehicle experienced a G-force of greater magnitude than the thresholds set forth in Table 4.3. G-forces are measured in the X and Y directions using a Freematics ONE vehicle data logger, which includes 3-axis acceleration data and is anchored to the vehicle frame via the vehicle's OBD-II port.

Table 4.3: Gravitational Force Event Thresholds (Naturalistic Teenage Driving Study [56])

Event Type	Axis	Threshold (G)
Harsh acceleration	Y	> 0.35
Hard braking	Y	< -0.45
Right swerve	X	> 0.05
Left swerve	X	< -0.05

Using the same analysis as for the PDT metrics, we found no evidence that pipeline affects the number of excessive force events occurring during a drive ($F(4, 12) = 1.44$, $p = 0.28$). See Figure 4.12.

Surveys Lastly, we survey subjects using the NASA-TLX survey [27] and our own Likert-scale survey to analyze perceived task difficulty, as well as perceived landmark goodness, navigation confidence, and navigation difficulty between pipelines. Both surveys were administered for each pipeline, immediately following the completion of each leg.

The NASA-TLX survey consists of six sub-scales, which when combined aim to measure the total workload induced by the task—in this case navigating a route leg from start to finish using a particular pipeline for navigation. The scales are ordinal,

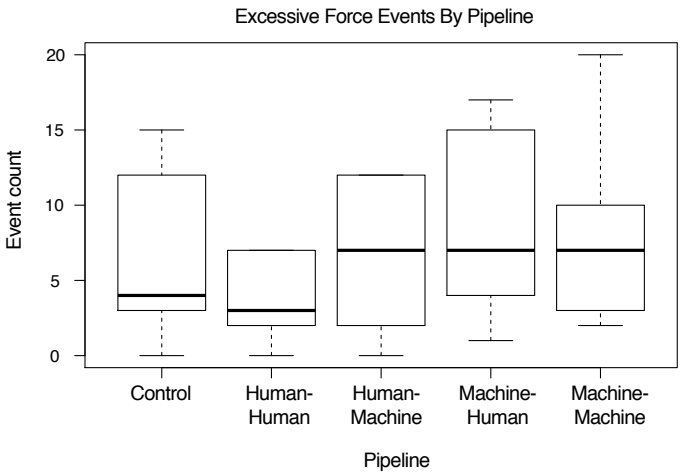


Figure 4.12: Gravitational force events by pipeline

with 20 levels ranging from *very low* to very high. (See Appendix C for a full copy of the NASA-TLX survey.) The sub-scales are *mental demand*, which measures the mental and perceptual acuity required to complete the task; *physical demand*, which gauges how strenuous the task was; *temporal demand*, which measures perceived time pressure or rush to complete the task; *overall performance*, which indicates the subject's opinion of how successful she was at completing the task; *effort*, a combined measure of mental and physical exertion; and *frustration level*, how annoyed and irritated the subject felt during the task [27]. **It is very important to note that the Performance sub-scale considers level 0 to equate to total success and level 20 to total failure, the opposite of what one might expect.**

We evaluate each sub-scale independently, so that individual effects can be parsed out. Figure 4.13 shows the score distributions by pipeline, across each sub-scale. Because the scales are ordinal, we use the non-parametric Kruskal-Wallis analysis of variance to test for differences between pipelines. Table 4.4 lists the results across each sub-scale.

We found no evidence to suggest that pipeline affects any of the NASA-TLX

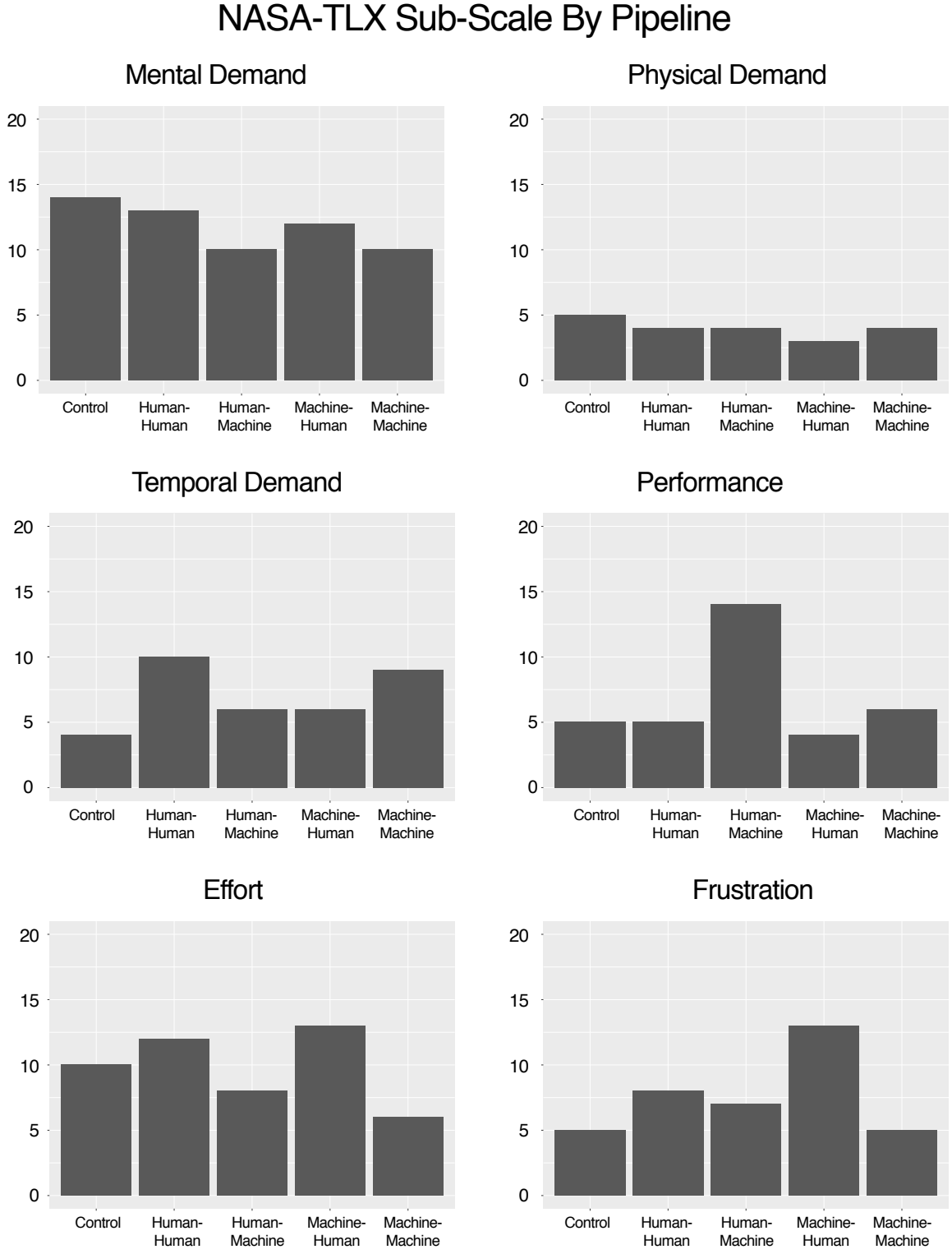


Figure 4.13: NASA-TLX scores by sub-scale

Table 4.4: Kruskal-Wallis analysis of variance by pipeline for NASA-TLX survey

Sub-Scale	χ^2	p
Mental Demand	$\chi^2(4) = 2.53$	0.64
Physical Demand	$\chi^2(4) = 0.25$	0.99
Temporal Demand	$\chi^2(4) = 1.01$	0.91
Perceived Performance	$\chi^2(4) = 1.99$	0.74
Effort	$\chi^2(4) = 0.92$	0.92
Frustration	$\chi^2(4) = 3.59$	0.46

sub-scales.

In addition to the NASA-TLX survey, after each route leg we administered a three-question Likert scale survey addressing the quality of landmarks selected by the pipeline and the confidence subjects felt at navigational decision points. Each question is a statement with which the subject indicates their agreement, selecting from *strongly agree*, *agree*, *not sure*, *disagree*, and *strongly disagree*. The statements are as follows:

The landmarks I was told about helped me find turns.

I knew what each landmark was going to look like when I heard its description.

I felt confident in where to perform each maneuver (turn) on the route.

In Figure 4.14 we show the distribution of agreement with each statement by pipeline. Each color represents a different score, with the lightest equating to "strongly disagree" and the darkest to "strongly agree". The more width a color occupies, the more subjects gave that answer as their response.

In Table 4.5 we analyze these results using a Kruskal-Wallis test to determine if there are significant differences in the distribution of answers between pipelines. For each survey question, we rejected the null hypothesis at significance level of 0.10; there is evidence that pipeline does affect participant responses to each of these questions.

Table 4.5: Kruskal-Wallis analysis of variance by pipeline for landmark survey

Question	χ^2	p (* denotes significance at the 0.1 level)
Landmarks helped find turns	$\chi^2(4) = 8.11$	0.08^*
Landmarks could be visualized from descriptions	$\chi^2(4) = 10.64$	0.03^*
Subject was confident at decision points	$\chi^2(4) = 13.83$	0.01^*

In order to determine which pipelines are significantly different from others in terms of their affect on each survey question, we use Dunn's test with a Bonferroni adjustment for post-hoc analysis. Dunn's is a pairwise comparison test which, for each combination of pipelines (a, b), tests the null hypothesis that the probability of drawing a larger value from a than from b is 0.5. The alternative hypothesis is that one group stochastically dominates another: the chances of sampling a larger value form that group is greater than 0.5. The Bonferroni adjustment adjusts p-values to account for having done multiple comparisons.

For the "confidence at decision points" metric, we find that Machine-Machine pipeline is significantly more likely to have a higher (more agreeable) score than the control pipeline ($Z = -2.78$, $p = 0.05$) as well as the Human-Machine pipeline ($Z = -3.21$, $p = 0.01$). For the "landmarks helped find turns" metric, we find

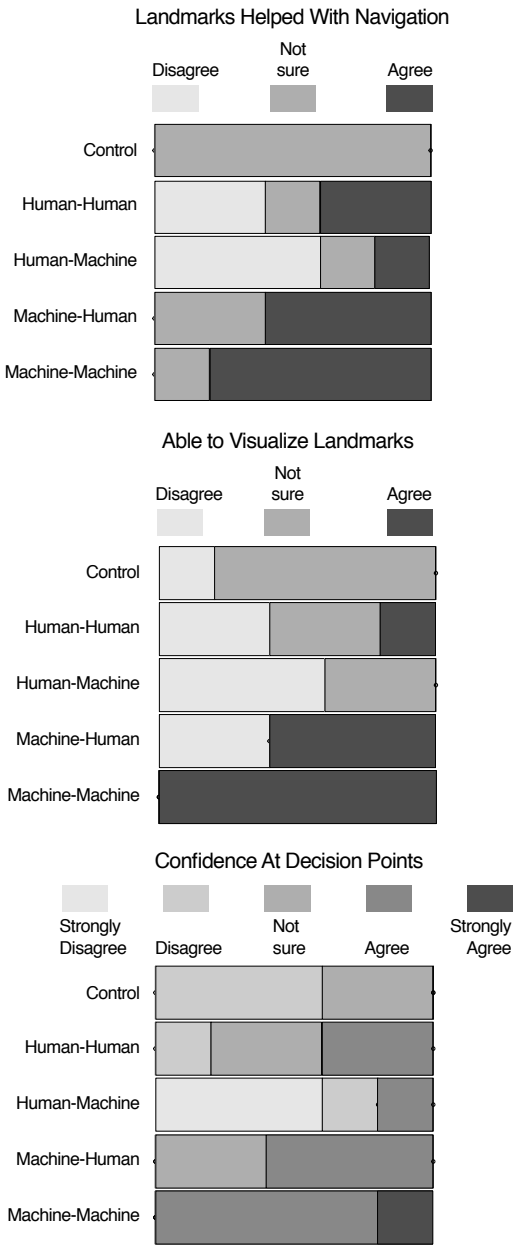


Figure 4.14: Landmark effectiveness survey scores

no evidence of significant differences in distribution between two specific pipelines. For the "landmark descriptions" metric we we find that Machine-Machine pipeline is significantly more likely to have a higher (more agreeable) score than the human-machine pipeline ($Z = -3.02$, $p = 0.02$).

Discussion

Contrary to existing literature, we did not find the inclusion of landmark descriptions in navigation instructions to have a significant effect on drivers' cognitive load, erratic driving behavior, or perceived task load. We did find that instructions inclusive of landmark descriptions generated entirely by machine (machine-machine pipeline) lead to increased driver confidence at decision points as compared to navigation instructions which included only street name and distances (control pipeline). This finding is in line with participants' subjective written comments, which indicated that including stop lights and stop signs in instructions was helpful.

Without a larger study, it is not possible to definitely say whether or not any of Torchbearer's pipelines were effective in terms of reducing cognitive load, harsh G-force events or perceived task load. While we found no evidence of such effects in our small field study, a larger study, preferably consisting of 30 participants, would offer more definitive insight.

Threats to Validity

As alluded to previously, the principal threat to validity is the extremely small sample size employed in our field experiments. However, even within this small-scale study there are potential biases: first, study participants had relatively high familiarity with area of the route, given that all were residents of Bozeman, Montana. If landmark descriptions are more helpful in terms of cognitive load, erratic driving

reduction or reduced task load in areas drivers are unfamiliar with, we would significantly underestimate the effect.

Due to time constraints, and the instrumented-vehicle experiment as opposed to a simulated one, each leg of the test route did not include a large number of maneuver points. Additionally, there was little variation in terms of road and environment type (surface versus highway, urban versus rural). A simulator-based experiment could allow for efficiently varying the driving environment.

Subjects may have been predisposed to "like" the concept of including landmarks in navigation instructions, even if there was no observable effect in terms of workload or driving behavior. This could potentially bias the results of the "confidence at maneuver points" survey—if subjects felt like they were "supposed" to like landmarks, they may have been inclined to indicate an increased sense of confidence.

CONCLUSION

We proposed Torchbearer, a system that uses multiple pipeline-based approaches to automatically generate landmark descriptions for use in navigation instructions. Each pipeline leveraged a different combination of human, crowd-sourced input and algorithmic approaches, including object detection, deep saliency detection and geosocial data mining. Together with a mobile application, each of these pipelines can be used to provide spoken turn-by-turn driving directions, inclusive of landmark descriptions.

While the goal of Torchbearer was to reduce cognitive load, erratic driving behavior and perceived workload for drivers, our field study did not find evidence of any significant effect on these metrics between Torchbearer pipelines and a street name only control pipeline. We suspect that a larger study is needed, with better controls for prior route knowledge, to accurately determine if such an effect exists.

The primary point for future work centers around additional field evaluation, with more subjects, and a driving simulator to analyze different road types and environments. Additionally, experiments should be undertaken regarding landmark location, including the efficacy of including landmarks along the leg of a route to indicate to a driver that she is on the correct route. The object detection algorithm should be trained to recognize additional types of road infrastructure, such as crosswalks.

Future Work

A count-based approach should be investigated, where the edge between two maneuver points is analyzed for recurring salient landmarks of the same type, such as stop lights, and an instruction of the form "turn left onto ¡street¿ at the ¡nth

stop light" presented. The counter-approach could also be investigated, where the recurrence along an edge of a landmark type counts against the saliency, such that a landmark would only be chosen if the driver will not encounter one of that type until the maneuver point.

Further insight into semantic saliency can be gained by additional mining of geosocial data: while we currently consider overall check-in data, a data source such as Facebook or Instagram could be used to determine the relevance of a landmark to an individual driver. For example, if a Walgreens pharmacy is a candidate landmark, its saliency score could account for the fact that the driver has visited a Walgreens store on n previous occasions.

While Torchbearer currently uses fixed distances from maneuver points for locating landmarks, it is possible that speed of travel affects the optimal position of landmarks. Further study should be done to determine if increasing landmark distance from intersection as speed increases is beneficial.

In an attempt to improve our ability to analyze the effects of pipeline on cognitive load, an arithmetic task can be incorporated into the field experiment, where a subject is asked to solve math problems during the drive. This consumes more of the subject's available cognitive demand, leaving less to put towards the PDT; this can help yield significant effects in PDT metrics by making difference between pipeline more apparent. Additionally, other metrics may provide insight into the potential benefits of Torchbearer, such as the total time taken to drive a leg, the amount of time the subject's eyes leave the road and the subject's willingness to pay for the technology provided by a given Torchbearer pipeline.

Much of these additional areas of investigation will alter only the portion of the Torchbearer system which selects the best landmark—existing methods for finding and describing landmarks will be used. In this way, Torchbearer has provided a

robust base against which future landmark-based navigation systems can be built.

REFERENCES CITED

[1] Image segmentation with watershed algorithm, Oct 2017.

[2] Agarwal, P., Burgard, W., and Spinello, L. Metric localization using google street view. In *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on* (2015), IEEE, pp. 3111–3118.

[3] Barnes, R., Lehman, C., and Mulla, D. Priority-flood: An optimal depression-filling and watershed-labeling algorithm for digital elevation models. *Computers & Geosciences 62* (2014), 117–127.

[4] Baroni, M., Dinu, G., and Kruszewski, G. Don't count, predict! a systematic comparison of context-counting vs. context-predicting semantic vectors. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)* (2014), vol. 1, pp. 238–247.

[5] Bayly, M., Young, K. L., and Regan, M. A. 12 sources of distraction inside the vehicle and their effects on driving performance. *Driver distraction: Theory, effects, and mitigation* (2008), 191.

[6] Beeharee, A. K., and Steed, A. A natural wayfinding exploiting photos in pedestrian navigation systems. In *Proceedings of the 8th conference on Human-computer interaction with mobile devices and services* (2006), ACM, pp. 81–88.

[7] Birrell, S. A., and Young, M. S. The impact of smart driving aids on driving performance and driver distraction. *Transportation research part F: traffic psychology and behaviour 14*, 6 (2011), 484–493.

[8] Burnett, G. turn right at the traffic lights: The requirement for landmarks in vehicle navigation systems. *The Journal of Navigation 53*, 3 (2000), 499–510.

[9] Burnett, G. E., and Joyner, S. An assessment of moving map and symbol-based route guidance systems. *Ergonomics and safety of intelligent driver interfaces* (1997), 115–137.

[10] Caduff, D., and Timpf, S. On the assessment of landmark salience for human navigation. *Cognitive processing 9*, 4 (2008), 249–267.

[11] Choudhary, P., and Velaga, N. R. Modelling driver distraction effects due to mobile phone use on reaction time. *Transportation Research Part C: Emerging Technologies 77* (2017), 351 – 365.

[12] Corporation, M. Visual object tagging tool (vott). `https://github.com/Microsoft/VoTT`, 2018.

[13] Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09* (2009).

[14] Edquist, J., Horberry, T., Hosking, S., and Johnston, I. Effects of advertising billboards during simulated driving. *Applied ergonomics 42*, 4 (2011), 619–626.

[15] Elias, B., and Brenner, C. Automatic generation and application of landmarks in navigation data sets. In *Developments in spatial data handling*. Springer, 2005, pp. 469–480.

[16] Facebook. React native.

[17] Fingas, J. Google maps uses landmarks to provide natural-sounding directions, Apr 2018.

[18] for Statistics, N. C., and Analysis. 2016 fatal motor vehicle crashes: Overview. Report DOT HS 812 456, National Highway Traffic Safety Administration, 2017.

[19] for Statistics, N. C., and Analysis. Distracted driving 2016. Report DOT HS 812 517, National Highway Traffic Safety Administration, 2017.

[20] Goldberg, Y., and Levy, O. word2vec explained: Deriving mikolov et al.'s negative-sampling word-embedding method. *arXiv preprint arXiv:1402.3722* (2014).

[21] Golledge, R. G. Human wayfinding and cognitive maps. In *The Colonization of Unfamiliar Landscapes*. Routledge, 2003, pp. 49–54.

[22] Google. Google news corpus word2vec model.

[23] Harbulk, J. L., and Noy, I. Y. The impact of cognitive distraction on driver visual behavior and vehicle control. Report 13889 E, Ergonomics Division, Road Safety Directorate and Vehicle Regulation Directorate, 2002.

[24] Harel, J., Koch, C., and Perona, P. Graph-based visual saliency. In *Advances in neural information processing systems* (2007), pp. 545–552.

[25] Harms, L., and Patten, C. Peripheral detection as a measure of driver distraction. a study of memory-based versus system-based navigation in a built-up area. *Transportation Research Part F: Traffic Psychology and Behaviour 6*, 1 (2003), 23–36.

[26] Harris, Z. S. Distributional structure. *Word 10*, 2-3 (1954), 146–162.

[27] Hart, S. G., and Staveland, L. E. Development of nasa-tlx (task load index): Results of empirical and theoretical research. In *Advances in psychology*, vol. 52. Elsevier, 1988, pp. 139–183.

[28] Hile, H., Vedantham, R., Cuellar, G., Liu, A., Gelfand, N., Grzeszczuk, R., and Borriello, G. Landmark-based pedestrian navigation from collections of geotagged photos. In *Proceedings of the 7th international conference on mobile and ubiquitous multimedia* (2008), ACM, pp. 145–152.

[29] Klippel, A., and Winter, S. Structural salience of landmarks for route directions. In *International Conference on Spatial Information Theory* (2005), Springer, pp. 347–362.

[30] Kulkarni, A. P., Can, M., and Hartmann, B. Turkomatic: automatic recursive task and workflow design for mechanical turk. In *CHI'11 Extended Abstracts on Human Factors in Computing Systems* (2011), ACM, pp. 2053–2058.

[31] L. Reyes, M., and Lee, J. The influence of ivis distractions on tactical and control levels of driving performance.

[32] Lee, P. U., Klippel, A., and Tappe, H. The effect of motion in graphical user interfaces. In *International Symposium on Smart Graphics* (2003), Springer, pp. 12–21.

[33] Leshed, G., Velden, T., Rieger, O., Kot, B., and Sengers, P. In-car gps navigation: engagement with and disengagement from the environment. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (2008), ACM, pp. 1675–1684.

[34] Lovelace, K. L., Hegarty, M., and Montello, D. R. Elements of good route directions in familiar and unfamiliar environments. In *International conference on spatial information theory* (1999), Springer, pp. 65–82.

[35] Martens, M., and Van Winsum, W. Measuring distraction: the peripheral detection task. *TNO Human Factors, Soesterberg, Netherlands* (2000).

[36] May, A. J., and Ross, T. Presence and quality of navigational landmarks: effect on driver performance and implications for design. *Human factors 48*, 2 (2006), 346–361.

[37] May, A. J., Ross, T., and Bayer, S. H. Incorporating landmarks in driver navigation system design: An overview of results from the regional project. *The Journal of Navigation 58*, 1 (2005), 47–65.

[38] May, A. J., Ross, T., and Bayer, S. H. Incorporating landmarks in driver navigation system design: An overview of results from the regional project. *Journal of Navigation 58*, 1 (2005), 4765.

[39] Mikolov, T., Chen, K., Corrado, G., and Dean, J. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781* (2013).

[40] Neubeck, A., and Van Gool, L. Efficient non-maximum suppression. In *Pattern Recognition, 2006. ICPR 2006. 18th International Conference on* (2006), vol. 3, IEEE, pp. 850–855.

[41] of San Francisco, C. Street intersections. ://data.sfgov.org/Geographic-Locations-and-Boundaries/Street-Intersections/ctsg-7znq/data.

[42] Otsu, N. A threshold selection method from gray-level histograms. *IEEE transactions on systems, man, and cybernetics 9*, 1 (1979), 62–66.

[43] Owkes, M., and Desjardins, O. Driver distraction: A review of the literature. *Journal of Computational Physics 270*, 1 (Aug. 2014), 587–612.

[44] Pan, J., Sayrol, E., Giro-i Nieto, X., McGuinness, K., and O'Connor, N. E. Shallow and deep convolutional networks for saliency prediction. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (June 2016).

[45] Pettitt, M., Burnett, G. E., and Stevens, A. Defining driver distraction. In *12th World Congress on Intelligent Transport SystemsITS AmericaITS JapanERTICO* (2005).

[46] Quesnot, T. Linked landmark data: Toward the automatic detection of landmarks on the web of data. *Advancing Geographic Information Science: The Past and Next Twenty Years* (2012), 227.

[47] Quesnot, T., and Roche, S. Measure of landmark semantic salience through geosocial data streams. *ISPRS International Journal of Geo-Information 4*, 1 (2014), 1–31.

[48] Raubal, M., and Winter, S. Enriching wayfinding instructions with local landmarks. In *International conference on geographic information science* (2002), Springer, pp. 243–259.

[49] Regan, M. A., Hallett, C., and Gordon, C. P. Driver distraction and driver inattention: Definition, relationship and taxonomy. *Accident Analysis & Prevention 43*, 5 (2011), 1771–1781.

[50] Ren, S., He, K., Girshick, R., and Sun, J. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems* (2015), pp. 91–99.

[51] Roerdink, J. B., and Meijster, A. The watershed transform: Definitions, algorithms and parallelization strategies. *Fundamenta informaticae 41*, 1, 2 (2000), 187–228.

[52] Röser, F., Hamburger, K., Krumnack, A., and Knauff, M. The structural salience of landmarks: results from an on-line study and a virtual environment experiment. *Journal of Spatial Science 57*, 1 (2012), 37–50.

[53] Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., et al. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision 115*, 3 (2015), 211–252.

[54] Salient. *The Oxford English Dictionary*. Oxford University Press, 2018.

[55] schneider, a. Mazal tov! google and waze officially tie the knot, Oct 2014.

[56] Simons-Morton, B. G., Zhang, Z., Jackson, J. C., and Albert, P. S. Do elevated gravitational-force events while driving predict crashes and near crashes? *American Journal of Epidemiology 175*, 10 (2012), 1075–1079.

[57] Simonyan, K., and Zisserman, A. Very deep convolutional networks for large-scale image recognition. *CoRR abs/1409.1556* (2014).

[58] Smith, A., and Page, D. U.s. smartphone use in 2015. Report 202.419.4372, Pew Research Center, 2015.

[59] Sorrows, M. E., and Hirtle, S. C. The nature of landmarks for real and electronic spaces. In *International Conference on Spatial Information Theory* (1999), Springer, pp. 37–50.

[60] Tsimhoni, O., Smith, D., and Green, P. Address entry while driving: Speech recognition versus a touch-screen keyboard. *Human factors 46*, 4 (2004), 600–610.

[61] Walker, J., Alicandri, E., Sedney, C., and Roberts, K. In-vehicle navigation devices: Effects on the safety of driver performance. In *Vehicle Navigation and Information Systems Conference, 1991* (1991), vol. 2, IEEE, pp. 499–525.

[62] Wenig, N., Wenig, D., Ernst, S., Malaka, R., Hecht, B., and Schöning, J. Pharos: improving navigation instructions on smartwatches by including global landmarks. In *Proceedings of the 19th International Conference on Human-Computer Interaction with Mobile Devices and Services* (2017), ACM, p. 7.

[63] Zaidel, D. M., and Noy, Y. I. Automatic versus interactive vehicle navigation aids. *Ergonomics and safety of intelligent driver interfaces* (1997), 287–307.

APPENDICES

APPENDIX A

FIELD EXPERIMENT ROUTE AND LANDMARKS

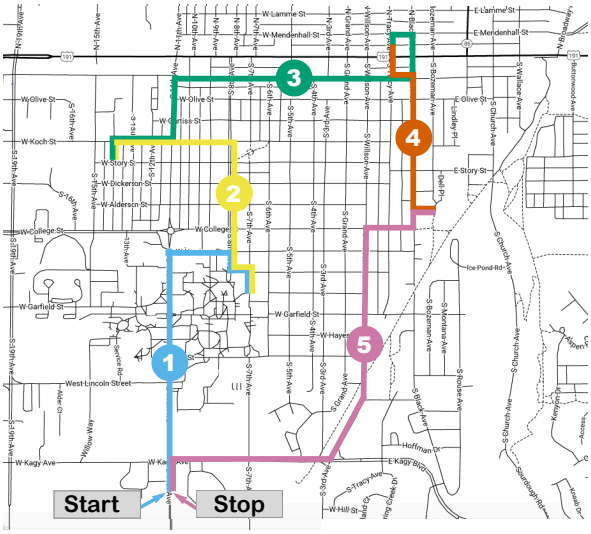


Figure A.1: The test route driven by subjects in Bozeman, Montana. Subject drive each leg using a different pipeline for navigation.

Table A.1: Leg 1: Instructions and Landmarks By Pipeline

	Machine-Machine	Machine-Human	Human-Machine	Human-Human
Turn left onto West College Street	before The Daily coffee shop	at the red and white yield sign	before The Daily coffee shop	at the round about
Turn right onto South 8th Avenue	before the Loaf n' Jug gas station		at the stop sign	at the stop sign
Continue left onto West Harrison Street	at the Hapner Hall college residence hall	at the stop sign	at the Jake Jabs College of Business and Entrepreneur-ship college hall	at the brick building with windows
Turn right onto South 7th Avenue		at the crosswalk sign		at the crosswalk
You have arrived at your destination	Hannon Dining hall college dining hall	crosswalk	brick building	brick building with windows

APPENDIX B

HUMAN SUBJECTS CONSENT FORM

SUBJECT CONSENT FORM FOR PARTICIPATION IN HUMAN RESEARCH AT MONTANA STATE UNIVERSITY

Using Landmarks To Provide Better Driving Directions

You are being asked to participate in a driving study. This study may help us obtain a better understanding of which types of navigation instructions are easiest for drivers to follow. You were identified as a potential subject because you 1) have a valid driver license, 2) have minimum motor vehicle liability insurance as required under Montana law and 3) have access to a vehicle.

Procedures Involved
Participation is voluntary and you can choose to not answer any questions you do not want to answer and/or you can stop at anytime. If you are a student, participation or non-participation will not affect your grade or class standing. If you agree to participate you will be asked to:

- Drive your own vehicle on streets in Bozeman, following driving directions spoken to you by a computerized voice on a mobile phone. These directions will tell you where and when to turn, similar to how Google Maps or Apple Maps provides spoken driving directions. You will not know anything about the route before you begin driving.
- Wear a headset which has an LED light visible only in your peripherals, and a button on your finger which you can press against the steering wheel. The light will blink at random intervals as you drive. Each time the light blinks, you will be asked to press the button.
- Complete a short survey about your experience using the system.
- The entire study will take about 1 hour.

Risks
You will be subject to the normal risks involved in everyday driving. The task of watching for a blinking light and pressing a button might be distracting, which could cause you to pay less attention to operating the vehicle.

Benefits
The study is of no benefit to you.

Alternatives available
There is no affect on you if you decide not to participate in this study.

Source of Funding
N/A

Cost to Subject
None

Confidentiality

Your personal information will be kept private and secure. Any results which are published or made publicly available will not include any personally identifiable information. All data which can be linked to you will be stored on a password-protected computer or stored on an encrypted, restricted-access cloud storage provider.

If you sustain any bodily harm during this study, you will be referred to a trained caregiver and emergency medical care will be summoned if needed. However, there is no compensation available from MSU for injury.

There is no compensation available from MSU related to motor vehicle liability, or for damages to your vehicle or personal property.

Should you have any questions about this research, please contact Fred Vollmer at (360) 927-5124 or [fredric.vollmer@msu.montana.edu]. If you have additional questions about the rights of human subjects please contact the Chair of the Institutional Review Board, Mark Quinn, (406) 994-4707 [mquinn@montana.edu].

AUTHORIZATION: I have read the above and understand the discomforts, inconvenience and risk of this study. I, _____ (name of subject), agree to participate in this research. I understand that I may later refuse to participate and that I may withdraw from the study at any time. I have received a copy of this consent form for my own records.

Signed: _____

Investigator: _____

Date: _____

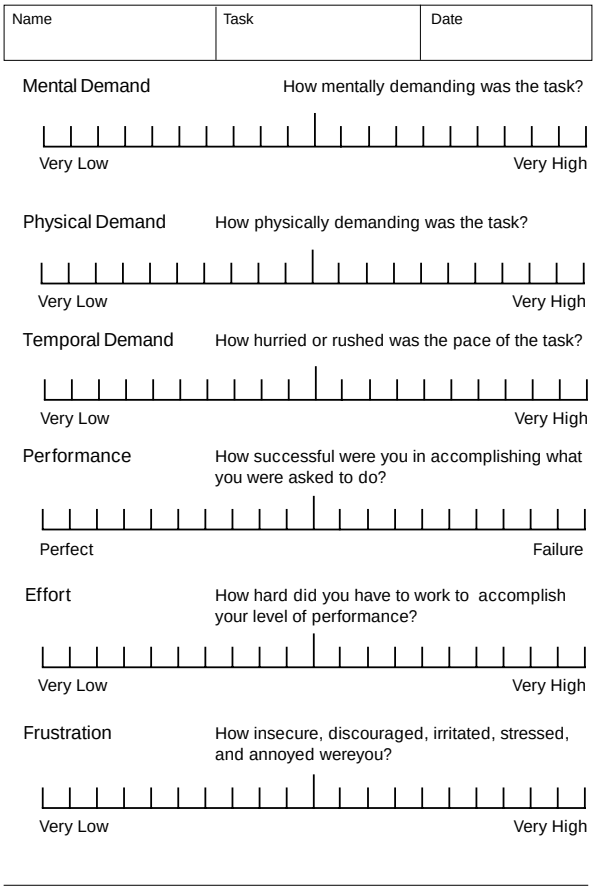
APPENDIX C

NASA-TLX SURVEY

Figure 8.6

NASA Task Load Index

Hart and Staveland's NASA Task Load Index (TLX) method assesses work load on five 7-point scales. Increments of high, medium and low estimates for each point result in 21 gradations on the scales.



APPENDIX D

MECHANICAL TURK SAMPLE QUALIFICATION EXAM

Qualify for Image Landmark Selection HITs

Click the button below to go through the quick tutorial, then answer the questions below to instantly qualify. NOTE: There is one ONE correct answer for each question. Your score will be out of 100%.

In order to pass the test, please do the quick tutorial!

Open Tutorial

1. Where would you draw the landmark selection box in the following image?

(The stop light)

(The car)

(The middle of the intersection)

2. Where would you draw the landmark selection box in the following image?

(The pedestrian)

(The car)

(The building)

3. Where would you draw the landmark selection box in the following image?

(The telephone pole)

(The house)

(The house)

4. Where would you draw the landmark selection box in the following image?

(The crosswalk sign)

(The garbage cans)

(The cars)

4. Where would you draw the landmark selection box in the following image?

(The restaurant sign)

(The tree)

(The cars)