AUTONOMOUS PERCEPTION AND NAVIGATION IN UNKNOWN INDOOR ENVIRONMENTS

by

Thomas Victor Ilyevsky

A Dissertation

Submitted to the Faculty of Purdue University

In Partial Fulfillment of the Requirements for the degree of

Doctor of Philosophy

School of Electrical and Computer Engineering

West Lafayette, Indiana

August 2021

THE PURDUE UNIVERSITY GRADUATE SCHOOL
STATEMENT OF COMMITTEE APPROVAL

Dr. Jeffrey Mark Siskind, Chair

School of Electrical and Computer Engineering

Dr. Robert Givan

School of Electrical and Computer Engineering

Dr. Ronnie B Wilbur

Department of Speech, Language, and Hearing Sciences

Dr. Thomas Talavage

Department of Biomedical Engineering, University of Cincinnati

Approved by:

Dr. Dimitrios Peroulis

This thesis is dedicated to my parents, Leonid and Tatyana Ilyevsky, and my partner, Emily Stillman.

ACKNOWLEDGMENTS

This work was supported, in part, by the US National Science Foundation under Grants 1522954-IIS and 1734938-IIS, by the Intelligence Advanced Research Projects Activity (IARPA) via Department of Interior/Interior Business Center (DOI/IBC) contract number D17PC00341, and by Siemens Corporation, Corporate Technology. Any opinions, findings, views, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views, official policies, or endorsements, either expressed or implied, of the sponsors. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes, notwithstanding any copyright notation herein.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS

m	meters
s	seconds

ABBREVIATIONS

AMT	Amazon Mechanical Turk
API	Application Programming Interface
ASCII	American Standard Code For Information Interchange
CARLA	Car Learning to Act
EE	Electrical Engineering building
FC	Fully-Connected
FOV	Field Of View
FSM	Finite-State Machine
GUI	Graphical User Interface
HAMP	Hampton Hall of Civil Engineering building
IMU	Inertial Measurement Unit
KITTI	Karlsruhe Institute of Technology and Toyota Technological Institute
KNOY	Knoy Hall of Technology building
LiDAR	Light Detection and Ranging
MSEE	Material Sciences and Electrical Engineering building
ME	Mechanical Engineering building
NDH	Natural-language Dialog History
NMS	Non-Maximal Suppression
O2O	Office-to-office
OCR	Optical Character Recognition
PHYS	Physics building
R2R	Room-to-room
RCM	Reinforced Cross-Modal Matching
ReLU	Rectified Linear Unit
RGBD	Red Green Blue Depth
ROS	Robot Operating System
Seq2Seq	Sequence-to-Sequence
SF	Speaker-Follower

SLAM	Simultaneous Localization And Mapping
TBS	Tactical Behavior Specification
VLN	Vision-Language Navigation
YOLO	You Only Live Once

ABSTRACT

Standard off-the-shelf SLAM algorithms allow robots to build 2D maps of their environments and consequently enable them to navigate to (x, y) coordinates in those maps. However, this is a large step removed from a robot finding and going to a professor's office or locating an elevator and taking it up one floor. The robot would have to robustly detect and localize doors and elevators in a hallway. Additionally, given directions to this hallway, the robot would have to accurately follow them in a previously unknown environment. In this thesis, we propose solutions to these two key challenges associated with finding a goal in an unknown indoor environment. We present a robust algorithm that relies on image and laser-range data to detect doors. This algorithm is combined with a set of common-sense rules to enable a robot to efficiently find a specific door in a hallway. To follow directions in an unknown environment, we propose a convolutional neural network-based approach that takes a local crop of the 2D SLAM map and a command as input to produce navigational goal points and feedback for the robot as output. All of these methods are deployed on a real robot and evaluated in the form of live trials in previously unseen and unmodified office environments.

1. INITIAL SYSTEM

This chapter includes a submission to TRO, which describes a system that finds a goal door on a single floor of a single building. I was responsible for the following technical section: Section 1.3.5. Jared Johansen was responsible for the following technical sections: Section 1.2.3, Section 1.3.1, Section 1.3.2, Section 1.3.3, and Section 1.3.4.

We worked on the comparison experiments in Section 1.6 together. Jared Johansen worked on the system we used to collect the dataset; we both worked on collecting and annotating the data. We both worked on the code that loaded and trained the system; I trained the models and computed the simulated validation results. We both worked on the system that ran the live experiments; Jared Johansen ran the experiments and computed the test results.

1.1 Introduction

The Amazing Race^{TM} is a popular reality television show in which two-person teams race to some designated location. They typically have to figure out where they are, navigate through foreign areas, and ask people for directions to their destination. State-of-the-art artificial-intelligence and human-robot-interaction research enables robots to solve such natural-language-driven navigation tasks [1]–[8]. However, these systems suffer from certain limitations such as requiring a specific syntactic structure for natural-language commands or requiring a map of the environment. A system design without these crutches is crucial for operating autonomously in new, unknown environments. Additionally, it is important for these systems to have productive interaction with humans for the purpose of receiving instructions to execute or to learn new information. However, it is impractical and inconvenient for every person to know the precise details of a robotic system in order to interact with it. Therefore, robotic system designs should be based on and thereby exhibit human-like behavior to enable natural and useful interaction with humans.

To drive research towards this outcome, we propose a novel task: The Amazing Race^{TM}: Robot Edition. The task is this: we place the robot in an unknown environment, without a map, and give it the name of a person, room, or building to find. Unlike much prior work in the field of VLN, [9]–[13], this task requires real navigation in a physical environment, not in a simulated one. Additionally, our full task is considerably more complex than the task underlying current VLN research. Current VLN research simply gives an agent directions which the agent then follows. Our task requires that the robot find and approach a person, engage that person in a dialog to obtain directions, follow those directions, and find the goal by reading door tags. Our task involves deploying a robot in a new environment where it does not have all of the information it needs to perform its responsibility. There are individuals in the environment who have more knowledge than it does. In order to effectively and efficiently carry out its task, it must seek out those individuals and obtain necessary information from them. This task is general in that any physical robot should be able to find a goal location in an unknown environment. For the purpose of constructing an initial solution to this task, we restrict the goal to finding a door with a specified number on a single floor of a building. Because the robot has no prior knowledge about the goal location nor the structure of the environment, it must seek a person for assistance. Once a person is found, the robot must engage them in a dialogue to obtain directions to the goal. It has to follow these directions to reach the hallway that the room is in, at which point it would have to systematically search for doors and read their door tags to locate the correct one. We present a novel finite-state-machine (FSM) system design that makes these logical steps to accomplish this task.

Our hypothesis is that this problem requires a specific set of abilities that are invoked in a particular order, equivalent to an FSM. We carefully chose a sequence of specific states for our FSM design that reflects the steps a typical person would take to efficiently find a room in an unknown office environment. Initially knowing nothing about the environment, many people would ask the first person they see for directions. After receiving these directions and potentially asking for clarification, they would follow them to the approximate location of the room and begin looking for the specific number. Our FSM design mimics this human behavior, which can result in a shorter and more efficient path to the goal as opposed to a simple exhaustive search of the environment. Although an exhaustive search would eventually succeed, it would not necessarily take the most efficient path, would ignore people who may have a rich understanding of the environment and a willingness to help, and would not push the envelope on robot cognition. Additionally, an exhaustive search becomes unsuitable as the size of the environment increases.

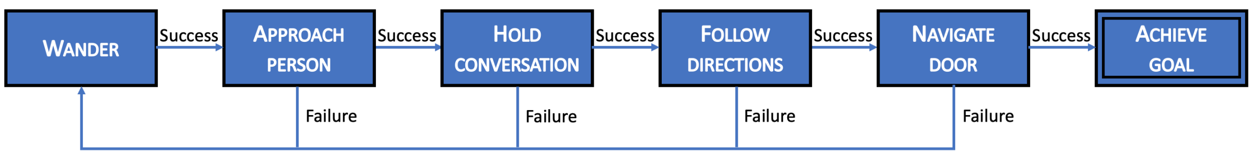


Figure 1.1. Finite-state-machine view of the system architecture.

In order to enable such human-like reasoning, we abstract low-level sensor data such as audio, video, and LiDAR into information that a human would have readily available such as navigation directions and potential goal locations. The states in our FSM design, as shown in Figure 1.1, also referred to as *behaviors*, make use of this information to handle new and/or complex situations. They also incorporate methods to handle complete, partial, or erroneous information similar to how a human would when encountering such in conversation or navigation. We believe this design mimics high-level aspects of human behavior on the same task. Additionally, to the best of our knowledge, this paper makes the following contributions:

- We present a method that abstracts noisy SLAM data into navigation actions that a robot can execute.
- We present a method that allows the robot to find and approach a person to ask for directions, ignoring those that appear unwilling to engage in dialog.
- Our method engages in multi-turn dialogue that involves generating natural-language clarification questions in response to an incomplete or inconsistent plan produced from incomplete or inconsistent directions.
- Our method detects and localizes doors, and their associated door tags, as potential targets, reading the door tags. The door-tag-reading process is also noisy, which we mitigate through specific mechanisms. The process of searching for doors makes use of common-sense knowledge: adjacent doors are numbered consecutively with odd and even numbers on opposite sides of the hallway. The knowledge is only a default; our robot is robust when the default is violated. But this knowledge leads to both more efficient and more human-like behavior.
- All forms of input, including the spoken language, the camera input used to find people and read door tags, and the LiDAR input used for SLAM are noisy. Further, action execution is noisy and unreliable. Nonetheless, our methods are robust in the face of all this noise.
- We demonstrate our methods on a real robot in the real world, where the position and actions are continuous rather than a discrete graph with a small finite, set of discrete actions.

In order to evaluate the performance of our system, we conducted 52 trials across 4 floors of each of 3 buildings previously unseen by the robot. We recruited 13 untrained volunteers to provide the robot with directions to the specified goal room number in each trial. Due to our system's ability to recover from individual behavior failures, we demonstrate a high success rate of 76.9%. Additionally, we document observations made during the trials as opportunities for further research into this task.

1.2 System Overview

In this section, we provide a high-level overview of our system's hardware, software, and architecture as well as the method used to construct a qualitative map from the quantitative map produced by SLAM. We provide detailed descriptions of our system's components in the next section.

1.2.1 Hardware and Software

Our robotic platform consists of a Clearpath Husky A200™ UGV equipped with an Open IMU UM7, Velodyne VLP-16 3D LiDAR, Axis M5525-E PTZ Camera, Blue Yeti microphone, and a System76 Laptop with two Nvidia GeForce GTX 1080 GPUs. Each of these is a commercial, off-the-shelf product. Clearpath integrated the IMU, LiDAR, PTZ Camera, and laptop onto the Husky A200™ UGV.

Our software[1] is implemented in a combination of C++ and Python, using ROS Kinetic as the communication framework between different components of the system. We use Google Cartographer [14] to perform simultaneous localization and mapping (SLAM) from data from the IMU and 3D LiDAR. We record speech with the Blue Yeti microphone and convert it into text using the Google Speech-to-Text [15] API. We use the Stanford Parser [16] to parse the text. We use Python's `pyttsx3` library [17] to synthesize speech through the laptop speaker. We use YOLOv3 [18] to detect people in images taken from the Axis camera. We use LSD (Line Segment Detector) [19] on images from the Axis camera as part

[1]↑All software used to produce the results in this manuscript is available at `https://github.com/qobi/amazing-race`.

of our method for detecting doors. We extract text from images of door tags taken with the Axis camera with Google's Optical Character Recognition [20] API. We use a LiDAR-camera calibration software package [21] to compute a calibration matrix between the 3D LiDAR and the Axis camera. This matrix lets the system compute 3D locations for detected people and doors in the environment.

1.2.2 Architecture

Our architecture is a finite-state machine illustrated in Figure 1.1. This design makes logical steps towards reaching the goal and recovers from failure at any of those steps. Each state has a success condition that leads to the next state in the pipeline as well as failure conditions which result in a transition to the initial state of the system. Given a goal description, the initial state is WANDER, as the robot needs to find a person to get directions to the goal. This state allows the robot to explore the environment while simultaneously attempting to detect and track people. Once a person is found, the robot enters the APPROACH_PERSON state, in which it drives towards the person and synthesizes speech to grab their attention. If the robot successfully reaches the person, it initiates a conversation with them in the HOLD_CONVERSATION state. The robot uses speech synthesis and speech recognition to request directions to the desired goal and interpret the person's response, respectively. It can ask an appropriate clarification question if the interpreted directions are incomplete. Once a complete set of directions are determined through dialogue with the person, a transition to the FOLLOW_DIRECTIONS state is made. FOLLOW_DIRECTIONS executes each direction in order by continuously mapping the environment to determine when to continue to the next instruction, such as turning `left` when a left turn becomes available. Successful completion of the directions implies that the robot is in the hallway that contains the goal. The robot then enters the NAVIGATE_DOOR state which involves detecting doors and driving up to them to inspect their door tags. This will let the robot ultimately confirm arrival at the desired goal.

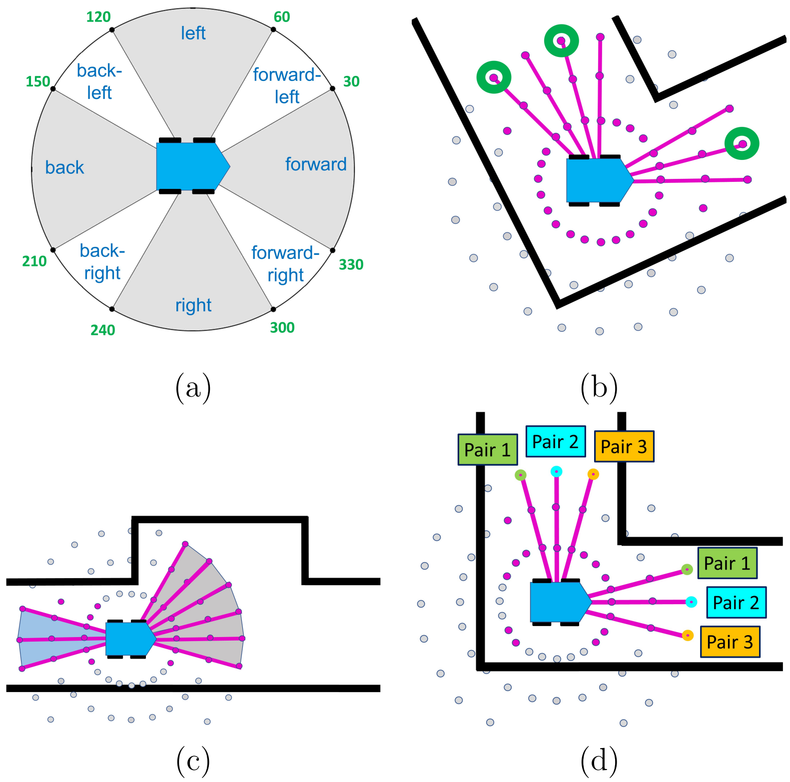


Figure 1.2. (a) Qualitative direction categories and their heading relative to robot orientation. (b) An example of trajectory generation with trajectory distance limited to 3.6 m. The black lines represent obstacles (i.e., walls). Target points of potential trajectories for 24 headings are shown as small points. The gray target points are filtered out as either unknown space or too near an obstacle. The pink target points represent drivable trajectories. The dark pink lines are headings associated with maximal drivable trajectories of distance 3.6 m. The dark green circles represent qualitative drivable trajectories with the qualitative direction labels forward, left, and back-left. (c) An example of how intersection detection and classification ignores large traversable areas that arise from alcoves and other large open spaces. A single scale, namely a distance of 3.6 m, is depicted. The gray traversable area is much wider than a hallway so the drivable trajectories in that traversable area are discarded. Only the drivable trajectories in the blue traversable area are taken as hallway trajectories. (d) An example of how intersection detection and classification, at a single scale of 3.6 m, forms candidate intersections as tuples of hallway trajectories at a given scale and selects at most one of these as the single detected and classified intersection. The target points of the hallway trajectories in Pair 2 are furthest from obstacles, so it will be selected.

1.2.3 Navigation Process

The physical environment has intersections connected by hallways. Humans give navigation instructions by describing paths through hallways between intersections. They use informal terms to classify intersections into various types (e.g., **elbow**, **three-way**, and **four-way**) and to distinguish between different hallways emanating from intersections by their heading (e.g., forward, left, and right). To facilitate the interpretation of these informal navigation instructions, our robot constructs and maintains two maps of the environment, one quantitative and one qualitative. The *quantitative map* is an occupancy grid constructed by SLAM. It is a 2D matrix of cells, where each cell corresponds to a 5 cm square of the ground that takes one of three possible classifications: *free*, *obstacle*, or *unknown*. Generally, in an indoor office environment, free cells would be rooms or hallways, obstacle cells would be walls or objects, and unknown cells would be unexplored areas of the building or anything outside of the walls. The *qualitative map* is a graph whose vertices represent detected intersections labeled with intersection type and whose edges denote detected hallway paths. We refer to these as *registered* intersections and hallway paths since the process of constructing and maintaining the qualitative map can add, remove, merge, and update registered intersections and hallway paths when detecting new ones. The quantitative map, along with the robot's current pose (position and orientation) in that map, is continually maintained and updated by a background process running SLAM. The qualitative map is constructed and updated from the quantitative map and robot pose by a background process continually running at 1 Hz. We refer to the latter background process as the *navigation process*. The qualitative map produced by the navigation process is used by several of our robot behaviors.

Trajectory Generation

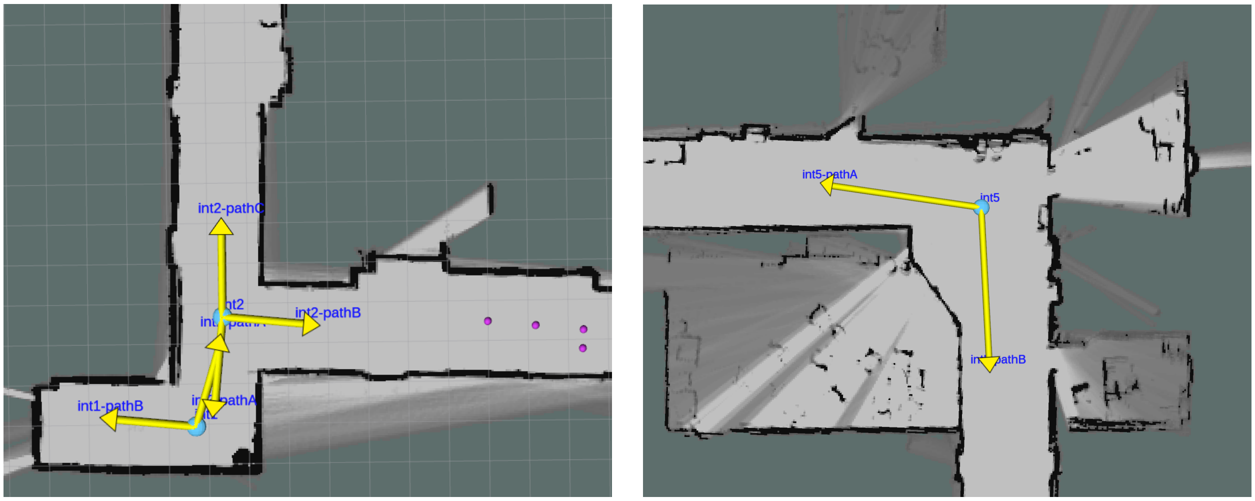
The first step of the navigation process is to construct various sets of trajectories, short paths that the robot can drive from its current pose. A *trajectory* is a target point in world coordinates at a specified distance and heading from the current pose. Low-level robot navigation uses trajectory target points as driving instructions. We nominally consider all distances that are integral multiples of 1.2 m from 1.2 m to 7.2 m, combined with all headings that are integral multiples of $\frac{360°}{64}$, as *potential trajectories*. These are filtered as follows. We first remove all trajectories which require traversing a point that is within 0.6 m of an obstacle or unknown space to reach the target point. This yields a set of *drivable trajectories*. The 0.6 m threshold was chosen as it is our robot's circumscribed radius; it would not fit through passageways smaller than this. Since we need to search each trajectory for obstacles, the 1.2 m quantization was chosen to reduce the number of potential trajectories considered while still considering sufficiently many to successfully interpret and execute human navigation instructions. We then filter the set of drivable trajectories, keeping only the ones with the largest distance for each heading. This yields a set of *maximal drivable trajectories*. Finally, we label each maximal drivable trajectory with one of eight *qualitative directions* based on its heading (Figure 1.2a). We then filter the set of maximal drivable trajectories, keeping at most a single trajectory for each qualitative direction, the one with the median heading. This yields a set of *qualitative drivable trajectories*, there being at most eight of these, each labeled with a distinct qualitative direction. Figure 1.2(b) illustrates the process of trajectory generation.[2]

Intersection Detection and Classification

The next step in the navigation process is to determine whether the robot is in an intersection, and if so, to determine the type of that intersection (e.g., **elbow**, **three-way**, or **four-way**). It does this at multiple scales, with each distinct trajectory distance taken as a scale, to tolerate different building designs with different hallway lengths and widths (Figure 1.3).

At each scale, this process starts with all drivable trajectories at that scale. The first step is to eliminate drivable trajectories that would not be considered as driving through hallways. This is done by grouping all adjacent drivable trajectories (with headings differing by $\frac{360°}{64}$) to represent a *traversable area* (Figure 1.2c). The width of a traversable area is determined as the maximal Euclidean distance between any two target points of drivable trajectories in

[2]↑While we classify eight distinct qualitative directions and qualitative driving directions, the remainder of the manuscript only considers the four primary ones.



(a) (b)

Figure 1.3. (a) These intersections, with very short hallways, can only be detected with a distance ≤2.4 m. (b) This corner, with a decorative front to a lab space, can only be detected with a distance ≥4.8 m. All intersections are correctly detected and classified by considering multiple distances.

the traversable area. If a traversable area is wider than the specified width of a hallway for that building, all trajectories associated with that traversable area are discarded. This allows the robot to ignore alcoves and similar open spaces, and yields a set of *hallway trajectories*.

The hallway trajectories are grouped into pairs, triples, and quadruples whose headings are ≈90° apart. These constitute potential intersections of type **elbow**, **three-way**, and **four-way**, respectively. Each such tuple constitutes a *potential intersection*. Potential intersections whose elements are not separated by obstacles are discarded. This yields a set of *intersection candidates*, each of which has the robot's position as its initial location. Intersection candidates are scored based on the sum of the distances of the location and each element's target point from the nearest obstacle.

This process may produce multiple intersection candidates, e.g., an **elbow** will always be present when a **three-way** is also present and an intersection candidate produced at a larger scale may also be present at a smaller scale. The navigation process prioritizes intersection candidates as follows: 1. quadruples over triples, 2. triples over pairs, 3. larger scales over smaller scales, and 4. higher scores over lower scores. This yields at most a single detected and classified intersection at the current robot position. For reasons to be discussed later, each detected and classified intersection is given a unique identifier and each hallway trajectory in that intersection tuple is also given a unique identifier.

As an example, consider Figure 1.2(d), which illustrates intersection detection and classification at a single scale, namely 3.6 m. Three pairs of hallway trajectories (green, blue, and orange) are shown (out of a possible nine), each hallway trajectory associated with the highlighted (green, blue, and orange) target point. Each such pair constitutes an intersection candidate. Pair 2 (blue) will have the highest score (since its target points are furthest from obstacles) and will be kept as the single detected and classified intersection.

Intersection Refinement

Since the navigation process performs intersection detection and classification repeatedly at 1 Hz, it can detect the same intersection multiple times, particularly if the robot remains in an intersection for more than 1 s. This is akin to how object detectors can place multiple but different boxes around an object. We address this by performing a kind of nonmaximal suppression (NMS) in a fashion analogous to how object detectors deal with this problem. Moreover, intersection detection and classification is performed relative to the robot pose. As this changes, the classified intersection type might change due to noise (e.g., classifying an intersection as an **elbow** when it is a **three-way**). Further, through a process described below, detected and classified intersections are registered as vertices in the qualitative map. Registered intersections are labeled with their location, which we wish to be the center of the physical intersection. However, detected intersections are registered with their location being the robot position at the time of detection, which might not be the center of the physical intersection.

To address this, we perform intersection refinement. We suppress detection of new intersections when the robot is within 2 m of a registered intersection. Further, we continuously refine the locations and classification labels of registered intersections in a background process. This process recomputes the intersection candidates while imagining the robot to be at every point in a 3×3 grid, with 0.4 m spacing, centered on the current location of each registered intersection, using the current SLAM occupancy grid. The intersection candidates are pooled and prioritized as in initial intersection detection and classification to yield a single redetection and reclassification. The registered intersection is updated to reflect the redetected and reclassified intersection, including its location. We enforce a constraint that intersections cannot shift more than 2.4 m from their original location. For efficiency, this refinement is only applied to a registered intersection when the robot is within 5 m of that intersection.

Each detected and registered intersection contains 1. its location, 2. its type, and 3. a tuple of hallway trajectories. The hallway trajectories represent the hallway paths between intersections. During refinement, the redetected intersection might contain different hallway trajectories than the current registered intersection. This can happen when, for example, a group of people stop to talk and block a hallway trajectory in the intersection. During intersection refinement, we wish to maintain the same unique identifiers associated with each intersection and hallway trajectory in that intersection. This requires constructing a correspondence between the hallway trajectories in the registered intersection and those in the redetected intersection. The optimal correspondence is found with the Hungarian algorithm [22] applied to a bipartite graph whose left vertices are the hallway trajectories in the registered intersection, whose right vertices are the hallway trajectories in the redetected intersection, and whose edges are given a cost which is the angular distance between the hallway trajectory headings. Edges in this bipartite graph are only created when the angular distance is $<30°$. This correspondence is used to reassign the unique identifiers from the hallway trajectories in the registered intersection to those in the redetected intersection.

As described previously, a redetected intersection may have different type than a registered intersection, and thus may have a different number of hallway trajectories. Thus, the correspondence as produced above may fail to assign a hallway trajectory from the registered intersection to one in the redetected intersection either because the redetected intersection is of a different type with fewer hallway trajectories or because of eliminated edges. A hallway trajectory in the registered intersection that does not have a corresponding hallway trajectory in the redetected intersection is maintained, with its unique identifier, in the registered intersection in a *deactivated* state, so that it can be reactivated later with the same unique identifier.

Intersection Graph

The unique identifiers associated with registered intersections and their hallway trajectories, whose consistency is maintained over time by intersection refinement, allow us to construct a graph to represent the qualitative map. When the robot comes within 0.5 m of a registered intersection, it selects the hallway trajectory whose target point is closest to the robot position as the one used to enter the intersection. The robot's path is searched backward until it comes within 3 m of a hallway trajectory from the last registered intersection it visited. An edge in the qualitative map is registered between these two registered intersections and represents a hallway path. The weight of this edge is taken as the Euclidean distance between the locations of the two registered intersections. If the backward search process does not produce a hallway trajectory, that hallway trajectory is left unconnected, to represent hallways that have not been completely explored yet. This graph is continually constructed and updated by the navigation process. See Figure 1.4 for an example of the qualitative map produced for a single floor of a building.

1.3 States of the System Architecture

In this section, we describe the the implementation of each state in our system.

1.3.1 Wander

WANDER is the initial state of the system and lets the robot continuously navigate through the environment, trying to detect and track people until it finds an approachable person. The general strategy is to explore the environment in a human-like manner, usually moving forward, choosing a direction to go when at an intersection, and only turning around when reaching a dead end.

Wander Substates

The WANDER state has five substates: 1. MAKE_DECISION, 2. ROTATE_RECOVERY, 3. ROTATE, 4. DRIVE_FORWARD, and 5. DRIVE_THROUGH_INTERSECTION. WANDER

Figure 1.4. Qualitative map produced for a single floor of a building. The red line indicates the path that the robot traveled. Blue spheres are registered intersections. Yellow arrows are hallway trajectories associated with each registered intersection.

enters the MAKE_DECISION substate first, where it analyzes whether it is in a registered intersection and which qualitative directions are available. If no qualitative directions are available (e.g., when it is first initialized), it enters the ROTATE_RECOVERY substate, which causes it to spin in place 360°. This substate helps to update the quantitative and qualitative maps in the immediate vicinity, which determine whether any qualitative drivable trajectories are available. If none are available, it stays in this substate. If qualitative drivable trajectories are available, the robot selects the one whose heading is closest to its current orientation as the *recovery angle*. The robot enters the ROTATE substate to match its orientation with that of the recovery angle and will then enter the DRIVE_FORWARD state. This series of substates helps it find its way out of alcoves, entrances, or elevator landings and into hallways.

When in the DRIVE_FORWARD state, the robot continuously drives ***forward*** while monitoring for registered intersections. This ***forward***, which we use to indicate when and how a robot drives down a hallway, is distinct from the qualitative drivable trajectory forward and will be explained in Section 1.3.1. When it enters a registered intersection, the robot enters the MAKE_DECISION substate to determine what to do. First, it determines which hallway trajectory it entered the registered intersection from. Hallway trajectories included in registered intersections maintain a *visitation time* as an indication of when they were last visited. The visitation time for the hallway trajectory used to enter the registered intersection is updated to the current time. Then, it temporally orders all active hallway trajectories of that registered intersection. If it has visited each hallway trajectory, it selects the oldest one and takes it. If there are one or more hallway trajectories that it has not taken, it randomly selects one and takes it. It then updates the visitation time of the selected hallway trajectory with the current time. Maintaining visitation times allows the robot to explore the environment in a thorough manner, preferring to visit areas in the map that it has not seen or that it has seen least recently.

After selecting the hallway trajectory to take, if the hallway trajectory is to the left or right of the robot's pose, the robot will enter the ROTATE substate to rotate 90° to the left or right, respectively. Thereafter, or in the case that the robot chose to drive straight through the intersection, it enters the DRIVE_THROUGH_INTERSECTION substate to move out of the intersection and into the hallway. This substate prevents the robot both

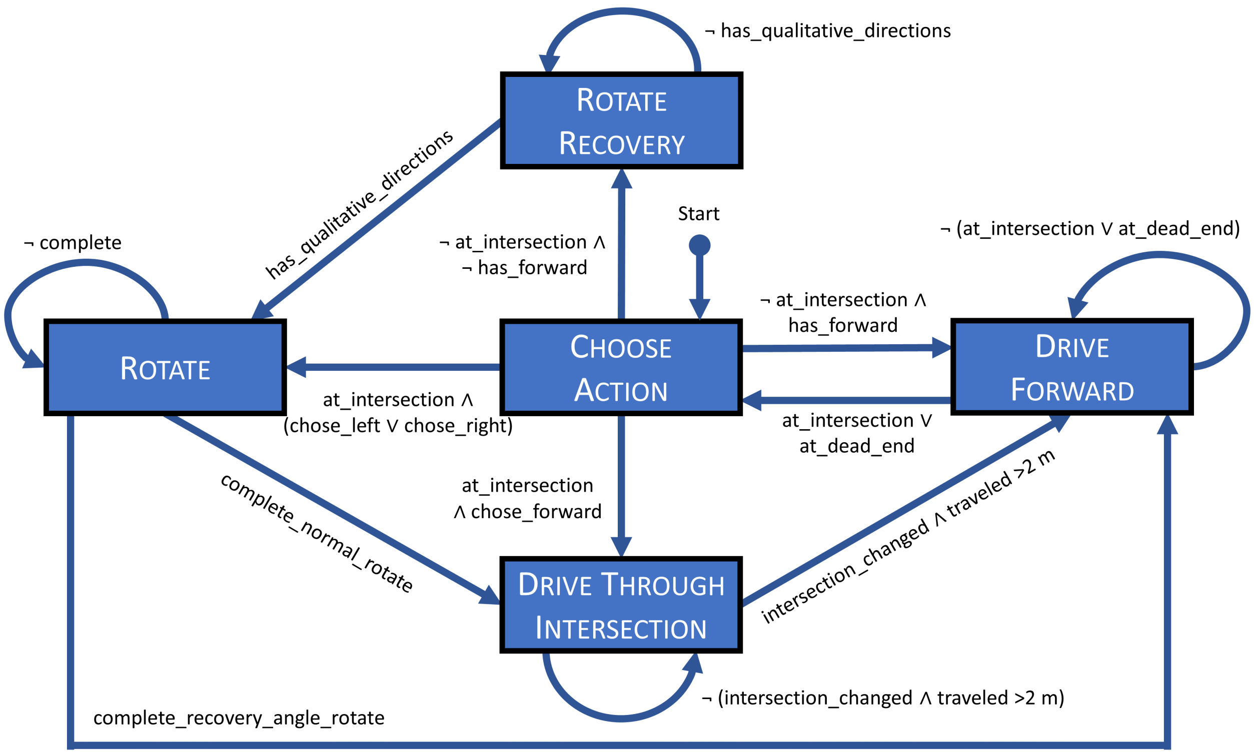


Figure 1.5. The Wander finite-state machine.

from immediately recognizing it is in a registered intersection and causing it to re-evaluate what to do. The Drive_through_intersection substate has the robot drive ***forward*** continuously. Once it has traveled more than 2 m and the intersection type has changed, the robot returns to the Drive_forward substate.

If, while driving ***forward***, the robot reaches a dead-end (which is determined by there being a back qualitative drivable trajectory but no forward, left, or right qualitative drivable trajectories), the robot performs a single 360° spin to ensure that it truly is at a dead-end (instead of simply having failed to detect an adjoining hallway). After performing this spin, it will enter the Make_decision substate, which, if it is still at a dead-end, will enter the Rotate_recovery substate. This substate will see that there is a single qualitative drivable trajectory available (back), which it uses as its recovery angle. The robot would enter the Rotate substate, rotate 180°, and then return to the Drive_forward substate. These substates can be modeled as a FSM as shown in Figure 1.5.

Forward Driving Goals

When the robot starts driving down a hallway, it might not be in the middle of the hallway, and its orientation might not match that of the hallway. Despite starting in a suboptimal pose, we want the robot to move towards and drive down the middle of the hallway. To do so, when the robot enters the DRIVE_FORWARD substate, we take its current orientation as the ***forward*** *orientation*. This ***forward*** orientation will get refined over time as the robot drives down the hallway to more accurately reflect the same orientation as the hallway.

While the navigation process provides a set of qualitative drivable trajectories, due to the suboptimal starting pose of the robot, there may not be a qualitative drivable trajectory associated with the ***forward*** orientation. Therefore, we pass the ***forward*** orientation and a *cone angle* to the navigation process, ask it to find all maximal drivable trajectories within that cone and return the median one as the *median drivable trajectory*. Initially, the cone angle is $\pm 15°$ from the ***forward*** orientation. If no drivable trajectories are found within the cone, this process is repeated with a cone angle of $\pm 30°$ and then $\pm 45°$. When drivable trajectories are found within the cone, the median drivable trajectory is used as the ***forward*** *driving goal*.

This gets the robot moving towards the middle of the hallway and having its orientation more closely reflect that of the hallway. As it nears this first driving goal, these steps are repeated, but only up to a window of $\pm 30°$. After nearing its second driving goal, the robot is typically in the middle of the hallway and has the same approximate orientation as the hallway. For all remaining ***forward*** driving goals after this point, only a window of $\pm 15°$ is used. Decreasing the maximum possible window after navigation to the first and second driving goals is done to avoid ***forward*** driving goals that lead to undesirable behavior. As the robot drives down a hallway, there may be alcoves on the left or right. If a wide cone angle is used, drivable trajectories can be within the alcove and the median drivable trajectory that is returned from the navigation process can cause the robot to veer off from the center of the hallway. In the worst case, it can drive into the alcove, have no qualitative direction forward, and think it has arrived at a dead-end. Alternatively, if instead of there being an alcove, there is a hallway, the robot can inadvertently drive around a corner (without realizing it) and still think it is driving ***forward***. See Figure 1.6 for an illustration of this entire process.

Earlier, we stated that the ***forward*** orientation is updated over time to more accurately reflect the same orientation as the hallway. At first blush, it may seem logical to always use the robot's current orientation as the ***forward*** orientation. However, this fails under the following scenario. As the robot drives down a hallway, obstacles may appear in front of it (e.g., people walking and/or sometimes stopping in front of it—whether unintentionally or intentionally—to see how the robot will react). Standard local path planners handle these types of unexpected obstacles by navigating around them. As the robot is driving around such an obstacle, its orientation may change significantly such that it no longer matches that of the hallway and in some cases can be nearly perpendicular to the hallway. If, at this moment, the robot needed to update its driving goal, and used its current orientation as the ***forward*** orientation, the navigation process could either determine that there were no drivable trajectories or return a drivable trajectory that would move the robot in a direction it should not go. Either could derail the robot from properly driving down the hallway.

To handle this case, once the robot has moved 2 m from its starting point, we compute the angle from its starting point when it began driving down to the hallway to its current position and use this as the ***forward*** orientation. This gives a much more accurate approximation of the orientation of the hallway and makes the ***forward*** orientation robust to the situation described above.

There is one additional mechanism we employ to help the robot drive ***forward*** around obstacles but we will postpone that discussion until Section 1.3.4 as it makes more sense in that context.

Person Detection and Tracking

The ultimate goal of the Wander state is to locate a person to approach. Therefore, while the robot is navigating through the environment, it uses the Axis camera and 3D LiDAR to continuously detect and track people in that environment.

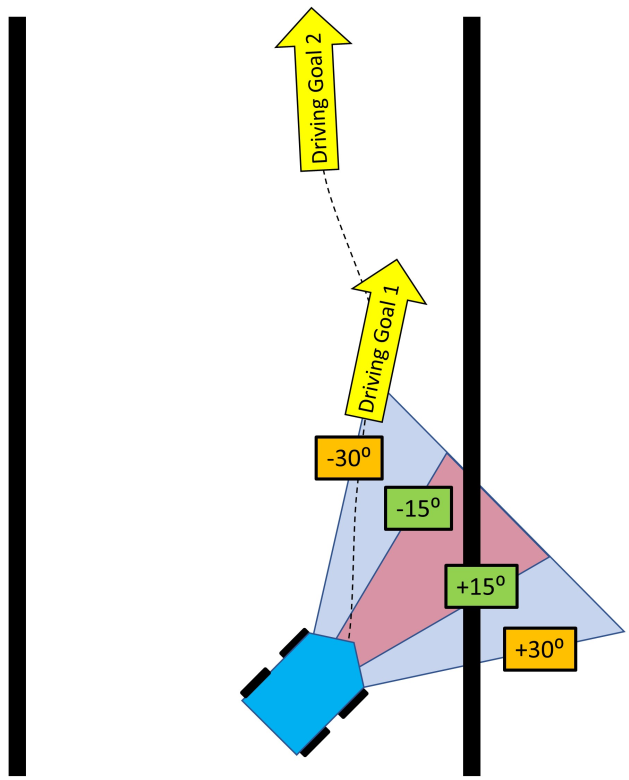


Figure 1.6. Example of how the robot determines a ***forward*** driving goal despite starting with a suboptimal pose. The robot is 1 m from the wall and its orientation is ≈45° off from that of the hallway. The transparent red cone indicates that the navigation process was unable to find any drivable trajectories using a cone angle of ±15° from the ***forward*** orientation. The transparent blue cone indicates that the navigation process was able to find a drivable trajectory using a cone angle of ±30° and thus a cone angle of ±45° is not required. The median drivable trajectory that the navigation process returned is used as a ***forward*** driving goal (labeled “Driving Goal 1”). Upon approaching that driving goal, the robot repeats this process, but only up to a cone angle of ±30° if necessary, and the yellow arrow labeled “Driving Goal 2” will be its second ***forward*** driving goal. This process incrementally moves the robot to the center of the hallway and changes its orientation to more closely match that of the hallway.

YOLOv3, a real time object-detection system, is used to detect people. It can detect multiple objects of different classes and provide their spatial location via bounding boxes. These boxes are used to determine the location of people in the quantitative map.

The camera's horizontal field of view angle and resolution are known. Each pixel in the camera can be mapped to a particular angle relative to the position of the camera. These angles can be used to extract the distance of the detected object from the LiDAR data. Because bounding boxes from object detectors are imperfect, and because standard object detectors localize detections with boxes even though the shape of most natural objects (e.g., humans) is not rectangular, some of the LiDAR points will shoot beyond the object and hit a more distant object or wall. Instead of taking the average of these points, we keep the closest one, which represents the nearest part of the detected person. By knowing the pose of the robot, camera, and LiDAR, we can determine the precise position of that object in the quantitative map.

These person detections, along with their quantitative map coordinates, are fed into a detection-based tracker. Typically, a detection-based tracker combines bounding boxes from adjacent image frames into tracks if their pixel coordinates sufficiently overlap. However, in our case, this method is unsuitable because the robot is constantly moving. Therefore, instead of combining detections that occupy the same spatial region in the camera's field of view, we combine detections that occupy/overlap the same location in the quantitative map. In this way, if the robot moves or rotates, the moving field of view does not affect the ability of the tracker to accurately piece together detections into tracks.

For all existing tracks, we compute the Euclidean distance from the last known location to each new detection. These distances constitute the cost to pair the track with the detection. We use the Hungarian algorithm to minimize the cost across all pairings of tracks with new detections and find the optimal pairing. We perform a sanity check to make sure that no pairings are unreasonable. We first determine the walking speed of the person, by computing how far they have traveled since the last detection and dividing that by the amount of time that has elapsed since then. If the walking speed is >2.5 m/s, we break the track-detection pairing and the detection becomes a new track. All pairings that meet this threshold are combined into a single track. Because detectors can be noisy and imperfect, the tracker employs forward projection on all tracks for up to 0.5 s to compensate for missed detections. Any tracks that go 0.5 s without an additional detection are pruned.

Tracks are classified based on the length of time they have been active and the average walking speed of the person. Once a track is $\geq$1.5 s long, the average walking speed is computed over the previous 1.5 s. The person's walking direction is determined by whether the most recent location is closer to the robot than the location $\approx$1.5 s ago. If the most recent location is closer by more than 0.3 m/s, the person is considered *approaching the robot*. If the most recent location is further by more than 0.3 m/s, the person is considered *walking away*. Otherwise, the person is considered *stationary*. Tracks that are $<$1.5 s long are considered as having just begun and are thus ignored as part of any decision making on the part of the robot. Sample outputs of each of these four situations are shown in Figure 1.7.

Tracks that are classified as stationary or approaching the robot are considered "approachable." When such occurs, the robot transitions to the APPROACH_PERSON state.

1.3.2 Approach_person

Once an approachable person is successfully located, the system enters the APPROACH_PERSON state to introduce itself to the person, drive up to them, and face them just as a human would. Using the person's location in the quantitative map, the robot computes a driving goal that is 0.8 m from the person (in a direct path from the robot to the person) with a pose that is facing the person, and begins driving there. If the person is approaching the robot, once they are within 7 m, the robot will hail them and ask them for help. When the robot is $\leq$2 m from the person, it will introduce itself to initiate the conversation.

Throughout this process, the person's location in the quantitative map is being monitored and updated, and new potential driving-goal locations are being computed. If any of these are $\geq$0.5 m from the previous driving goal, the robot updates its driving goal. This allows the robot to approach the person in a closed-loop fashion. If instead of stopping, the person chooses to walk up to the robot, once they are within 0.8 m of the robot, the robot will immediately stop and consider this as having successfully approached the person. It will continue the process of monitoring and updating the driving goal until it either reaches the

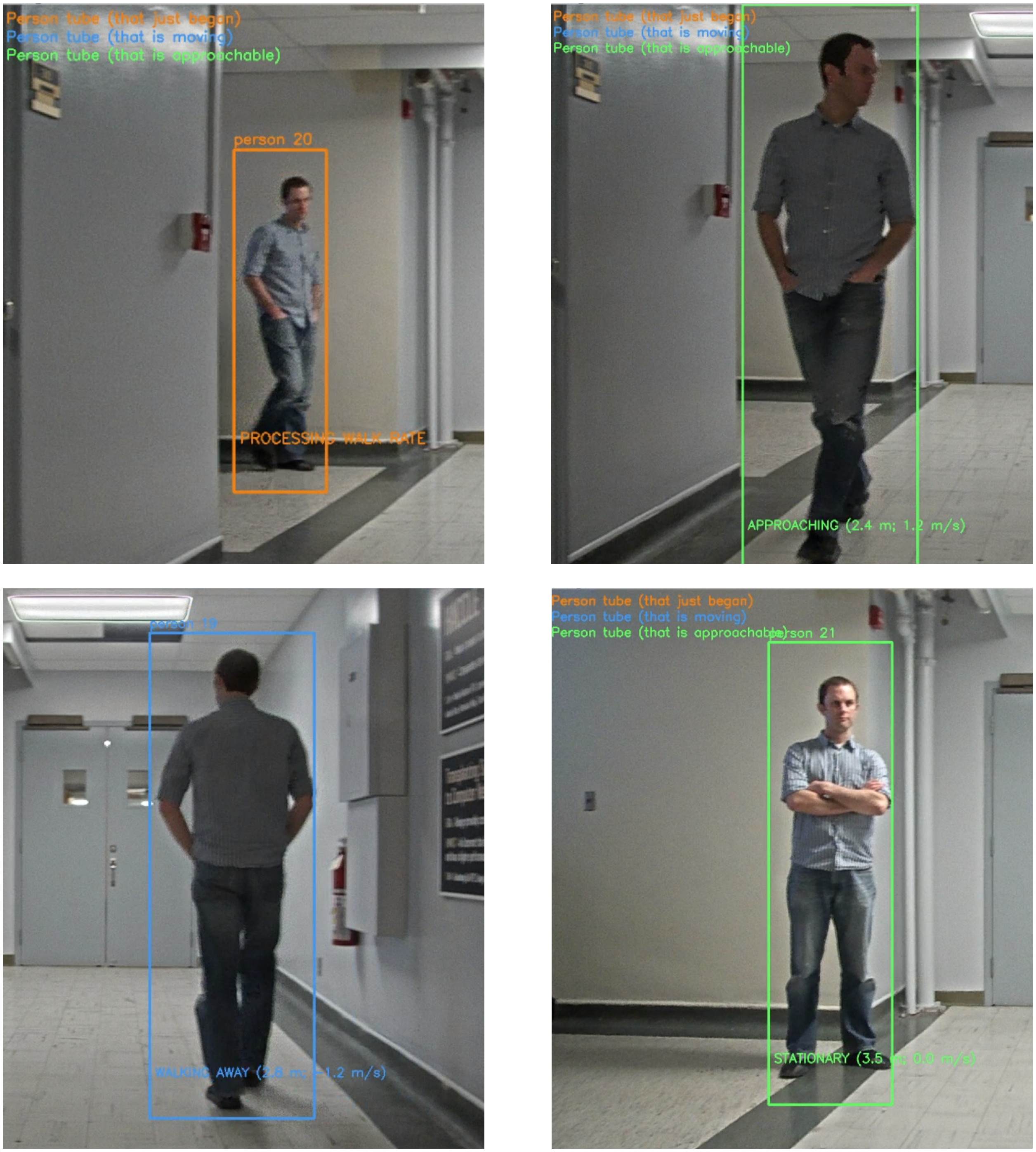


Figure 1.7. Approachable person detector output. Tracks that have just begun and whose walk rate/polarity are still being determined are colored orange. Tracks that are walking away are colored blue. Tracks that meet the criteria for being either stationary or approaching the robot, are colored green. Near the bottom of each bounding box, the person's distance from the robot is displayed on the left and their walking speed is displayed on the right. Positive implies approaching the robot; negative implies walking away.

person, they leave the field of view (e.g., enter a room), or begin to walk away. If either of the latter two scenarios occur, the robot will begin this entire process over with the next-closest approachable person. If there are none present, it returns to the WANDER state.

1.3.3 Hold_conversation

Once the robot approaches the person, it enters the HOLD_CONVERSATION state whose objective is to construct a plan of how to navigate to a specific destination by engaging in dialogue with the person. This plan is represented as a list of navigational actions that can be executed by the robot as commands. If the person were to formally communicate the plan to the robot, this task would be trivial. What makes this nontrivial is that the communication is done informally through spoken natural language.

Inferring a plan from spoken dialogue with a person presents several challenges. First, people often ramble and speak incoherently. Second, Google's Speech-to-Text API is imperfect and sometimes returns incorrect and/or nonsensical results. Third, state-of-the-art parsers, trained on written text rather than spoken text, create abnormal parse trees that make extracting relevant and accurate information more difficult.

Although, we use an off-the-shelf speech recognizer and only process the output of that, our system is robust in light of two crucial differences between spoken and text input. The first is that spoken input is typically far less grammatical than written text. The second is that all current speech recognizers, including the one that we use, are far from perfect, and introduce significant errors in the transcription process. Our system is robust in light of these two differences whereas a VLN system, which is trained and evaluated only on written text provided by humans, is not.

Our HOLD_CONVERSATION behavior does not construct a semantic interpretation from the parse tree. We use word spotting for a small set of concepts, where each concept can be lexicalized with wide variety. We use the parse tree as a filter for how these concepts can be linked together to form a plan. The plan is only a partial representation of what was said, a sequence of directions and intersections ending in a goal.

Table 1.1. Various kinds of actions that can fill plan steps as produced by HOLD_CONVERSATION.

Kind	Concept	English description
dir	`forward`	drive ***forward*** until a stop condition is encountered
	`left`	rotate 90° left
	`right`	rotate 90° right
	`turn-around`	rotate 180°
	`either`	`left` or `right` depending on availability
int	`elbow`	intersection of type **elbow**
	`three-way`	intersection of type **three-way**
	`four-way`	intersection of type **four-way**
	`int-L`	intersection with a left hallway trajectory
	`int-R`	intersection with a right hallway trajectory
	`int-F`	intersection with a left and/or right hallway trajectory, as well as a forward hallway trajectory
	`end`	intersection with a left and/or right hallway trajectory, but noforward hallway trajectory
goal	`goal-F`	goal is somewhere up ahead
	`goal-L`	goal is somewhere up ahead on the left
	`goal-R`	goal is somewhere up ahead on the right
	`person`	our next task is to find a person for further directions

To overcome these obstacles, we employ the following general approach to plan inference, i.e., constructing a plan from natural-language dialogue. We maintain a plan that consists of a sequence of steps, that are filled with actions of various kinds, including directions, intersections, and goals. Steps in the plan may be unfilled, denoted by $\square$. Initially, the plan consists of a single unfilled step. The current (partial) plan is used at every dialogue turn to generate a query to the person. The response is processed in the context of the current (partial) plan to fill in unfilled steps, potentially add new steps that are either filled or unfilled, and potentially change or delete steps. This processing takes two forms. One is a sequence of steps for filling in parts of the plan based on the person's response. The other is a set of rewrite rules that update parts of the plan based on the local context in the plan. If, after processing, the plan contains one or more unfilled steps, a new query is generated for the person and processing continues. This process continues until the plan contains no unfilled steps. To prevent an indefinitely long conversation, the length of the plan is restricted to ten steps. If a plan gets to that length, the robot ends the conversation, executes those steps, then searches for another person to ask.

The sequence of plan actions must be coherent. The rewrite rules attempt to render plans coherent. For example, if the plan indicates that the robot eventually reaches an **elbow** in the hallway, it would be incoherent for it to drive straight at the **elbow**. The robot uses its current knowledge to formulate and pose a query about a missing piece of information. The person's response might provide information about one or more steps, or change a step that was incorrect. Rewrite rules are applied to fill in or modify steps that may not have been explicitly given but were implied. This process is repeated until a complete plan is constructed.

Spoken Communication

Our robot converses in real time, adhering to the norm of taking turns while talking and using common sense to factor in previous utterances into subsequent utterances. Speech recognition and speech synthesis are mutually exclusive operations so a person's response is only processed when they finish speaking.

Information Extraction

In the HOLD_CONVERSATION state, the robot attempts to fill in the steps of a plan with actions from a natural-language utterance. Our plans are formulated with a certain plan structure that stipulates that adjacent steps form pairs that constitute commands to the robot. The first step in a pair must be a direction action. The second step in a pair will be a stop condition and takes the form of either an intersection action or a goal action. Only the last step in the plan can be a goal action. All other stop conditions must be intersection actions. Thus a plan can be though of as a sequence of commands, each command being a direction for the robot to drive in until the specified stop condition, with the robot ultimately checking that it has reached the goal. The various actions that can fill plan steps of various kinds are shown in Table 1.1.

Prior to any dialogue, the plan is initialized with a single empty step: [$\square$]. In order to populate the plan, the robot's first question to the person is 'Can you tell me how to navigate to $\langle$destination$\rangle$?' The steps we use to parse the response and populate the plan with actions are illustrated in Figure 1.8. First, the utterance is chunked by splitting it at key elements such as '.' or 'then.' Each chunk is fed into the Stanford Parser. This results in more accurate parse trees than feeding the whole utterance into the Stanford Parser. Then, the parse trees are searched for phrases that contain direction keywords (e.g., 'turn around,' 'left,' or 'right') to insert the corresponding actions into the plan. If no direction keywords are found, the utterance is searched for any intersection or goal keywords to insert the corresponding actions into the plan.

Each phrase that contains one of the direction keywords is concatenated with its preceding and succeeding phrase to provide context for information extraction. Starting with the leaf node in the parse tree that corresponds to the direction keyword, the tree is searched for the nearest parent with one of the following POS tags: ADJP, ADVP, CONJP, FRAG, INTJ, LST, NAC, NP, NX, PP, PRN, PRT, QP, RRC, UCP, VP, WHADJP, WHAVP, WHNP, or WHPP. The trees corresponding to the left and right siblings of this parent node are used as the preceding and succeeding phrases, respectively. The preceding phrase is further refined by searching for the first verb that precedes the direction keyword; if that does not exist,

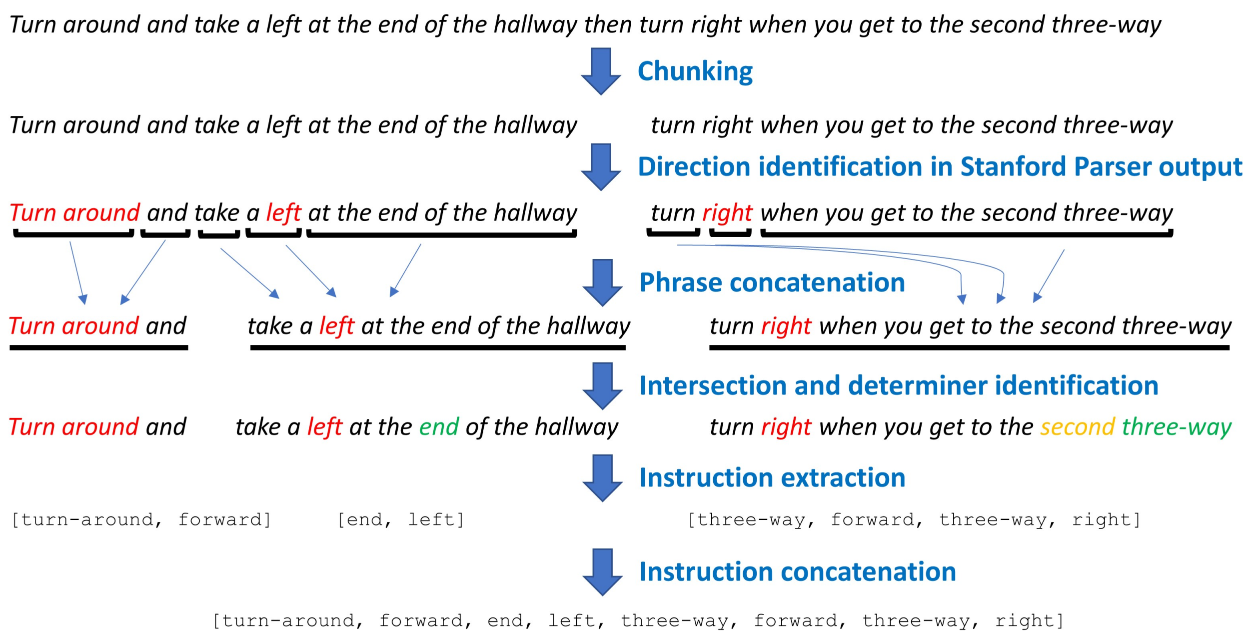


Figure 1.8. Instruction extraction via parsing steps.

we search for the first preposition that precedes the direction keyword. All words preceding the verb or preposition are eliminated from the preceding phrase. If no verb or preposition is found, no preceding phrase is used. In the event that the direction keyword's parent has no left or right siblings, we use the entire tree from the chunk as the concatenated phrase.

Each set of concatenated phrases is searched for the presence of an intersection keyword, direction and intersection determiners (e.g., 'first,' 'second,' or 'last'), and destination verbs (i.e., 'be,' 'find,' 'see'). If a destination verb is present, a goal action is created based on the direction keyword (i.e., 'left' → `goal-L`, 'right' → `goal-R`, otherwise `goal-F`). Otherwise, the direction keyword, direction determiner, intersection keyword, and intersection determiner are used to populate the plan with corresponding actions. For example, in the right chunk in Figure 1.8, it can be inferred from the determiner 'second' for the intersection keyword 'three-way' that the plan should contain two steps, each with a `three-way` intersection action. This also implies that the robot should go straight at the first one and turn right at the second one. If there were no direction keywords found, the phrase is searched for intersection or goal keywords that may have been missed, adding in the corresponding plan actions. The plan actions extracted from each chunk are concatenated to create the final plan. For a given utterance, this plan must be consistent with plan structure as described above to ensure proper execution.

1. The first step must be a direction action.
2. The last step must be a goal action.
3. A direction action must always follow an intersection action.
4. Two intersection actions cannot be adjacent.
5. Two direction actions cannot be adjacent (unless the first one is turn-around).

These properties of plan structure allow us to formulate patterns to describe all minimal invalid action sequences. Each such invalid action sequence can be rendered valid by appropriately adding, deleting, or modifying plan steps. Such plan modification is performed by rewrite rules. For example, consider the plan [`forward`, `left`], which implies that the robot must arrive at an intersection with a left turn. This plan can be rewritten as [`forward`, □, `left`] to indicate that it needs to populate a plan step with an action that specifies the intersection type at which it will make the left turn. These rewrite rules

Table 1.2. Plan rewrite rules. In the following, *dir* denotes any direction action, *ntadir* denotes any direction action except `turn-around`, *int* denotes any intersection action, and *goal* denotes any goal action. The first matching rule applies when multiple rules match.

goal	$\rightsquigarrow$	$\square$ *goal*
… *goal* $\square$	$\rightsquigarrow$	… *goal*
$\square$ `int-L` *dir* …	$\rightsquigarrow$	`forward` `int-L` *dir* …
$\square$ `int-R` *dir* …	$\rightsquigarrow$	`forward` `int-R` *dir* …
int `turn-around` …	$\rightsquigarrow$	`turn-around` …
`turn-around` *int* …	$\rightsquigarrow$	`turn-around` `forward` *int* …
int …	$\rightsquigarrow$	$\square$ *int* …
… `forward` `forward` …	$\rightsquigarrow$	… `forward` …
… *goal* …	$\rightsquigarrow$	… …
… `elbow` `int-L` …	$\rightsquigarrow$	… `elbow` …
… `elbow` `int-R` …	$\rightsquigarrow$	… `elbow` …
… `three-way` `int-L` …	$\rightsquigarrow$	… `three-way` …
… `three-way` `int-R` …	$\rightsquigarrow$	… `three-way` …
… `four-way` `int-L`…	$\rightsquigarrow$	… `four-way` …
… `four-way` `int-R`…	$\rightsquigarrow$	… `four-way` …
… *ntadir* `forward` …	$\rightsquigarrow$	… *ntadir* …
… *ntadir* *dir* …	$\rightsquigarrow$	… *ntadir* $\square$ *dir* …
… int_1 int_2 …	$\rightsquigarrow$	… int_1 $\square$ int_2 …
… *int* *goal* …	$\rightsquigarrow$	… *int* $\square$ *goal* …
… *dir*	$\rightsquigarrow$	… *dir* $\square$
… *int*	$\rightsquigarrow$	… *int* $\square$
… `elbow` `forward` …	$\rightsquigarrow$	… `elbow` …
… `elbow` $\square$ …	$\rightsquigarrow$	… `elbow` `either` …
$\square$ `forward` …	$\rightsquigarrow$	`forward` …
$\square$ `turn-around` …	$\rightsquigarrow$	`turn-around` …

are repeatedly applied to the plan until it no longer contains any invalid action sequences. Table 1.2 shows the complete list of plan rewrite rules.

Dialogue

The goal of HOLD_CONVERSATION is to construct a complete plan, with no unfilled steps. If the plan is not complete, a query is generated and posed to the person for the first unfilled step in the plan. Information is extracted from the person's response to fill the corresponding unfilled step in the plan. The type of the query depends on the pattern in which the unfilled step occurs. There are two query types: *single*, which requests a single piece of information, and *open-ended*, which gives the person more freedom in their response. Table 1.3 illustrates the different queries for each pattern and query-type.

If the query-type is single, HOLD_CONVERSATION looks for the presence of that particular type of information in the response. After finding it, all words up to it are removed and the remaining text is parsed in an open-ended fashion. Consider an example:

Robot: Which direction do I start out going?

Person: you start left, then turn right at the end of the hallway

Our parsing method extracts 'left' as the answer to the query and then replaces the corresponding $\square$ in the plan with `left`. Then it parses 'then turn right at the end of the hallway' in an open-ended manner and inserts the extracted instructions into the plan after `left`. If the query-type is open-ended, the response is parsed in the same way as the response to the robot's first question. The robot repeatedly poses questions, processes the respond, and applies rewrite rules until it generates a complete plan with no missing information. A complete and consistent plan indicates success and results in a transition to the FOLLOW_DIRECTIONS state.

If the plan is complete but not consistent, or if no plan could be constructed, then the plan [`right`, `person`] is created and a transition to the FOLLOW_DIRECTIONS state is made. This short plan causes the robot to rotate away from the person so that they are no longer in the field of view and will not be redetected as an approachable person. After rotating 90° to the right, the robot transitions to the WANDER state and will seek out a new person to ask for directions.

Addressing Corner Cases

To help facilitate a more natural conversation, we have a small amount of code to address a few corner cases that may arise in spoken conversation. If a response to a query is not heard within 5 s, the robot states this fact and repeats its query. If the robot is not able to extract any useful information from a response, it indicates such and may provide some information about what it does understand (e.g., directions and intersections). If the robot goes two turns without the plan changing (e.g., it fails to understand the person's instructions or it hears no response), the robot ends the conversation and carries out whatever portion of the plan is usable. If the person indicated the robot misunderstood their last utterance, the robot backs up one step, using the previous iteration of the plan and its corresponding query. If the person indicates that they would like to start over, the robot resets the plan and asks its original query. Some sample conversations from our trials are shown in Table 1.4.

1.3.4 Follow_directions

With a successfully extracted plan, the robot enters the Follow_directions state. The plan includes direction, intersection, and goal actions. Direction and intersection actions are grounded in the environment by the navigation process described in Section 1.2.3. The goal action is a transition condition to Follow_directions that it has completed the plan provided by the person and should transition to the next state.

Plan Preprocessing

In Section 1.3.1, we described the Drive_through_intersection substate and its purpose in ensuring the robot exited the registered intersection before being able to detect another registered intersection and re-evaluate what to do. That behavior is important to Follow_directions as well for a similar reason: if the robot entered a registered intersection, rotated in the direction specified in the plan, and immediately began looking for the subsequent intersection, it could mistakenly think it had reached it, despite still being in the same registered intersection. We want the robot to completely exit the current registered intersection before beginning to look for the subsequent one. To facilitate this behavior, Follow_directions takes the plan received from Hold_conversation, searches for all but the last instance of the pattern [*int*, *dir*], and inserts the following two actions after the direction action: [`forward-through-int`, `forward`]. As an example, the plan [`forward`, `elbow`, `left`, `elbow`, `left`, `goal-F`] would become

Table 1.3. Query templates for plan patterns. The notation ⟨nth⟩ refers to a direction determiner generated based on how many direction concepts of the same type appear prior to *dir* in the current partial plan. The notation ⟨jth⟩ refers to an intersection determiner generated based on how many intersection concepts of the same type appear prior to *int* in the current partial plan. The first matching template applies when multiple templates match.

Pattern	Query-type	Query
☐	open-ended	'Could you tell me how to navigate to ⟨destination⟩?'
☐ *int* ...	single	'Which direction do I start out going?'
☐ *goal*	single	'Which direction do I start out going?'
`turn-around` ☐ ...	open-ended	'What do I do after turning around?'
dir ☐ ...	open-ended	'What do I do after starting to go *dir*?'
`turn-around` *dir* ☐ ...	open-ended	'What do I do after turning around and going *dir*?'
... `left` ... `left` ☐ ...	open-ended	'What do I do after I turn left (this being the ⟨nth⟩ left I take)?'
... `left` ☐ ...	open-ended	'What do I do after I turn left?'
... `right` ... `right` ☐ ...	open-ended	'What do I do after I turn right (this being the ⟨nth⟩ right I take)?'
... `right` ☐ ...	open-ended	'What do I do after I turn right?'
... `elbow` *dir* ☐ ...	open-ended	'Where do I go after the elbow (this being the ⟨jth⟩ elbow)?'
... *int* `forward` ☐ ...	open-ended	'What do I do after going through the ⟨jth⟩ *int*?'
... *int* ☐	open-ended	'What do I do after getting to the ⟨jth⟩ *int*?'
... int_1 ☐ int_2 ...	single	'When I am at int_1, which direction will I go to get to int_2?'

Table 1.4. Sample conversations from our trials.

Speaker	Utterance
Robot:	Could you tell me how to navigate to 345?
Person:	yeah, turn around go to the end of the hall and you'll take a lot to the bathroom.
Robot:	What do I do after getting to the first end of the hall?
Person:	you take a left at the bath.
Robot:	What do I do after I turn left?
Person:	app
Robot:	I did not understand that. What do I do after I turn left?
Person:	you go to the end of the hall.
Robot:	What do I do after getting to the second end of the hall?
Person:	turn right.
Robot:	What do I do after I turn right?
Person:	it'll be the third door on the left.
Robot:	Thanks for your help. Have a great day!
Plan:	[`turn-around`, `forward`, `end`, `left`, `end`, `right`, `goal-L`]

Robot:	Could you tell me how to navigate to 276?
Person:	yeah, turn around then turn right then your first left and then the door will be on your left.
Robot:	Thanks for your help. Have a great day!
Plan:	[`turn-around`, `forward`, `int-R`, `right`, `int-L`, `left`, `goal-L`]

Robot:	Could you tell me how to navigate to 1273?
Person:	yes, turn right.
Robot:	What do I do after I turn right?
Person:	
Robot:	I did not hear you. What do I do after I turn right?
Person:	and then turn right.
Robot:	What do I do after I turn right (this being the second right I take)?
Person:	find room 1273.
Robot:	Thanks for your help. Have a great day!
Plan:	[`forward`, `int-R`, `right`, `int-R`, `right`, `goal-F`]

[`forward`, `elbow`, `left`, `forward-through-int`,
`forward`, `elbow`, `left`, `goal-F`].

Follow_directions Substates

The FOLLOW_DIRECTIONS state has five substates: 1. MAKE_DECISION, 2. DRIVE_FORWARD, 3. ROTATE, 4. DRIVE_THROUGH_INTERSECTION, and 5. COMPLETE. FOLLOW_DIRECTIONS maintains a step counter, indicating the current step to execute. It enters the MAKE_DECISION substate first, initializing the step counter to the first step in the plan. When the current action is `forward`, it will enter the DRIVE_FORWARD substate wherein it drives ***forward*** until the subsequent intersection in the plan is found. In the example [`forward`, `elbow`, `left`, ...] the robot would drive ***forward*** until it detects an `elbow` intersection. As noted in Table 1.1, some of the intersection keywords are less specific and only contain an indication of what hallway trajectories one would expect to find at the specified intersection. For example, `int-L` would specify a registered intersection of any type that includes a hallway trajectory labeled with the qualitative direction left.

Once the specified intersection has been detected, the robot will return to the MAKE_DECISION substate to determine what it needs to do next. It the next step in the plan contains a direction action that requires rotation (i.e., `left`, `right`, or `turn-around`), the robot will enter the ROTATE substate. In this substate, the robot will rotate in-place by a specified amount according to the direction specified in the plan: `left` $\rightarrow 90°$, `right` $\rightarrow -90°$, and `turn-around` $\rightarrow 180°$. After rotating, the robot will return to the MAKE_DECISION substate. If the subsequent step is `forward-through-int`, the robot enters the DRIVE_THROUGH_INTERSECTION substate. As before, it requires that the intersection type change and the robot travel at least 2 m from where it rotated before it exits this substate and returns to the MAKE_DECISION substate.

Before each of the substates DRIVE_FORWARD, ROTATE, and DRIVE_THROUGH_INTERSECTION finish performing their task and return to the MAKE_DECISION substate, they increment the step counter accordingly. Eventually, the step counter will reach the goal action. When this occurs, MAKE_DECISION transitions to the COMPLETE substate. This

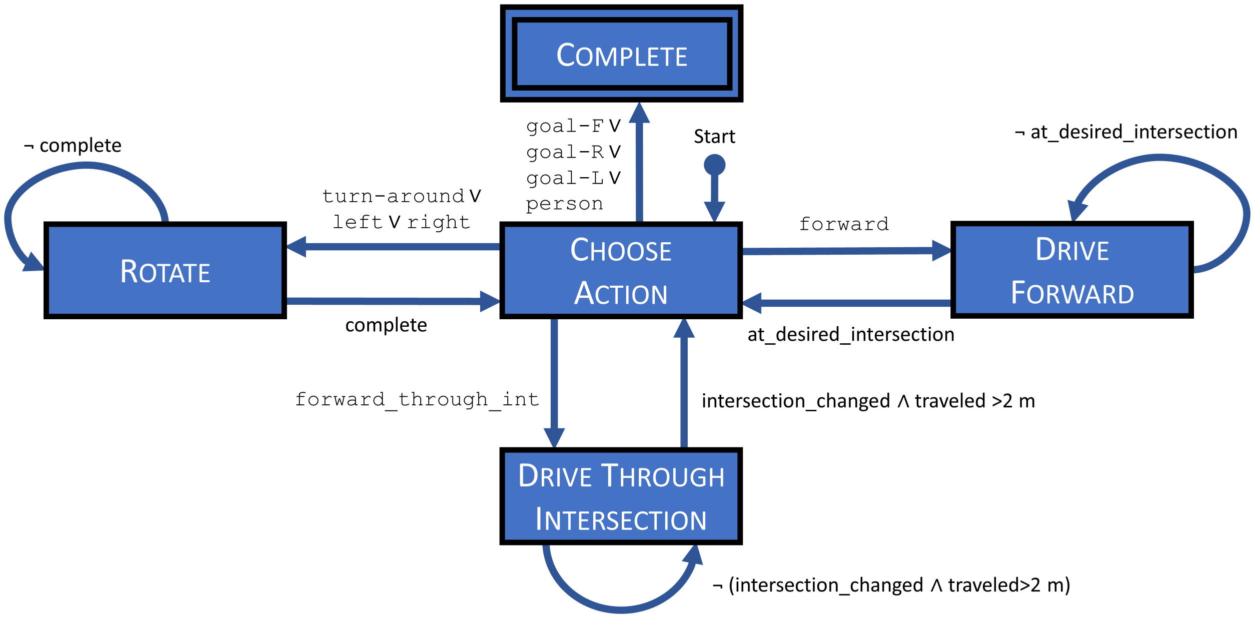


Figure 1.9. The FOLLOW_DIRECTIONS finite-state machine. Transition conditions in `Courier` represent the next step in the plan. All others represent transition conditions derived from sensor data.

is indicative of FOLLOW_DIRECTIONS having successfully executed the plan and the robot will transition to the next state. When the goal action is `goal-F`, `goal-L`, or `goal-R`, the robot has reached the same hallway as the goal and must look for it. Thus the robot will transition to the NAVIGATE_DOOR state. When the goal action is `person`, the robot has carried out as many steps as the previous person was able to provide and must now seek out a new person to ask for instructions. Thus the robot will transition to the WANDER state. A diagram of this FSM is shown in Figure 1.9.

Forward Driving Goals

When approaching a person, the robot may have moved to one side of the hallway and be in a suboptimal pose (similar to that described in Section 1.3.1). Additionally, because the conversation will have taken some amount of time, the person may have materialized as an obstacle in the quantitative map, directly in front of the robot. If the first action in the plan is `forward`, the robot may be unable to do so, even when using the technique described in Section 1.3.1, where we search with increasingly wider cone angles in the hopes of finding

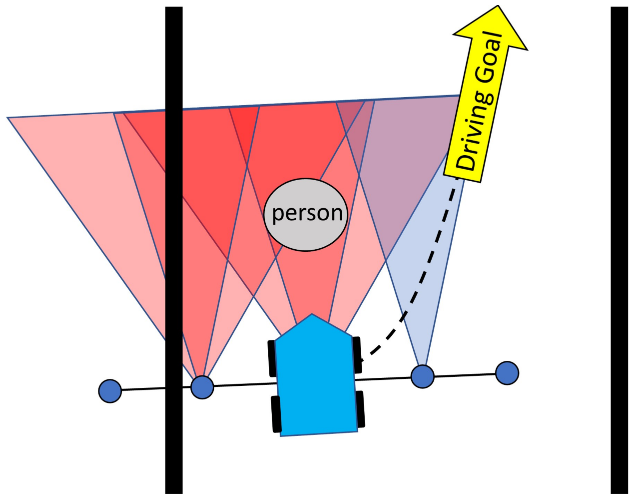
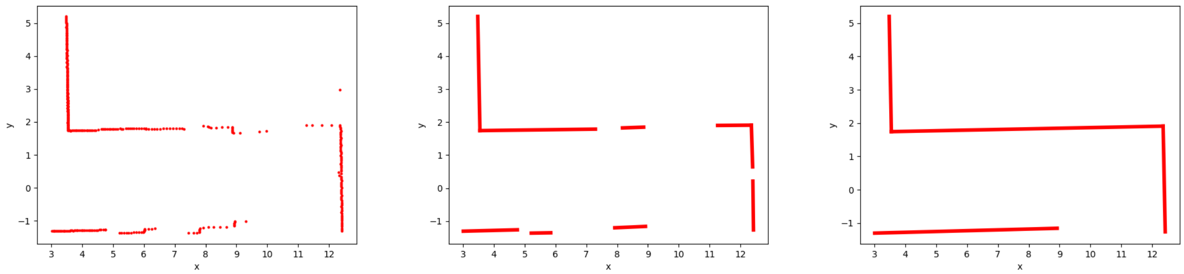


Figure 1.10. Example of how the robot determines a ***forward*** driving goal when the person it just conversed with has materialized as an obstacle in the quantitative map. The two transparent red cones coming from the robot indicate that no drivable trajectories are free at a cone angle of either $\pm15°$ or $\pm30°$. (To prevent this figure from becoming too cluttered, we ignore the cone angle of $\pm45°$). The two transparent red cones coming from the point that is 0.5 meters to the left of the robot indicate that there are no drivable trajectories at either $\pm15°$ or $\pm30°$. The blue cone coming from the point that is 0.5 meters to the right of the robot indicates that a drivable trajectory was found and it is used as the ***forward*** driving goal.

a median drivable trajectory. If no drivable trajectories are available from the robot's pose (which is often the case because of the robot's proximity to the person), we repeat the same search for a median drivable trajectory, but use positions that are horizontal to the robot's pose. Starting at the robot's current pose, we incrementally search left then right at distances of ±0.5 m, ±1 m, and ±1.5 m, with the expectation that at some horizontal position, we will be far enough away from the person to find a median drivable trajectory. We then use this median drivable trajectory as our ***forward*** driving goal. This helps to move us around the person and drive in the direction indicated. Note that this mechanism, also present in the Wander state, is the one we alluded to in Section 1.3.1. Figure 1.10 provides a visual explanation of this process.

While in the Drive_forward substate, the robot will continue driving ***forward*** until it either detects the intersection specified in the plan or reaches the end of the hallway. If



(a) Raw LiDAR points. (b) Extracted line segments. (c) Merged line segments.

Figure 1.11. Process for extracting walls from LiDAR data.

it reaches the end of the hallway without detecting the desired intersection, the robot will perform a single 360° spin to ensure that it truly is at a dead-end. If, after performing this spin, it has not detected the specified intersection, it indicates failure to carry out the plan and will return to the WANDER state.

1.3.5 Navigate_door

Once the robot completes execution of the plan with a goal other than `person`, it will be located in the hallway containing the desired door and transitions to the NAVIGATE_DOOR state to conduct a systematic search to find the door. The search procedure depends on accurate door localization and common-sense reasoning. We leverage several key characteristics of our environment: doors are all of a similar shape, each door has a door tag displaying the room number, and room numbers are consecutive odd on one side of the hallway and consecutive even on the other. This lets the robot inspect the doors in a logical fashion.

To detect doors, doorways, and elevators, we rely on four standard pieces of sensor information from the robot: 1. the camera image, 2. the raw LiDAR data, 3. the quantitative map, and 4. the robot pose. We use the LiDAR data to first determine regions in the image that can contain doors, then search these regions for door proposals, and ultimately assign a confidence score to each proposal. Because the LiDAR data provides information about obstacle distance relative to the robot, we can use the robot's position in the quantitative map to compute absolute locations for the door proposals.

Detecting Walls

We rely on 3D information from the LiDAR data to determine the walls. This data is represented as a collection of 2D (x, y) coordinates that represent the location of obstacles in the environment surrounding the robot. An illustration of such data can be seen in Figure 1.11a. We apply the Douglas-Peucker algorithm [23] to reduce the number of these points and establish line segments that can correspond to walls in the environment. We define each line segment as the 4-tuple (x_1, y_1, x_2, y_2) where (x_1, y_1) and (x_2, y_2) correspond to a pair of endpoints. The resulting line segments can be seen in Figure 1.11b. Due to occlusion or recession, this algorithm can produce disjoint line segments that correspond to the same wall. We employ two stages of clustering to merge these disjoint line segments into *walls*.

Because line segments that correspond to the same wall must share the same orientation, we cluster the line segments with hierarchical clustering based on their angles. We refer to these clusters as *orientation clusters*. To disambiguate line segments belonging to parallel but distinct walls, we perform a second stage of clustering. For each line segment within an orientation cluster, we rotate it by the negative of its angle so that the resulting line segment is parallel to the x-axis in a standard Cartesian coordinate system. The transformation is shown in Equation 1.1.

$$\begin{bmatrix} x_1 & x_2 \\ y_1 & y_2 \end{bmatrix} = \begin{bmatrix} \cos -\theta & -\sin -\theta \\ \sin -\theta & \cos -\theta \end{bmatrix} \begin{bmatrix} x_1 & x_2 \\ y_1 & y_2 \end{bmatrix} \tag{1.1}$$

Now, for each resulting line segment (x_1, y_1, x_2, y_2), its distance to the x-axis is equal to $y_1 (= y_2)$. Thus, we cluster all of these line segments based on this distance to distinguish between parallel walls. At the end of this step, our final clusters consist of line segments that belong to the same wall. For each of these clusters, we merge all of the line segments into a single line segment $(\hat{x}_1, \hat{y}_1, \hat{x}_2, \hat{y}_2)$ by taking the two most extreme endpoints of the line segments. Each of these merged line segments represents a wall in the robot's immediate environment. The final result can be seen in Figure 1.11c.

Now that we have the locations of the walls, we can create regions in the image that could contain doors. Because most doors, doorways, and elevators have a standard height of 2.2 m, we desire regions that are bounded above by this height and below by the ground level, 0 m. Knowing the camera intrinsics, K, and extrinsics, R and t, we can apply the transformation in Equation 1.2 to project the wall line segments onto the image at these height levels by setting h appropriately.

$$\begin{bmatrix} u_1 & u_2 \\ v_1 & v_2 \\ 1 & 1 \end{bmatrix} = K \begin{bmatrix} R & t \end{bmatrix} \begin{bmatrix} \hat{x}_1 & \hat{x}_2 \\ \hat{y}_1 & \hat{y}_2 \\ h & h \\ 1 & 1 \end{bmatrix} \tag{1.2}$$

Because we take real data from a moving robot as input, this projection might not always result in accurate pixel coordinates, which can cause complications when we search for edges that correspond to the tops of doors. Therefore, we incorporate a vertical distance tolerance of ± 15 cm for the top boundary. A visualization of the boundaries and their tolerances can be seen in Figure 1.12. Each pair of a top and bottom boundary is considered as a distinct wall region.

Generating Door Proposals

We use edge detection as a basis for generating door proposals, incorporating mechanisms that are robust to noisy edge detections which are prevalent in images obtained from a moving robot. We employ LSD (Line Segment Detector) to detect line segments in the image without any parameter tuning. Each line segment is represented as the 4-tuple (u_1, v_1, u_2, v_2). Inspired by Shi *et al.*, we quantize the line segments into several bins depending on their orientation. We do this to isolate the line segments that can potentially belong to a door, specifically the two posts and the top. Therefore, we keep any vertical line segments as possible door posts. Then, we separate any lines in the top half of the image into three bins based on whether the line segment has a positive slope, negative slope, or a slope close to zero, respectively. An example result is shown in Figure 1.13.

Figure 1.12. Projected walls with region boundaries. Green lines are the tops and bottom of walls projected at heights of 0 m and 2.2 m respectively, and the orange lines are the ±15 cm tolerances for the top.

Figure 1.13. Line segment detections, color coded by their orientation. Green lines are horizontal. Vertical lines are teal. Blue lines have a downward slope. Red lines have an upward slope.

Given the detected edges and the projected wall regions, we can search for possible doors in the environment. We iterate over each wall region to find doors in that region. First, any line segments, of any orientation, that lie outside of the horizontal range defined by the wall region are removed from consideration. Then, we iterate over pairs of vertical lines to generate door proposals. However, we only consider lines that are within a certain distance range of each other, approximately equivalent to the width of doors. We approximately localize the vertical lines in 3D space by projecting the raw LiDAR data to image coordinates, by Equation 1.2, and then computing the closest LiDAR point to each vertical line. That LiDAR point's 3D location is used as the approximate location for the corresponding vertical line. Then we cluster the vertical lines based on their 3D locations to create a reduced set of possible door posts represented as (u_1, v_1, u_2, v_2). Because $u_1 = u_2$ for vertical lines, we compute the average u_1 within a cluster to compute u_1 and u_2. Then we use u_1 to find v_1 and v_2 by computing the corresponding points on the top and bottom boundaries of the wall region, respectively. This creates a smaller set of lines that extend from the bottom boundary to the top boundary in the wall region on the image. Using these 3D locations of the lines in this reduced set, we compute a pairwise distance between them and only keep pairs whose distance is within the range $[0.5\text{ m}, 1.25\text{ m}]$.

Scoring proposals

We introduce a metric to determine how confident we are that a given pair of vertical lines corresponds to a door or elevator as some proposals could correspond to signs, posters, or wall structures. Akin to Del Pero *et al.*, we measure how much each proposal explains, or covers, the detected line segments in the image by computing the coverage of the vertical lines and top bar (the segment connecting the top points of the vertical lines) of the proposal. Multiple line segments may correspond to the same component of the door proposal, but could be disjoint and/or overlap due to noise or occlusion. First, we isolate the detected line segments that could correspond to the top of the door, by only retaining segments whose orientation matches the orientation of the top boundary of the wall region as well as lie within the top boundary's vertical tolerance. Then, using the same approach as for the vertical lines, we compute the approximate 3D location for each endpoint of each of the remaining line segments, and only consider line segments whose endpoints are within the horizontal range: 0.15 m left of the left vertical line and 0.15 m right of the right vertical line.

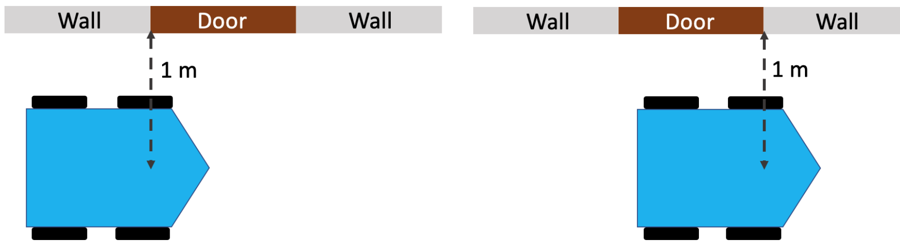
With these remaining segments, we can compute what fraction, c_{top}, they cover of the top horizontal area between the two vertical line segments. We take the same approach to the vertical line segments themselves. For each of the two vertical line segments, we find all of the original line segments that are within 0.2 m and compute how much they cover those line segments vertically, resulting in c_{left} and c_{right}. These three values are aggregated to compute a score for each door as indicated in Equation 1.3.

$$score = \frac{c_{left} + c_{right} + c_{top}}{3} \tag{1.3}$$

Rather than have binary decisions about whether or not a proposal is a door, a confidence score can allow an online system to be adjusted to a desired false-positive rate. In our system, we use a confidence score threshold of 0.75, only considering detections with a score greater than or equal to this.

Localizing detections

Using the 3D locations of the door posts, we describe the location of the door as the tuple $d = (x_{min}, y_{min}, x_{max}, y_{max})$ where min and max correspond to the closer and further door posts respectively. This allows for driving to both sides of the door as the door tag might be on either side. Because this process is performed in an online fashion on continuous image frames received from the camera, the same door can be detected multiple times at different time steps. Therefore, detections from different time steps are hierarchically clustered by the Euclidean distances between their center locations. For accurate clustering, we rely on additional information from the LiDAR data to accurately cluster door detections. For each door bounding box, we find all the 3D points whose corresponding pixel coordinates are within its bounds. We then take the median of these 3D points as the door detection's



(a) The close side of a door.　　(b) The far side of a door.

Figure 1.14. Driving-goal positions for a door.

center location. This allows for computing accurate door coordinates and filtering out false positives for driving-goal generation.

Driving-Goal Generation

For each of the resulting clusters, we compute the average locations of the door posts to create the final driving goals for that door. Only clusters whose size is >3 are considered. These clusters are classified as being on the *right side* of the hallway or the *left side* based on a comparison between the robot's trajectory down the hallway and the door locations (relative to the beginning of the hallway). Within these classifications, the clusters are sorted by increasing distance relative to the beginning of the hallway. This allows for driving to a specific door. For example, driving to the third door on the left can be achieved by retrieving the coordinates of the door whose index is two, by zero-indexing, among doors classified as being on the left side of the hallway. The robot uses the coordinates of the door posts to create two driving goals so that it can position itself appropriately to read the door tag. Figure 1.14 illustrates the two desired positions. The door's angle $\theta = \tan^{-1} \frac{y_{max}-y_{min}}{x_{max}-x_{min}}$ is computed and used to determine a driving goal that is 1 m perpendicular to the door and with a target orientation orientation matching that of the door. Upon arrival at this driving goal, the robot pans its camera in the direction of the door to read its door tag with Google's Optical Character Recognition API.

Common-Sense Navigation

Generally, the robot will drive straight down the center of the hallway, using the `forward` command, and only drive up to a door once it is within 3 m. This behavior results in a good view of doors up ahead. If the index and/or classification of the desired door is unknown, common-sense knowledge about room labeling in an office environment is leveraged to guess what they are, based on the first door tag that is read. This can enable potentially more efficient goal-finding than exhaustive search alone. This common-sense knowledge is based on assumptions about doors being labeled in increasing or decreasing order, both numerically and alphabetically (in the case of a letter suffix), and even and odd doors being on different sides of the hallway. Given the door tag of the first door the robot inspects, it will use parity to determine what side the goal door is on. If the goal door is on the same side, it will compute the expected index of the goal door and drive down the hallway until it detects that door. Otherwise, it will start with the first door on the other side of the hallway. For example, if the desired door is room 335 and the first door tag read is 331, then room 335 will be two doors down on the same side of the hallway. Alternatively, if the first door tag is 330, only doors on the other side of the hallway will be inspected.

A number of mechanisms are incorporated to allow the robot to find the goal door in spite of any of these assumptions being invalidated. Firstly, at any given point, all the door tags read so far are used to determine whether or not the goal door was missed. This is done using the range and trend (whether they are increasing or decreasing) of the door values to check whether the robot should have come across the door in its path. If the door has not been missed, the robot will continue to inspect doors on the current side of the hallway until it reaches the end of the hallway. In both of these cases, the robot returns to the start of the hallway and begins an exhaustive search of all doors in the hallway to find the goal door. If the robot still has not found the goal door after this exhaustive search, it returns to the WANDER state.

Although object detection networks like YOLO and Faster-RCNN [26] are able to detect people well, and as such we use YOLOv3 in our APPROACH_PERSON behavior, they do not detect doors well. To demonstrate that our method of door detection is more reliable, we ran

Table 1.5. YOLO vs. our door-detection method

	doors approached	total doors	success rate	false positives
YOLO	20	41	48.8%	1
ours	36	41	**87.8%**	2

a set of experiments with the NAVIGATE_DOOR behavior. We ran this behavior in isolation in four hallways of EE and recorded the number of doors it successfully drove up to. Then, inside of NAVIGATE_DOOR we replaced our door-detection algorithm with a YOLO model trained on a dataset of doors [27] and reran the same set of experiments. We also recorded the number of false positives, where the robot drove up to a location that did not have a door. The results in Table 1.5 show that our approach allows the robot to discover a much larger percentage of doors, 87.8% vs. 48.8%, than when YOLO is used to detect the doors.

1.4 System Evaluation

Our system design was developed and validated in three buildings (EE, MSEE, and PHYS). In order to evaluate the generalizability and robustness of our system's performance, we performed 52 trials in 4 distinct floors in each of three new buildings (HAMP, KNOY, and ME) it had not been deployed in before. With three exceptions described below, we froze our software after development before the evaluation trials. For these trials, we recruited 13 volunteers to provide directions to the robot.[3] Volunteers were untrained university students who had never interacted with the robot before nor were aware of what it understood or was capable of. Each volunteer assisted with 1–6 trials. We demonstrate in our trials that this method is able to successfully navigate in the real world using instructions obtained via interaction between a real robot and untrained humans who speak freely and have not been coached to use a specific language subset.

[3]↑Our original intent was to not have any volunteers but instead to have the WANDER state find people naturally occurring in the environment to model real visitors soliciting help in finding their way from locals. COVID-19 prevented us from doing this.

1.4.1 Experimental Setup

For each trial, the robot was placed on a floor of one of the three new buildings and given a random door number as the goal in the form of a string. This was done by speaking a command to the robot in the form of "Husky, find X", where X is a room number. We supported giving any command that started with the word "Husky", included the word "find" anywhere in the command and contained a word that met the following regular expression:

```
\s[a-zA-Z]{0,1}[0-9]{1,4}[a-zA-Z]{0,1}\s
```

which we would use as the goal door string. For example, we supported any of the following commands: "Husky, find room X", "Husky, can you find X", or "Husky, will you please go and find a room for me. It is X." For each trial, the robot was placed on a floor of one of the three new buildings and given a random door number as the goal. The volunteer was instructed to stand in a location where the robot would be allowed to WANDER for some time before seeing them. If the robot re-entered the WANDER state at any point in the trial, the volunteer was instructed to relocate to a position that the robot would come across after being allowed to wander for some time. A trial was considered a complete success if the robot reached the correct goal door and read its door tag. Otherwise, it was considered a failure. Trials that were manually terminated early were also considered a failure. We manually terminated a trial if the robot got stuck and could not move after manual intervention. We also manually terminated a trial if the FOLLOW_DIRECTIONS or NAVIGATE_DOOR behaviors could not succeed within two attempts. For the FOLLOW_DIRECTIONS behavior, an attempt was when the robot drove past the intersection it was supposed to turn at, but did not detect it. For the NAVIGATE_DOOR behavior, an attempt was when the robot drove past the goal door but did not detect it. A trial was considered a complete success if the robot reached the correct goal door and read its door tag. A trial was concluded and subsequently declared a failure if the robot did not reach the goal after several attempts. In some trials, some minor manual intervention was used to partially rotate the robot or prevent it from crashing into a wall to allow the trial to continue.[4]

[4]↑An online appendix at https://github.com/qobi/amazing-race contains the floor plan of each building and describes each trial in detail, including the building, floor, volunteer, goal, a transcript of the dialog, the

Table 1.6. Trial results.

Building	Successes	Total	Success rate
HAMP	11	17	64.7%
KNOY	13	16	81.3%
ME	16	19	84.2%
all	40	52	76.9%

Table 1.7. Behavior success rate.

Behavior	Total instances	Average instances per trial	Success rate
Wander	390	7.5	0.99
Approach_person	385	7.4	0.47
Hold_conversation	183	3.5	0.86
Follow_directions	182	3.5	0.68
Navigate_door	99	1.9	0.40

1.4.2 Trial Results

The number of successful trials is shown in Table 1.6 on a per-building basis. Overall, the system was able to successfully reach the given goal in 76.9% of the trials. Of the 12 floor plans tested, the robot was able to successfully find the goal at least twice on every floor and usually three or more times (8/12 floors). This result highlights the ability of our system to generalize to new floor plans. We provide an online appendix which shows the environmental complexity of these floor plans as well as the diversity of natural-language dialogue that the robot is able to handle. It shows how distinct the floor plans of the test buildings are from the buildings that our system was developed in.

One of the characteristics of our system is its ability to recover from individual behavior failures. Of the 52 trials, only 3 (≈5.7%) succeeded without any behavior failures. When failures do occur, either from shortcomings of the individual behaviors or incorrect instructions from volunteers, the robot is able to recover and make multiple attempts at its task. Table 1.7 shows the success rate of each behavior. Despite there being multiple instances of each behavior due to failure, the robot is ultimately able to reach the goal door with the subsequent recovery from the individual behavior failures. This ability effectively increased the success rate by a factor of 13.3. Our system handles a very comprehensive space of natural language, as well as a comprehensive space of visual complexity, both in terms of image and SLAM data. Figure 1.15 shows the complex and diverse floor plans that our system was developed and tested in. Figure 1.16 show examples of the visual complexity our robot is able to handle. Table 1.8 show some examples of the complex natural language dialogues that our robot was able to handle.

We assessed how often the robot was able to perform the following three behaviors successfully in succession: HOLD_CONVERSATION, FOLLOW_DIRECTIONS, and NAVIGATE_DOOR. We computed three metrics in Table 1.9. We first measured how often the robot produced a correct plan from dialogue. We then measured how often the robot produced a correct plan from dialogue and executed it correctly. Finally, we measured how often the robot

plan extracted, the map constructed with the route taken, indication of success or failure, and a description of the failure reason upon failure. The appendix also includes three videos depicting three complete trials, one in each building.

Figure 1.15. The floor plans of each building. Our training set consists of EE, MSEE, and PHYS. Our test set consists of HAMP, KNOY, and ME. Areas of the map that are green represent areas that the robot can drive in. Areas of the map that are red represent areas that the robot cannot drive in. These include carpeted areas and hallways that are narrower than the robot's circumscribed radius.

Table 1.8. Sample conversations from our trials.

Speaker	Utterance
Robot:	Could you tell me how to navigate to 1067?
Person:	turn left and go straight until the end of the hallway, then turn right and go straight to the end of the hallway and then
Robot:	What do I do after getting to the second end of the hall?
Person:	and then
Robot:	I did not understand that.
Robot:	I understand verbal directions, like turn-around and turn left. I understand intersections like an elbow or end of hallway. I understand words like goal or destination.
Robot:	You can give me instructions in small chunks or all at once. If I misunderstand something, you can tell me that my interpretation was wrong or incorrect and I will back up one step. If I totally misunderstand, you can ask me to start over.
Robot:	What do I do after getting to the second end of the hall?
Person:	start over
Robot:	Sorry about that. Let's start over.
Robot:	Could you tell me how to navigate to 1067?
Person:	turn left then go to the end of the hallway then turn left again and go to the end of the hallway and then turn right and keep going straight until you see the door, which is on the left.
Robot:	Thanks for your help. Have a great day!
Plan:	[`forward`, `int-L`, `left`, `end`, `left`, `end`, `right`, `goal-L`]
Robot:	Could you tell me how to navigate to 453?
Person:	turn around go forward and the door will be directly in front of you.
Robot:	What do I do after turning around and going forward?
Person:	the door will be in front of you.
Robot:	I did not understand that.
Robot:	I understand verbal directions, like turn-around and turn left. I understand intersections like an elbow or end of hallway. I understand words like goal or destination.
Robot:	You can give me instructions in small chunks or all at once. If I misunderstand something, you can tell me that my interpretation was wrong or incorrect and I will back up one step. If I totally misunderstand, you can ask me to start over.
Robot:	What do I do after turning around and going forward?
Person:	the door will be stopped at the inn at the intersection and the door will be there.
Robot:	Thanks for your help. Have a great day!
Plan:	[`turn-around`, `forward`, `goal-F`]

EE3

MSEE2

PHYS2

HAMP1

KNOY0

ME0

Figure 1.16. Images from one floor of each building.

Table 1.9. Success rate of the robot at extracting a correct plan from conversation, following that plan, and finding the goal door.

	Successful instances	Total Instances	Success Rate
correct plan	125	183	68.3%
correct plan and correct execution	92	183	50.3%
correct plan, correct execution, and goal found	35	183	19.1%

produced a correct plan from dialogue, executed it correctly, and found the goal door. Out of all of the conversations that the robot had with a volunteer, it correctly adhered to their instructions and found the goal door 19.1% of the time (see Table 1.9). While the success rate of successfully completing all three of these behaviors in sequence is relatively low, we are able to increase our per-trial accuracy by allowing the robot to fall back to WANDER in the FSM and trying again. However, improving each of these individual behaviors would reduce the robot's need to fall back to WANDER and allow it to find the goal faster.

We conducted a post-trial survey with our volunteers. They were asked to read a statement and rate it on a 5-point Likert scale from 1: absolutely disagree to 5: absolutely agree. There were a total of 25 statements; the first 5 were provided by us; the last 20 came from Weiss *et al.*, who performed a survey of people who interacted with their robot after conducting an experiment similar to ours. Table 1.10 contains the statement, number of responses, as well as the mean and standard deviation for each statement. In general, the volunteers indicated that they were able to communicate with the robot, it was able to understand the navigational instructions they gave it, and it asked appropriate follow-up questions. They also indicated that it was helpful when the robot told them what it understood.

After running the trials, we further considered the scenario when no person could be found in the environment. To support this scenario, we integrated portions of NAVIGATE_DOOR into WANDER. While the robot is wandering around, it simultaneously detects doors and looks for people. If it sees a door and there are no approachable people, it will drive up to the door and inspect its door tag. If it sees an approachable person, it will approach that person and ask for navigational instructions to the goal. To test out this new capability, we ran 6 trials in one of the test buildings (KNOY). In 3 of the trials, the robot successfully located the door before finding any person in the environment. In 3 of the trials, the robot

Table 1.10. Survey results from our participants. All statements had to be rated on a 5-point Likert scale from 1: absolutely disagree to 5: absolutely agree, which mimics Weiss *et al.* Statements 1–5 were provided by us. Statements 6–25 came from Weiss *et al.*

Index	Statement	Number	Mean	Standard Deviation
1	I was able to communicate with Hosh using natural language	11	4.18	0.75
2	Hosh was able to understand the navigational instructions I gave it	11	3.64	0.50
3	Hosh asked appropriate follow-up questions	11	3.82	0.98
4	It was helpful when Hosh told me what it understood	11	4.55	0.69
5	Communicating with Hosh was easy	11	3.55	0.69
6	I am afraid of making errors if I am interacting with Hosh	11	3.09	1.22
7	I could work with Hosh if I got a good training	11	4.64	0.50
8	Under time pressure I could never be successful in dealing with Hosh	11	2.36	1.29
9	The interaction with Hosh is easily understandable	11	4.27	0.65
10	I could solve all occurring interaction problems with Hosh by myself	11	3.45	0.82
11	It will be easy to use Hosh	11	3.64	0.67
12	It will be easy for me to skilfully interact with Hosh	11	3.91	0.54
13	Hosh reacted to my behavior	11	2.82	1.25
14	The interaction with Hosh should be a give and take	11	3.82	0.98
15	The interaction with Hosh was like a give and take	11	3.18	0.60
16	Hosh and me would compose a good team	11	3.36	0.81
17	Hosh needs my support to carry out its task	11	3.91	0.94
18	Hosh will be useful to me	11	2.91	0.70
19	The effort to carry out tasks together with Hosh will be immense	11	2.36	0.67
20	I can imagine to carry out tasks faster with Hosh	11	2.91	0.83
21	I do not think it is necessary to introduce Hosh to the everyday working life	11	2.73	1.01
22	Hosh could support me in everyday transactions	11	3.45	0.69
23	Hosh will ease burdensome tasks	11	3.45	0.69
24	I would not like to work together with Hosh	11	1.91	1.04
25	I think it is a good idea to “use” the Hosh robot	11	3.91	0.70

found a person, whom it asked for navigational instructions, and followed those to the goal hallway and then found the goal door successfully.

1.4.3 Observations and Improvements

As the trials were conducted, many observations were made about each behavior's interaction with the complex environments in which the robot was evaluated. Although the system recovers from most individual behavior failures, reducing the number of these would lead to faster and more efficient goal finding. In this section, we discuss these observations and how they can potentially be used as points of improvement in future work.

Approach_person

We observed several challenges associated with successfully and safely navigating to a person to ask them for directions. YOLOv3 cannot distinguish between people and pictures of people on the walls. In some trials, this led to the robot repeatedly driving up to walls containing pictures of people and attempting to initiate HOLD_CONVERSATION. Potential solutions include using body size as a prior to determine whether the detected person has the correct size for the detected distance or incorporating body-pose estimation to help make the distinction. Also, constraints can be applied to the locations of person detections, such as having to touch the ground, to eliminate detections of people on posters and signs from consideration.

Another point of difficulty was detecting and localizing people that are far away. People that are small in the camera's field of view are more difficult to detect. Even if they are detected, the bounding boxes are small, thus having fewer corresponding 3D LiDAR points, making it difficult to localize, track, and classify person tracks accurately. Similarly, consistent, accurate localization of people was also difficult when they were occluded by objects such as drinking fountains or trash cans, or when they were positioned close to a wall. These objects would often overlap with the person detections, which would affect the localization of the person themselves as some 3D LiDAR points overlapping the box would actually correspond to the object or wall instead of the person. These issues created tracks that would rapidly switch between being approachable and not approachable, which caused the system to correspondingly switch between Wander and Approach_person rapidly. This explains the high number of instances of Wander and Approach_person on a per-trial basis as well as the low success rate of Approach_person. The robot would eventually reach the person, but would sometimes do so in an inefficient manner.

Hold_conversation

The most common observation made when volunteers were providing instructions to the robot was their use of terminology and hand gestures that the robot did not understand. Some volunteers tried to describe directions using gestures, distances, angles, and objects. There is opportunity for developing a more generalizable and robust conversation engine that is capable of handling the diversity possible in a set of navigation instructions. A quantitative map with richer semantic information could support dialogue and navigation related to concepts such as objects, colors, or landmarks. Additionally, the plan structure could also be expanded to include distances, angles, objects, landmarks, text, arrows, and signs.

Aside from this, there was one trial where Google's Speech-to-Text API returned a non-ASCII character (°), which was not supported by our message passing, and caused our Hold_conversation behavior to die. We manually restarted the Hold_conversation behavior and the trial continued to success.

Follow_directions

Many of the building floors featured complexities not seen during development, such as open spaces and variation in both hallway width and turn angles. In particular, one building had hallways that branched off at 45° angles. This caused some confusion both in processing and executing the navigation instructions provided by the person. Some people would indicate that the robot should drive 'straight' when it got to those intersections while others indicated the robot should 'turn.' When told to drive 'straight,' Follow_directions would detect that it could no longer drive ***forward*** after reaching the end of the hallway

Table 1.11. System Trials.

group	live trials	# of trials	success rate
ours	yes	52	76.9%
Birmingham	yes	46	67.4%
Munich	yes	1	100.0%
UW	no	14	71.4%
Cornell	yes	46	76.1%

where the hallway would veer off at 45° and thus indicate failure. When told to turn 'left,' FOLLOW_DIRECTIONS would not detect the left turn and also indicate failure. An area of future research includes developing adaptive methods for determining novel intersection types in new indoor environments.

In a different building, the robot could not detect a particular narrow hallway opening (≈1.5 m wide when most hallways were 2–4 m wide) due to a hyperparameter corresponding to the hallway width. We adjusted it accordingly after the first 4 trials, and left it unchanged for the remainder of the trials. Dynamically detecting characteristics like this in novel environments is an area of future research.

Finally, as the robot is driving around, it is building a quantitative map in real-time and relies on map updates to determine where there are registered intersections. When the conjoining hallways are narrow, the robot sometimes does not have enough data from its LiDAR to have built a reliable-enough quantitative map to detect the intersection correctly. Instead of detecting the intersection, the robot drives past it unaware. Planned future work includes using a neural network to improve the detection of intersections and execution of the FOLLOW_DIRECTIONS behavior in spite of an incomplete map.

Navigate_door

The NAVIGATE_DOOR behavior exhibited difficulty in finding the goal door in some situations. First, many doors were occluded or only partially visible in the robot's field of view and the door detection method does not support detecting doors in these scenarios. A point for future work could be to train a neural-network-based object detector to robustly

Table 1.12. Comparison with Related Work.
(a) Test environment.

Group	Number of test environments	Test environment	Train/Test environments are different	Map provided
ours	12	floor of building	yes	no
Birmingham	1	11 m^2 with hallway, 2 offices, and 1 conference room	unspecified	no
Munich	1	downtown Munich	unspecified	no
UW	1	floor of building	yes	yes
Cornell	1	1 km^2 outdoor facility with 12 buildings	unspecified	no

(b) People interaction and direction-giving.

group	interacts with live people	untrained people	multi-turn conversation	open-ended conversation	spoken dialogue
ours	yes	yes	yes	yes	yes
Birmingham	yes	unspecified	yes	yes	no
Munich	yes	yes	yes	no	no
UW	no[a]	yes	no	no	no
Cornell	no[b]	no	no	no	no

[a]↑This paper takes, as the input to their trial, a single, open-ended text instruction from an untrained user.
[b]↑This paper takes, as the input to their trial, a single, open-ended text instruction from a predefined grammar called TBS.

(c) Directions, goals, and recovery.

group	follows directions	can detect when directions are wrong	robot detects goal independently	recovers from failure
ours	yes	yes	yes	yes
Birmingham	no	no	yes	yes
Munich	yes	no	unspecified	yes
UW	yes	yes	unspecified	yes
Cornell	yes	no	yes	no

detect doors in spite of occlusion. Additionally, the network could be trained to detect other objects as well to enable the robot to find something beyond a room.

Another observation is that although the common-sense reasoning led the robot to efficiently find the goal in some trials, in many cases, it was disrupted by both missed and false-positive door detections. In these scenarios, the Navigate_door behavior resorted to exhaustive search to find the goal door and was typically able to locate it. However, on a larger scale, exhaustive search would be impractical so a more robust reasoning and navigation system is an area of future work. To help remedy missed doors, we adjusted the threshold for the hierarchical clustering of door detections from 0.25 m to 0.5 m to create fewer but larger clusters. This parameter change was done after the first 7 trials and kept for the remaining trials.

We also found two issues involving using Google's Optical Character Recognition (OCR) to read door tags. The first issue involved misreads, which happened twice in our trials. The robot drove up to the correct door, but read "goio" instead of "g010." The same thing occurred with "g055" being misread as "go55." A simple string replacement for commonly mistaken characters/digits would solve this problem (e.g., o $\Rightarrow$ 0, i $\Rightarrow$ 1, S $\Rightarrow$ 5, etc.). The other challenge with OCR was distinguishing *any text read* from text read from the door tags. In one failed trial, the robot drove up to a door and read an advertisement about an event occurring in a particular room. The room it was referring to was our goal room (which happened to be about 2 m away, across the hall). In another failed trial, the robot failed to detect the goal door, drove up to the subsequent door, but still read the goal door's door tag. In both of these cases, the robot erroneously claimed success. Being able to distinguish between a door tag and other signage or text, as well as being able to assign a single door tag to a single door, would alleviate this issue. We counted the two trials with OCR misreads as successes (since the robot did successfully accomplish all of the behaviors leading up to finding the correct door) and counted the two trials with the advertisement and the adjacent door as the mistaken goal as failures.

Lastly, after the first 24 trials, we discovered an issue where the robot was not clearing its set of detected door locations in between different instances of the Navigate_door behavior within the same trial. This was fixed for the remaining trials as the intention was to only reason about doors in the current hallway as that is where the goal should be located. However, in future work, retaining and reasoning about all detected doors could enable the robot to find the goal without directions as well as return to rooms it has already visited if given instructions to do so.

1.5 Related Work

Several research groups have constructed a system that accomplishes a comprehensive task similar to ours. We first compare and contrast our system to these other comprehensive systems. Then, we compare our individual behaviors to other research that has focused on similar behavior-specific tasks.

1.5.1 Systems

In Table 1.11 and Table 1.12, we compare our system to those of the following groups: Birmingham [1], whose task is to find an object within a small office environment; Munich [2], whose task is to navigate the streets of Munich and reach a downtown plaza; UW [3], whose task is to follow natural-language directions to an office on a building floor; and Cornell [4], whose task is to follow a natural-language directive (in a specified grammar) to an outdoor goal location.

All five groups perform their task in the real world. Our results, along with Birmingham and Cornell, are based on a large number of trials, which helps to support the success rate reported. Our success rate is comparable to the other groups, but, as we will show in the subsequent tables, we also take on more challenges.

Table 1.12a describes the test environment and how many distinct environments each group tested on. We test in 12 distinct environments, which is much larger than any other group. Having multiple trials in a large number of distinct environments provides some additional support to the generalizability of our system. We also stress that our 12 environments are distinct from the environments we trained on and developed for. Doing so prevents us from "training on (or developing for) the test set," which carries the risk of overfitting to a particular environment and biasing the generalizability of a system's performance. Like most of the other systems, we operate in an unknown environment, without a map.

Table 1.12b compares how the robot interacts with people and gets directions. Unlike UW and Cornell, which have a single set of directions provided to the robot at the beginning of the trial, we have to find people in our environment that we can ask for directions. These people are untrained and unfamiliar with what the robot understands or is capable of. Along with Birmingham, we allow for open-ended conversation, which increases the diversity of responses. Finally, our conversation takes place via spoken dialogue which no other systems do. (Munich uses a GUI to get responses, while Birmingham uses text input.) These differences make our task considerably harder than those of other systems.

Finally, in Table 1.12c, we point out a few additional ways in which our task sets us apart from the other systems. Our robot has to follow directions, rather than simply exploring until the goal is found. Exploring may work when the environment is small, but becomes intractable as the environment increases in size. When the directions it has are wrong, both our robot and UW's are capable of detecting this. In UW's case, they backtrack and recompute the next-most-likely path to the goal. In our case, we simply revert to WANDER and seek out another person to ask for directions. Our robot is capable of discovering and detecting the goal independently, while it is unclear whether all systems are capable of that ability. Lastly, like most other systems, we are able to recover from individual behavior failures and continue working towards the goal.

1.5.2 Behaviors

As described earlier, our system combines a number of behaviors, each of which solve one of the sub-tasks presented by The Amazing Race™ task. The first is autonomous navigation in an indoor environment. Barrett *et al.* employ a learning mechanism to acquire the semantics for direction keywords such as 'left of,' 'right of,' and 'behind' and then use these semantics for planning and describing robot paths. Because our focus was to design a complete system for navigation in an unknown environment, we predefined a similar set of directions for the WANDER and FOLLOW_DIRECTIONS state and used a novel method for determining whether they are available.

The next crucial sub-task is to find a person and receive and interpret directions from them. The Jackrabbot lab at Stanford has developed a robot that complies with social conventions such as understanding [29] and predicting [30] human trajectory in crowded scenes. When looking for and approaching people, our robot also complies with expected conventions such as determining which people are approachable, introducing itself as it approaches, and not invading personal space when having a conversation.

Bauer *et al.*, Bauer *et al.*, Bauer *et al.* define a set of heuristic rules and a complex finite-state machine to obtain specific pieces of information from a person through a touch screen interface on their robot. Our system, however, takes full spoken natural-language utterances as input and extracts instructions from them. Oh *et al.* and Boularias *et al.* require a specific syntactic structure of natural-language commands called Tactical Behavior Specification to simplify the parsing problem. Our parsing method does not depend on such a strict grammar because it would be impractical for interacting with people unfamiliar with the robot. Kollar *et al.* train a model to extract spatial-description clauses from natural-language directions to determine the corresponding path in the environment. Our method does not rely on training.

A single person's response may have insufficient, vague, or incorrect information. Thomason *et al.* present a dialogue system that can compensate for this issue by generating queries about missing pieces of information, specifically the action, patient, recipient, or some combination of them. Our dialogue system does not predefine what or how much information we seek and instead dynamically infers and generates queries for what information is missing.

There has been prior work [34]–[37] that, like our work, has also focused on achieving multi-turn dialogue understanding on a physical robot. However, this work operated within smaller and simpler environments than our work; we train and evaluate our system on several large unmodified office environments. Banerjee *et al.* collected data from a physical robot to train their algorithm to follow natural-language instructions. However, evaluation of the approach was only performed in simulation. Blukis *et al.* demonstrated their algorithm on a quadcopter, but the environment was very small with just a few objects on a green surface.

The environment was only varied by placing different objects in different positions. Lukin *et al.* constructed a framework to control a physical robot with spoken language, but it was only tested in a simulator in which the virtual robot could execute navigation commands from a finite discrete set. Thomason *et al.* proposed a method to allow a robot to learn object-related concepts through dialogue. Only a single demonstration on a real robot was done in a single room with several objects on a table.

In order to navigate in the environment, given the natural-language directions, the robot needs a semantic map of the environment. Hemachandra *et al.* rely on AprilTag fiducials to classify regions in the environment for autonomous navigation. However, this prevents practical operation in any unknown environment because it would have to be labeled with these fiducials. Sünderhauf *et al.* explore new areas and create a semantic map through the classification of sequences of image frames as places (e.g., office or kitchen). Pangercic *et al.* create semantic object maps in a kitchen using a RGBD camera that enables their robot to perform fetch and place tasks. We use the occupancy grid, door detection, and text recognition to make a semantic map of an unknown environment.

Many approaches, including ours, rely on edge or line segment detection as a basis for finding door posts. Stoeter *et al.* detects door frames by detecting vertical stripes in the image, but other objects or parts of the background could be responsible for vertical edges. More complex approaches look for edges in the image that form an upside-down U-shape to create door detections [24], [41]. These methods depend on high quality detected edges, which are not necessarily possible to obtain especially on images captured from a camera on a robot navigating in areas with varying, possibly poor, lighting conditions. Our method incorporates noise-tolerance mechanisms that simultaneously allow for door detection in spite of poor edge detection and measuring the confidence of those detections.

In the case of Shi *et al.*, only U-shapes that intersect the corridor lines are considered doors in order to avoid false positives caused by signs or posters. These corridor lines are obtained by finding the vanishing lines that have the most intersections with the bottoms of vertical lines. Olmschenk *et al.* employ a similar method to finding corridor lines by detecting the vanishing point and vertical lines in an image to determine where the wall meets the floor. This approach is impractical for two reasons: 1. walls orthogonal to the viewpoint are ignored and 2. patterns in the image, such as floor tiling, can induce strong edge detections that can be confused for corridor lines. We incorporate 3D information from LiDAR data to reliably and generally detect wall regions and avoid this issue.

Hanheide *et al.* employ a knowledge hierarchy that enables reasoning over known pieces of information in order to perform task planning, execution, and task-failure explanation. Because our robot has no prior knowledge about the environment, we extract information such as driving directions and door locations in an online fashion to execute navigation instructions and plan paths to potential locations of the goal.

1.5.3 Vision-Language Navigation

There has been considerable prior work on vision-and-language navigation (VLN). Some of this work [9]–[13], [43] trained and evaluated VLN models on the Room-2-Room (R2R) dataset [9]. This dataset consists of natural-language text instructions paired with corresponding trajectories in a simulated indoor environment. These trajectories are sequences of vertices in a discrete graph, where each vertex has a panorama of images to represent the view at that vertex. Our work differs significantly from all of this work in a number of key ways.

First, in the R2R simulator, robot position is represented as a vertex in a discrete graph and visual information, although from real images, is noise-free and deterministic at each vertex. In contrast, rather than just repeatedly outputting one of a small number of adjacent graph vertices to eventually reach a goal vertex, we address a more complex problem: controlling a physical robot in the real world with a noisy continuous position and action space and noisy continuous observations. While Anderson *et al.* trained in a simulated environment, it tested both in simulated environments and on a real robot in real environments. When the navigation graph was known *a priori*, performance in the real environment was comparable to that in simulation, however, results were very poor when the navigation graph was unknown and waypoints were predicted on the fly. Our system is able to successfully execute navigation instructions without a known map of the environment, just with the SLAM map that is built as the robot drives.

Second, the above prior work took single-turn text as input. Our system interacts with a person in multi-turn spoken dialogue and is designed to be robust in the face of noisy speech recognition. Such dialogue is crucial for clarifying potentially ambiguous instructions. Thomason *et al.* considers a cooperative VLN task and provides a new dataset of navigation trajectories along with dialogue between a navigator and an oracle. Although the dataset is intended to support research into language generation, they do not provide a method that generates language. They only consider the task of navigation from dialogue history which involves producing navigation actions given a dialogue history between two humans as input.

Third, our system has capabilities beyond those of these VLN approaches. These VLN methods assume that natural language is a given input at execution time, but in the real world, a person may not be available in the immediate vicinity. Our system is capable of autonomously exploring the environment, detecting approachable people, and driving up to them to ask for directions. Additionally, it makes use of of common-sense knowledge to search for doors, which leads to both more efficient and more human-like behavior.

Some VLN approaches, like our method, perform continuous control, rather than way-point selection. Roh *et al.* and Sriram *et al.* present methods that control an autonomous vehicle within two environments in the CARLA simulator given natural-language instructions. However, in contrast to our method, they only considered single-turn instructions as input and the trained autonomous vehicle did not engage in multi-turn dialogue. Also, our approach takes noisy SLAM data from a real physical robot as input. This data in noisier and less rich than the synthetically generated 3D images in the CARLA simulator. Roh *et al.* only evaluated their approach in the CARLA simulator, while Sriram *et al.* also conducted a single experiment on real data (the KITTI dataset) and one experiment on a physical electric vehicle. We, however, rigorously demonstrate our system's performance on a real robot in real indoor environments in 52 trials.

Some prior work only presented methods for part of the VLN task. Deitke *et al.* and Zhu *et al.* demonstrated vision-only navigation to a specified target object.

1.6 Comparison with Prior Work

We compared our method to five prior VLN approaches [9]–[13]. Since these methods cannot wander around a floor of a building, nor approach a person, nor hold a conversation, we had a volunteer provide navigational instructions to a nearby door. These methods also do not translate human language instructions to robot actions; they translate to edges in a graph, not robot motor control. However, one can imagine that part of our system, specifically the HOLD_CONVERSATION, FOLLOW_DIRECTIONS, and NAVIGATE_DOOR behaviors could be replaced with one of these VLN approaches. Thus we compare part of our system with existing VLN research on the shared subportion of the task and we use methods from our system to convert the output of the VLN systems into robot motor control. This simplified task involves placing the robot in a fixed position, providing it with a textual input of the navigational instructions, and evaluating how well these methods were able to successfully follow the instructions. Success is measured by distance from the goal where the goal is often only a few waypoints away.

To perform this evaluation, we first trained these methods offline and then deployed them on our robot to perform live trials involving navigation in an unknown physical environment with non-scripted instructions from a volunteer. In order to train these methods, we needed a training set with the same structure as R2R, where all navigation is performed on a fixed graph with images available for all orientations at each vertex. Thus we collected an additional dataset, in the style of R2R, using our robot in the same three buildings (EE, MSEE, and PHYS) we used to develop our algorithms, by placing the robot at fixed waypoints, approximately 3 m apart, and collected 36 images at each waypoint at different pan angles (every 30°) and tilt positions (0°, 0° and 30°). We used 0° twice since pointing the camera towards the ground was not possible with our robot configuration and we wanted to match the same positive tilt angle used in R2R. We then collected natural-language utterances from 27 participants that provided navigational instructions from various starting locations and orientations to various ending locations. We paired the utterances with the corresponding sequences of waypoints to construct a dataset with 2,814 samples. We appended a sentence referring to the ending location to each utterance. This was done for two reasons: 1) like our system, the VLN system would need to take the goal location as input and 2) we adhered to the style and structure of the R2R dataset whose utterances refer to the goal location. For example, if one of the participants was asked for instructions to room 232, and they replied "Go straight and then make a left," we augmented the utterance to be "Go straight and then make a left. Stop at room 232." We refer to this dataset as office-to-office (O2O). We trained Seq2Seq [9] and NDH [10] on R2R, using the code base provided by Thomason *et al.*, then fine tuned on O2O. We trained SF [12], RCM [13] and Babywalk [11] on R2R, using the code base provided by Zhu *et al.*, then fine tuned on O2O.

Evaluating with live trials in novel environments in a different building, however, required us to map the small finite action space output by these VLN models to the continuous action space needed to perform our task. Seq2Seq and NDH predict one of six actions to perform: rotate left or right by 30°, look up or down by 30°, drive forward, and end. For these approaches, we determine forward positions by searching the robot's SLAM map for positions in free space in front of the robot's current position. For the other actions, the robot would rotate left or right by 30° or tilt its camera up or down by 30°. During the trials, upon completion of an action, the robot took the current camera image, extracted its features, and decided what action to perform next. SF, RCM, and Babywalk predict an adjacent waypoint to navigate to. For these approaches, we would determine adjacent waypoints by searching the robot's SLAM map for positions in free space that corresponded to positions in front of, behind, left of, and right of the robot's current position. This was made possible by one of the contributions of our paper (Section 1.2.3).

During the trials, upon arrival at a waypoint, the robot collected 36 images at the same pan and tilt angles described above, extracted their features as described in Zhu *et al.*, and decided whether to drive to an adjacent waypoint or indicate completion.

A strict evaluation of success of a purely autonomous method that cascades the prior models that output discrete navigation actions designed to navigate in a symbolic graph with our code that instead navigates to physical waypoints from those discrete actions leads to very low success rate. Thus we relaxed our success criterion to manually intervene if the robot got too near an obstacle. We moved the robot away from the obstacle, keeping the same orientation, but positioning it closer to the middle of the hallway, to allow the trial to continue.

We conducted eight trials with each of the five methods in KNOY, where our approach had neither its best nor worst success rate. We positioned the robot in the same starting location and orientation for all eight trials and provided the same corresponding text instructions. Note that these starting locations, orientations, and text instructions are the same as 8 of the trials described in Section 2.8. The goal location was appended to the text instructions in the same way as our training set as described above. All but two trials were concluded when the method met the stop condition. For Seq2Seq and NDH, the stop condition is when the method predicts the stop action. For SF, RCM, and Babywalk, the stop condition is when the method predicts the current waypoint. The two trials, one using NDH and one using SF, were concluded manually prior to the stop condition because the robot became stuck driving in a loop in an open area. If the robot arrived within 1 m of the goal location, we considered it a success. We then manually drove the robot to the goal location. Using odometry, we measured the navigation error, which is the driving distance from the ending location to the goal location. We compared the results of the VLN systems that used the same starting locations, orientations, and text instructions from 8 trials from Section 2.7 to the 16 trials we ran of this system in the same test building (KNOY). This was not an apples-to-apples comparison for several reasons. Firstly, the starting locations, orientations, and input instructions were not the same as mentioned above. This is because the trials were originally intended to serve as a comparison to the trials conducted in Section 2.7. However, the VLN system trials were conducted on the same 4 floors of KNOY and the input utterances were very similar. Table 1.13 shows some of the utterances from our system's trials versus those from the VLN system trials. Although the language is not identical, it is similar, and we believe this allows us to compare our system's performance to that of the VLN systems. Secondly, our system relies on multi-turn spoken conversation instead of a single text input, unlike the VLN systems which can only take a single piece of text as input. Lastly, our system is able to recover from failures in individual behaviors, unlike the one-shot approach used in VLN systems and in our experimental design in Section 2. Table 1.14 shows that four of the five prior methods failed to correctly execute the

Table 1.13. Samples of language text used during our trials and the VLN trials.

Our system trials
turn around take your first right go down the hallway about 30 to 50 ft and it will be on your left.
turn right then turn left go straight turn left. and your destination is on the rake.
yes, go straight turn left and the bowl is on the right.
sure, turn around take your first left and go down the hallway room 417 will be on the right hand side.
VLN trials
Continue straight at the end of the hallway. Take a left and then take another right. Stop at room 339.
Around take a right and then take another right. Stop at room 313.
Continue straight, take a right and then left. Stop at room 278.
Turn around at the end of the hallway. Take a left. Stop at room 443.

instructions specified by the input utterances in all eight of their respective trials and one prior method succeeded only once. In contrast, our method succeeded in 13 of the 16 trials.

Table 1.15 shows how much of the instructions were followed and how close the robot got to the goal. Note that all of the trials reported in Table 1.14 and Table 1.15 were conducted on the same four floors of KNOY. However, for each table, the trials in row 6 had different starting positions, goal locations, and language text than those in rows 1-5. The trials in rows 1-5 had the same starting positions, goal locations, and language text as each other; these shared the same starting positions, goal locations, and language text as eight of the the trials described in Section 2.7. All prior VLN methods correctly follow only a small portion of the instructions and end up very far from the goal. In contrast, our method gets very close to the goal on average. For the methods that predict actions, we include the number of actions predicted and how many actions the robot successfully predicted before predicting an incorrect action. For the methods that use adjacent waypoints, we include the number of waypoints traversed and how many waypoints the robot successfully navigated before predicting an incorrect waypoint. For the methods that use actions, the robot would often turn to face the wall and predict the forward action repeatedly before predicting a rotation action. Were it not for our code that prevented the robot from performing illegal actions, the robot would have driven into walls almost immediately. For all methods, the robot would frequently drive in the opposite direction from that indicated in the instructions. In contrast, our system generally navigated the robot in the correct direction towards the goal.

Table 1.14. Comparison of our system with prior VLN systems. Successful trials were those where the robot stopped within 1 m of the goal location.

	Successful number of trials	Total number of trials	Success rate
Seq2Seq [9]	1	8	12.5%
NDH [10]	0	8	0.0%
SF [12]	0	8	0.0%
RCM [13]	0	8	0.0%
Babywalk [11]	0	8	0.0%
Our System	13	16	**81.3%**

Table 1.15. Navigation statistics from the results reported in Table 1.14. Correct actions is the average number of actions the system correctly predicted and executed before heading off in a wrong direction. Total actions is the average number of actions executed before the stop condition is met. Correct Waypoints is the average number of waypoints the system correctly predicted and navigated to before heading off in a wrong direction. Total Waypoints is the average number of waypoints navigated to before the stop condition is met. Navigation Error is the average driving distance from the ending location to the goal location.

	Correct Actions	Total Actions	Correct Waypoints	Total Waypoints	Navigation Error
Seq2Seq [9]	6.1	82.5	n/a	n/a	19.0 m
NDH [10]	4.6	93.9	n/a	n/a	37.4 m
SF [12]	n/a	n/a	3.9	26.8	30.0 m
RCM [13]	n/a	n/a	3.5	15.8	42.8 m
Babywalk [11]	n/a	n/a	0.1	8.9	34.2 m
Our System	n/a	n/a	n/a	n/a	**0.8** m

Table 1.16. Validation success rate of prior VLN systems, using metric from [11].

	EE	MSEE	PHYS	Average
Seq2Seq [9]	0.113	0.088	0.072	0.091
NDH [10]	0.097	0.088	0.066	0.084
SF [12]	0.140	0.057	0.044	0.080
RCM [13]	0.073	0.076	0.197	0.115
Babywalk [11]	0.132	0.080	0.098	0.103

In addition to these experiments, we also evaluated the prior VLN approaches in isolation, without our navigation system. We did this by training and validating them directly on O2O. We performed three-fold cross validation for each of the five methods. We split O2O into three folds, one for each of the three buildings. In a round-robin fashion, we trained each of the five methods on two of the folds and validated on the third fold. Table 1.16 shows the validation success rate for each method for each fold as well as their average success rate. On average, the success rates are very low. As opposed to our live trials, the prior VLN approaches had access to a known discrete graph of the environment in these experiments. Therefore, the results show that even with a known discrete graph of the environment, the prior VLN methods are generally unable to find a goal in our environment. Table 1.17 shows the validation navigation error for each method for each fold as well as their average navigation error. These results are close to what we observed in our live trials (see Table 1.15). This shows that the VLN approaches are not limited by the navigation mechanisms we used to deploy them on the robot. Instead, this confirms that they work poorly in our environment. Additionally, this is all on just the simplified subset of our task that is shared in common with prior VLN work. It does not involve searching for a person to ask directions from, engaging in a dialog with clarification questions to resolve incomplete, ambiguous, and inconsistent instructions, or reading door tags to find the goal, all of which make our task even more difficult than the standard VLN task.

1.7 Conclusion and Future Work

In this manuscript, we proposed The Amazing Race^{TM}: Robot Edition task in which a robot must find any designated location. We described a system design that solves this task

Table 1.17. Validation navigation error of prior VLN systems, using metric from [11].

	EE	MSEE	PHYS	Average
Seq2Seq [9]	27.368	37.566	40.901	35.278
NDH [10]	29.864	33.787	39.746	34.466
SF [12]	28.866	33.162	49.208	37.079
RCM [13]	27.592	32.464	37.772	32.609
Babywalk [11]	28.753	33.642	40.290	34.228

with the scope of finding a specific room on a single floor of a building. We demonstrated that our system architecture lets a robot complete this task by a making a series of logical steps akin to those a human would make. Our robot autonomously finds a person in the environment and engages them in dialogue for directions. It then follows these directions to reach the hallway in which it then searches the doors to find the designated goal.

We demonstrated that our system was successful in 76.9% of the trials that we conducted. As discussed, future work entails making improvements to each of the behaviors in our system to both improve the success rate and increase the scope of the task.

We make available our code and rosbags as well as describe our software configuration in order for others to make meaningful comparisons and build off of our work. Our rosbags[5] can be used as a dataset for comparison of different methods. Our code and software configuration can be deployed on most robot platforms, as the main dependencies are common sensor data in the form of LiDAR, camera images, and odometry. This would allow others to apply our system to other indoor environments to evaluate and compare against its performance. Additionally, we release our O2O dataset which can be used to evaluate VLN methods in the same way as the R2R dataset. This can allow others to improve VLN methods to be more robust to the complex and diverse environments present in our O2O dataset.

[5]↑A rosbag is a file that stores message and sensor data at each time step in a trial.

2. TALK THE TALK AND WALK THE WALK: DIALOGUE-DRIVEN NAVIGATION IN UNKNOWN INDOOR ENVIRONMENTS

This chapter includes a submission to IROS, which describes a machine-learned variant of the Hold_conversation and Follow_directions behaviors. I was responsible for the following technical section: Section 2.6. Jared Johansen was responsible for the following technical section: Section 2.4.

2.1 Introduction

Imagine an office environment where individuals work in separate areas to follow social-distancing guidelines. You need to give an important package to a colleague but you're unable to go in person. You summon a robot and give it navigational instructions in plain English to your colleague's office. The robot engages you in a dialogue to clarify some ambiguity in your navigational instructions and then follows the plan it infers to deliver the package. In this paper, we present a machine-learned system that makes a significant step towards this reality.

Our system consists of two main components. The first is a transformer-based network trained to interpret spoken natural language and convert it into a navigation plan that the robot can execute. Crucially, it's designed to support multiple turns in a conversation by taking the robot's current plan and a transcript of a spoken utterance as input and producing an updated plan and follow-up question (if necessary) as output. Some previous work (e.g., [3], [4]) only considers an agent receiving a single text instruction as input and producing a single plan as output. This is impractical, however, for real-life applications that depend on spoken language and speech recognition. A person might misspeak or the speech recognition could be erroneous. This necessitates support for live dialogue to rectify any potential errors or ambiguity. To train this network, we collected a novel dataset of navigational-instruction utterances, created transcript-plan pairs, and augmented them to support multi-turn dialogue.

The second component of our system is a 2D CNN-based network trained to produce navigational goals that correspond to the plan produced by the dialogue component. Navigational instructions generally consist of left or right turns in specific locations, such as "turn right at the end of the hallway." If the robot does not have a map of its environment in advance, it would be unable to directly generate an entire navigational plan to execute these navigational instructions. Therefore, the network takes the latest map (produced by SLAM) and the current step in its plan as input to produce a navigational goal as output. It is designed to produce forward goals by default until the robot arrives at the desired intersection, in which it should produce a goal corresponding to the desired direction. Additionally, it is trained to recognize when a step in the plan cannot be executed. To train this network, we collected a novel dataset of actual trajectories driven and SLAM maps produced by a robot on several floors of several buildings.

The dialogue and navigational components are combined into a system that enables dialogue-driven navigation in an end-to-end fashion in unknown, indoor environments. To test out the effectiveness of a robotic algorithm, it is important to test in the real world where it is required to operate in continuous space with noise and unanticipated conditions. To this end, we recruited volunteers to converse with the robot and provide navigational instructions to various locations in three real buildings. These volunteers were not involved in our natural-language dataset collection. These three test buildings were not part of either the dataset used to train the dialogue component or the dataset used to train the navigation component. We evaluated whether the navigational instructions were converted into a correct plan and whether the plan was correctly executed. We demonstrate our system's performance in 39 trials. We also demonstrate in Section 2.8 that algorithms that may work in simulation do not necessarily perform well in the real world.

Explicitly, this paper makes the following contributions:

- We provide a novel dataset of transcript-plan pairs for navigation in indoor environments. We apply a novel data augmentation method to train a transformer network to support multi-turn dialogue, allowing the robot to ask the person clarifying follow-up questions.

- We provide a novel dataset of robot trajectories paired with navigational commands in several indoor environments. We use automatic annotation and data augmentation techniques to train a 2D CNN on this data to produce navigational goals and feedback statuses that correspond to the input instructions.
- Unlike most prior work that demonstrates performance on synthetic data or their own training environments, we demonstrate performance in real indoor environments with real volunteers that are distinct from our training sets to show the generalizability of our approach.
- We train three prior vision-and-language navigation methods on our data and deploy them them on our robot in one of our test buildings. We show that our approach vastly outperforms these methods.

2.2 Related Work

There has been considerable prior work on vision-and-language navigation (VLN). Some of this work, [9]–[13], [43], trained and evaluated VLN models on the Room-2-Room (R2R) dataset [9]. This dataset consists of natural-language text instructions paired with corresponding trajectories in a simulated indoor environment. These trajectories are sequences of vertices in a discrete graph, where each vertex has a panorama of images to represent the view at that vertex. [48] and [49] presented similar methods to choose waypoints to reach a goal specified by natural-language instructions but in simulated outdoor environments. Our work differs significantly from all of this work in a number of key ways.

First, in the R2R simulator, robot position is represented as a vertex in a discrete graph and visual information, although from real images, is noise-free and deterministic at each vertex. In contrast, rather than just repeatedly outputting one of a small number of adjacent graph vertices to eventually reach a goal vertex, we address a more complex problem: controlling a physical robot in the real world with a noisy continuous position and action space and noisy continuous observations. While [43] trained in a simulated environment, it tested both in simulated environments and on a real robot in real environments. However, when the navigation graph was known *a priori*, performance in the real environment was comparable to that in simulation, but results were very poor when it was unknown and waypoints were predicted on the fly. Our system is able to successfully execute navigation instructions without a known map of the environment, just with the SLAM map that is built as the robot drives.

Second, the above prior work took single-turn text as input. Our system interacts with a person in multi-turn spoken dialogue and is trained to be robust in the face of noisy speech recognition. Such dialogue is crucial for clarifying potentially ambiguous instructions. [44] and [45] presented VLN approaches that perform continuous control, rather than waypoint selection, of an autonomous vehicle within two environments in the CARLA simulator given natural-language instructions. However, they also only considered single-turn instructions as input and the trained autonomous vehicle did not engage in multi-turn dialogue. Also, in contrast, our approach takes noisy SLAM data from a real physical robot as input. This data in noisier and less rich than the synthetically generated 3D images in the CARLA simulator. [44] only evaluated their approach in the CARLA simulator, while [45] also conducted a single experiment on real data (the KITTI dataset) and one experiment on a physical electric vehicle. We, however, rigorously demonstrate our system's performance on a real robot in real indoor environments in 39 trials.

Third, we output a feedback status along with our goal coordinates. This allows for direct feedback about whether or not the input instruction was successfully executed or whether it eventually failed (because the instruction was unachievable). The latter allows the robot to detect when an instruction it was given is incorrect.

There has been prior work, [34]–[37], [50], that, like our work, has also focused on achieving multi-turn dialogue understanding on a physical robot. However, this work operated within smaller and simpler environments than our work; we train and evaluate our system on several large unmodified office environments. [34] proposed a method to allow a robot to learn object-related concepts through dialogue. Only a single demonstration on a real robot was done in a single room with several objects on a table. [50] demonstrated a dialogue and navigation system for a physical robot; however semantic regions in the map were provided to the system in advance and there was no full evaluation of the system's performance, only preliminary experiments in a single environment. [35] constructed a framework to control a physical robot with spoken language, but it was only tested in a simulator in which the virtual robot could execute navigation commands from a finite discrete set. [36] demonstrated their algorithm on a quadcopter, but the environment was very small with just a few objects on a green surface. The environment was only varied by placing different objects in different positions. [37] collected data from a physical robot to train their algorithm to follow natural-language instructions. However, evaluation of the approach was only performed in simulation.

There has been other prior work [46], [51]–[55], that, like ours, first converts natural-language instructions into a plan that can then be interpreted and executed. [51] and [52] modeled instruction-action pairs to convert each instruction independently into an action in a noninteractive fashion to map an instruction sequence to an action sequence. Our system interacts with a person in spoken dialogue to clarify ambiguities and update the plan in the context of the dialogue. [53] and [54] determined a set of constraints from natural-language instructions and applied those constraints to a known map to generate a navigation trajectory. [55] and [56] both depended on having topological representations of the environment. Unlike our approach, these methods would not work in an unknown environment.

Finally, some prior work only presented methods for part of the VLN task. [46] and [47] demonstrated vision-only navigation to a specified target object. [57] presented a data-driven parser for understanding navigation directions for the purpose of human-robot dialogue but did not apply it to any form of navigation, simulated or real.

2.3 System Architecture

Our system consists of a dialogue component and navigation component. The dialogue component facilitates multi-turn conversation with a person to produce a plan. The plan is converted into commands that the navigation component then executes. The navigation component incrementally executes each command by analyzing the SLAM map and producing goal locations for the robot to drive to and a feedback status to indicate whether to advance to the next command. A diagram of our system can be seen in Figure 2.1.

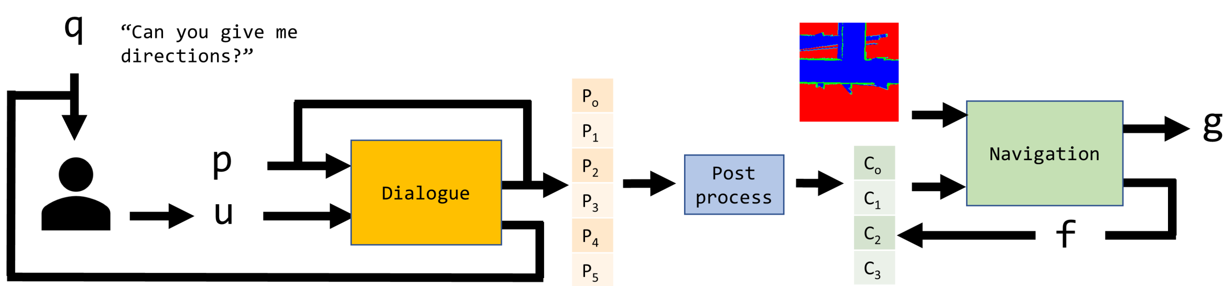


Figure 2.1. System Diagram. A question, q, is posed to a person. Their utterance, u, and the current plan, p, is fed into the dialogue component, which produces an updated plan and follow-up question. Dialogue loops until a complete plan, $[p_i]$, is produced. The complete plan is converted into robot commands, $[c_i]$, which are fed into the navigation network. The navigation component produces a goal location, l, and feedback status, f, which are used to carry out all commands.

2.4 Dialogue

The purpose of engaging in dialogue with a person is to construct a plan of how to navigate to a specific destination. If the person were to formally communicate the plan to the robot, using formalisms the robot is capable of executing, this task would be trivial. What makes this nontrivial is that the communication is done informally through spoken natural language. In order to facilitate this spoken dialogue, we collected and augmented a dataset to train a transformer-based network to take as input a *current plan* and *input transcript* of the spoken utterance, and to produce an *updated plan* and *follow-up question* as output.

2.4.1 Dataset collection

To collect a dataset of navigational instructions, we recruited 27 subjects that spoke English and were familiar with three buildings on our campus. For each of four floors, for each of the three buildings (PHYS, MSEE, and EE), we described a starting location and orientation of both a robot and a passerby. We asked them to imagine they were the passerby and that the robot had posed a single query, asking for navigational instructions to a location within that building. They were allowed to provide partial instructions or indicate they did not know the location of the destination. The subjects recorded a verbal response to each query and we used Microsoft's speech-to-text engine [58] to convert the spoken responses into text. We collected a total of 8797 navigational-instruction utterances and produced their corresponding transcripts. Some navigational-instruction utterances involve using an elevator to move to another floor. In this work, we handle the dialogue component of using an elevator to move between floors, but focus on performing navigation in a single floor. Future work will address accomplishing multi-floor navigation.

We identified a set of *plan concepts* of various kinds, including directions, intersections, goals, and other that cover the vast majority of navigational instructions contained in our transcripts, so that each transcript would have a corresponding plan with a sequence of these concepts. All plan concepts, their types and definitions can be found in Table 2.1. We posted

Table 2.1. Possible plan concepts for plan annotation. `int-L` and `int-R` have a left or right turn, respectively. `end` refers to the end of the hallway; `elbow` refers to an elbow. The first four direction concepts refer to the direction the robot should drive in. It will continue moving in that direction until it encounters the next step in the plan. `either` refers to turning left or right at an elbow when it is not explicitly stated (e.g., "go around the corner"). `goal-F`, `goal-L`, and `goal-R` respectively refer to goals that are ahead, on the left, or on the right. □ refers to an unknown or unspecified step in the plan. `change-floor` refers to using an elevator to move between floors.

Type	Plan concepts
Intersections	`int-L`, `int-R`, `end`, `elbow`
Directions	`turn-around`, `forward`, `left`, `right`, `either`
Goals	`goal-F`, `goal-L`, `goal-R`
Other	□, `change-floor`

each transcript on Amazon Mechanical Turk (AMT) and asked two workers to construct a coherent plan using these plan concepts.

When the constructed plan had a □, we asked the AMT workers to type a follow-up question that they would ask to resolve the □. Each of the 8797 transcripts had two plan annotations and (potentially) two follow-up questions. When the plans and follow-up questions were the same, we created a single sample. When the plans and follow-up questions were different, we kept both if they were both reasonable. Otherwise, we kept the most correct one. If neither were correct, we manually modified one to be correct and retained this as a sample. This process resulted in a total of 9818 unique samples consisting of [□] for the current plan, an input transcript, an updated plan, and a follow-up question. If the updated plan was a *complete plan* (i.e., no □ present), we used a default follow-up question of "Got it. Thanks!"

From among the 9818 samples, we found that 7467 (76.1%) had complete plans and 2351 (23.9%) had *partial plans* (i.e., a □ present). We sorted the samples with a partial plan into four categories: *empty*, *need-elevator*, *need-first*, and *need-last*, based on their plan pattern (see Table 2.2). Although these partial plan categories occur less frequently in our dataset than samples with a complete plan, they are realistic possibilities during live spoken dialogue due to pauses in speech, poor speech recognition, or incomplete navigational instructions. Therefore, we augmented our dataset to increase the number of samples in these categories. For *empty*, we generated random sentences that had an updated plan of [□]. The category *need-elevator* had a relatively large number of samples, so we did not perform any augmentation. For *need-first*, we used samples with complete plans, but removed certain keywords (e.g., "go straight" or "turn-around") from the beginning of the transcripts and replaced the first instruction in the plan with □. We performed a similar augmentation for *need-last*, but truncated text from the end of the transcript and replaced the corresponding concepts in the plan with □. Table 2.2 shows the total sample counts before and after augmentation.

Table 2.2. Number of samples for each partial plan category.

Category	Pattern	Original count	Augmented count
empty	$[\square]$	125	2125
need-elevator	$[\square,$ `change-floor`$, \ldots]$	2100	2100
need-first	$[\square, \neg$`change-floor`$, \ldots]$	68	2200
need-last	$[\ldots, \square]$	58	1818

2.4.2 Dialogue turn generation

The purpose of dialogue is to rectify any missing information (i.e., $\square$) in the current plan. The data we collected only simulates the first turn in a conversation, in which the current plan is [$\square$], and the input transcript contains complete, partial, or no information to the destination. To facilitate dialogue, we must train the network to handle subsequent turns of conversation in which a person responds to follow-up questions produced by prior turns. Therefore, we further augment the dataset by producing *follow-on samples* whose current plan is a partial plan of the form described in Table 2.2 and whose input transcript would be a valid response to a previously-asked, follow-up question.

To generate these follow-on samples, we use custom logic to create realistic input transcripts; correct, updated plans; and relevant, follow-up questions. The input transcripts, and portions of the updated plans, come from the *first-turn samples* (which we described in the previous section). For follow-on samples whose current plan comes from the *need-elevator* category, we find first-turn samples whose destination happened to be an elevator and whose updated plan was a complete plan. The transcripts from these first-turn samples contain navigational instructions to an elevator. We use these as the input transcripts of our follow-on samples. The updated plans from these first turn-samples contain a complete plan to an elevator. We strip off the goal concept which leaves a sequence of plan concepts to the elevator. Since the *need-elevator* category has a plan of the form [$\square$, `change-floor`, $\ldots$], we replace the $\square$ with the sequence of plan concepts to the elevator. An example of this type of follow-on sample can be seen in Table 2.3.

For follow-on samples whose current plan comes from the *need-first* category, we find first-turn samples whose input transcripts include words such as “forward,” “straight,” “backward,” or “turn-around” among the first ten words. We use the first ten words of these transcripts as the input transcript of our follow-on sample. The updated plan of our follow-on sample is the same as the current plan, but replaces $\square$ with `forward` or `turn-around`, depending on the first concept in the updated plan from the first-turn sample.

For follow-on samples whose current plan comes from the *need-last* category, we first determine whether the plan concept before the $\square$ is a direction concept or an intersection concept. When it is a direction concept, we find two types of samples from among our first-turn samples: one whose updated plan starts with `forward` and does not have ☐ within the first four steps and one whose updated plan's first step is `forward` and whose second step is a goal concept. With these, we create two new follow-on samples. For each, the input transcript of the first-turn sample is used verbatim in the follow-on sample. We take the updated plan from the first-turn sample and strip off the first `forward` concept to leave a sequence of plan concepts. We take the current plan in our follow-on sample, replace the ☐ with this sequence of plan concepts, and use this as the updated plan.

When it is a intersection concept, we find first-turn samples whose updated plans would be consistent when combined with the current plan to form the updated plan. For example, if the intersection is specifically `end` or `elbow`, we find two first-turn samples, one whose updated plan has `left` as the third step and one whose updated plan has `right` as the third step. These plans (third step and onward) are appended to the follow-on sample's current plan to serve as its updated plan. When it is `int-L` or `int-R`, we again find two first-turn samples, one whose updated plan has `left` or `right`, respectively, as the third step and one whose updated plan starts with `forward` and does not have a ☐ within the first four steps. The follow-on's updated plan is formed by appropriately combining the current plan with the relevant portions of the first-turn sample's updated plan.

Finally, for each sample, it's possible that the input transcript is nonsensical or an empty string (representing silence). We create additional samples for these cases in which the updated plan is simply the same as the current plan. It's also possible the person may indicate that the robot should start over. We create samples for this scenario as well where the updated plan is [☐]. To create relevant follow-up questions, we wrote custom logic that looked at the plan and generated a question (see Table 2.4). Table 2.3 shows one first-turn sample and one follow-on sample.

2.4.3 Augmented dataset training and validation

We used the source code at [59] to train a network based on Text Summarization with Pretrained Encoders [60] to take the current plan and input transcript as input and to

Table 2.3. Training samples.

current plan	[□]
input transcript	yeah, go straight and then make a right
updated plan	[`forward`, `int-R`, `right`, □]
follow-up question	What do I do after turning right?
current plan	[□, `end`, `left`, `goal-R`]
input transcript	think you might have to turn around
updated plan	[`turn-around`, `end`, `left`, `goal-R`]
follow-up question	Got it. Thanks!

Table 2.4. Partial plan follow-up questions.

Category	Follow-up question
empty	Repeat original question.
need-elevator	Ask for navigational instructions to the elevator.
need-first	Ask which direction to start out going.
need-last	Ask what do to after last instruction.

produce an updated plan and follow-up question as output. For training, we specified a maximum input length of 200 tokens. We divided the subjects into five folds to perform leave-one-fold-out cross validation. The average validation accuracy was 69.9% and the average follow-up question relevance was 99.0%.

2.4.4 Spoken dialogue

When initiating a conversation, the plan is initialized to [□] and begins with the robot posing a question. It waits for a response, which is transcribed to produce a transcript. The current plan, along with this transcript, are fed into the network, which produces an updated plan and follow-up question. The robot poses the follow-up question and this process repeats itself until the updated plan has no □.

To help facilitate a more natural conversation, we have a small amount of code to address a few corner cases that may arise in spoken conversation. If a response is not heard within 5 s, the robot states this fact and repeats its question. If the robot is not able to extract any information from a response, it indicates such and may provide some information about what it does understand (e.g., directions and intersections). If the robot goes two turns without the plan changing (e.g., it fails to understand the navigational instructions or hears no response), the robot ends the conversation and carries out whatever portion of the plan is usable.

2.5 Navigation Commands

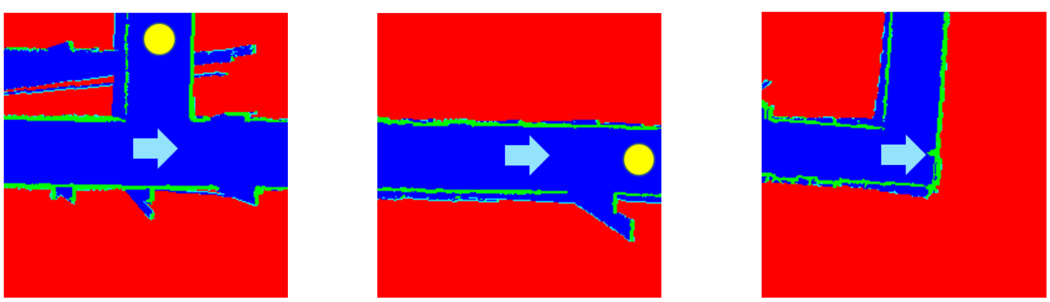
Once a complete plan is produced, it is post-processed to take the plan concepts and convert them into a sequence of commands for the navigation component to execute. The commands are generated from plan subsequences by pairing adjacent intersection and direction concepts in the plan, with one exception: `turn-around`, which is treated as a stand-alone command. Table 2.5 shows these conversions.

Table 2.5. Commands produced from plan subsequences.

Plan subsequences	Command	Description
`end,left`	**end__left**	turn left at end of hallway
`end,right`	**end__right**	turn right at end of hallway
`int-L,forward`	**int-L__forward**	go forward when left available
`int-L,left`	**int-L__left**	turn left when left available
`int-R,forward`	**int-R__forward**	go forward when right available
`int-R,right`	**int-R__right**	turn right when right available
`turn-around`	**int-B__backward**	turn around when possible
`elbow,left`	**elbow__left**	turn left at elbow
`elbow,right`	**elbow__right**	turn right at elbow
`elbow,either`	**elbow__either**	go through elbow

2.6 Navigation

The navigation component executes each command by continuously looking at the SLAM map to determine a goal location where the robot should drive and a feedback status to indicate whether to advance to the next command. When it arrives at the intersection indicated by the command, the navigation component predicts a goal location that executes the turn indicated by the command and a feedback status of *transition* indicating it can advance to the next command. However, if the robot is in the middle of a hallway, the navigation component predicts a goal location that drives it down the hallway and a feedback status of *forward* indicating that the command has not been executed. If it reaches the end of the hallway without detecting the intersection, the navigation component stops the robot and produces a feedback status of *failure* indicating that the robot has failed to execute the command. We illustrate this in Figure 2.2 which depicts three SLAM maps and commands depicting different scenarios along with their corresponding feedback statuses. The robot's position and orientation is depicted by the light blue arrow and the yellow circle represents the goal location where the robot should drive.



(a) **int-L_left** (b) **elbow_either** (c) **end_right**

Figure 2.2. (a) The robot can execute the command; it produces a goal location that drives it into the hallway on its left and a feedback status of *transition*. (b) The robot is in a hallway and cannot execute the command yet; it produces a goal location that drives it further down the hallway and a feedback status of *forward*. (c) The robot has reached the end of the hallway and cannot make a right turn; it stops and produces a feedback status of *failure*.

2.6.1 Network

We implement this navigation with a neural network that takes the command and SLAM map as input and predicts a goal location as (x, y) coordinates and feedback status with a 3-way classifier. The neural network consists of 4 convolutional layers that extract features from the SLAM map. These features are concatenated with a one-hot encoding of the input command which is passed through 2 fully-connected layers to create a final representation of the input. This representation is passed into two parallel layers, to predict the coordinates and feedback status, respectively.

2.6.2 Training

To train this network, we required training samples that consist of a SLAM map, input command, goal location, and feedback status. To collect the SLAM maps, we manually navigated the robot on several floors of three buildings (PHYS, MSEE, and EE), ensuring to cover every hallway. The robot was intentionally driven in the centers of hallways and turned in the centers of intersections to simulate ideal navigation. We used Google Cartographer [14] to determine the robot's positions and generate the SLAM maps as it was driving, and these were recorded at a rate of 10 Hz.

Each SLAM map is used to create multiple training samples by pairing each of the commands in Table 2.5. Then, for each map-command pair, we determine the corresponding goal location and feedback status. To do this, we first must determine what directions are available to the robot in the SLAM map by using the robot positions that were recorded. A direction is considered available if there is a recorded position in the SLAM map in that direction within 5 m of the SLAM map's center. We also use these positions to determine a goal location for each available direction. This process is illustrated in Figure 2.3. Then we use the available directions to determine the intersection type(s) based on the correspondence indicated in Table 2.6.

We can now use this information to form the training samples. For each command, if the SLAM map has the intersection corresponding to it, we make a sample whose goal location corresponds to the direction specified by the command and whose feedback status

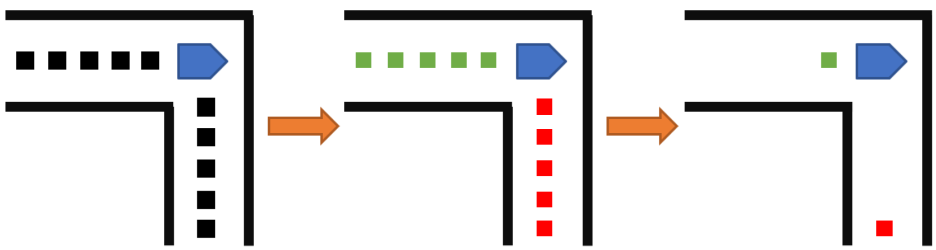


Figure 2.3. Left: Black squares represent all positions within 5 m of the center of the SLAM map. Center: Green squares represent positions corresponding to *backward*. Red squares represent positions corresponding to *right*. Right: Green square is goal location for *backward* and red square is goal location for *right*. Available directions are *backward* and *right*; using Table 2.6, the intersection types are `int-R`, `end`, `int-B`, and `elbow`.

Table 2.6. Intersection types based on available directions.

forward	backward	left	right	Intersection types
no	no	no	yes	`int-R`
no	no	yes	no	`int-L`
no	no	yes	yes	`hallway`, `int-L`, `int-R`
no	yes	no	no	`end`, `int-B`
no	yes	no	yes	`int-R`, `end`, `int-B`, `elbow`
no	yes	yes	no	`int-L`, `end`, `int-B`, `elbow`
no	yes	yes	yes	`int-L`, `int-R`, `end`, `int-B`
yes	no	no	no	`hallway`
yes	no	no	yes	`int-R`
yes	no	yes	no	`int-L`
yes	no	yes	yes	`int-L`, `int-R`
yes	yes	no	no	`hallway`, `int-B`
yes	yes	no	yes	`int-R`, `int-B`
yes	yes	yes	no	`int-L`, `int-B`
yes	yes	yes	yes	`int-L`, `int-R`, `int-B`

is *transition*. If the SLAM map does not have the intersection, but *forward* is an available direction, we make a sample whose goal location corresponds to *forward* and whose feedback status is *forward*. If the SLAM map does not have the intersection and *forward* is not available, we make a sample whose goal location is $(0.0, 0.0)$, and whose feedback status is *failure*.

We train the network on these samples, using two loss functions for each of the respective outputs. For the 3-way feedback status classifier, we use weighted cross-entropy and for regressing to the (x, y) goal coordinates we use mean-squared-error. During training, we randomly apply rotation and translation (y-axis only) transformations to simulate the robot having an off-center position and orientation in a hallway, allowing the network to still predict correct goals.

2.6.3 Command Execution

To achieve autonomous navigation on the robot,we apply the trained network continuously to the current input command and SLAM map at a rate of 10 Hz. Because noisy output is possible with noisy SLAM data, we consider multiple consecutive outputs from the network to make navigation decisions. This accomplishes two crucial things: the robot will not navigate incorrectly due to a single spurious output and the robot will be able to catch an intersection if it is driving past it. Specifically, the navigation component considers the 10 most recent outputs, feedback statuses and goal locations, of the network. Among these outputs, if the most common feedback status is *forward*, the robot will drive to the goal location of the most recent output. However, if it is *transition* or *failure* the robot will stop moving and wait for 10 additional outputs to make a final decision. If the most common feedback status of these outputs is *transition*, the robot computes the aggregates the goal locations of only the outputs whose status is *transition*. The robot will then drive to the aggregated goal location. Once that navigation is complete, the robot will then take the next command as input into the network. If it is *failure*, however, the robot will stop and not attempt to execute any more commands.

2.7 System Experiments

To evaluate the performance of our entire system, we tested in three buildings that were not part of our training sets.[1] These test buildings consist of multiple hallways and alcoves of varying lengths and widths, with common objects, such as water-fountains, trashcans, and chairs, found in various places (see Figure 1.15 for example floor plans). We placed the robot in different locations on different floors of the three new buildings. In each location, it was tasked with asking a person for navigational instructions to a goal location and then following the plan it constructed. We recruited volunteers, who were not part of our training set, to engage in the dialogue. After constructing a plan, it attempted to execute the commands to reach the hallway containing the goal location. The ROS *move_base* package [61] was used for autonomous path planning, with obstacle avoidance, to driving goals produced by the navigation network. A demonstration of our system can be seen in the attached video submission.

We consider several different forms of successes. If the navigational instructions were interpreted and followed correctly, this is a complete success. If the navigational instructions

[1]↑All software and datasets used to produce the results in this manuscript will be made available upon publication.

Table 2.7. Experimental results.

	HAMP	KNOY	ME	Total
Complete Success	5	10	6	21
Dialogue Success/Navigation Failure	2	3	5	10
Dialogue Failure/Navigation Success	4	1	3	8
Complete Failure	0	0	0	0
Total	11	14	14	39

were interpreted incorrectly and the robot either followed the interpretation correctly or signified an inability to do so correctly, this is a navigation success but dialogue failure. If the navigational instructions were interpreted correctly, but the navigation was unsuccessful, this is a dialogue success but navigation failure. If both were unsuccessful, it is a complete failure. Any trial that required manual intervention was considered a failure.

Table 2.7 shows a breakdown of the trials and their corresponding category on a per-building basis. The dialogue and navigation components had high success rates of 79.4% and 74.3%, respectively. In the case of dialogue, its success rate is slightly better than the validation accuracy reported in Section 2.4.3. In 92.3% of the trials, the dialogue component produced plans whose first 3 concepts were correct, which means that for a strong majority of the time, the robot would have a plan that starts it going in the right direction. In these trials, the robot always succeeded at reaching the intersection corresponding to the current input command by correctly outputting the *transition* classification. However, in 9 of the 10 trials with navigation failure, the robot was unable to predict an accurate driving goal while stopped in the correct intersection, even after rotating, and indicated failure. In the other single trial, manual intervention was employed to prevent the robot from crashing while turning a corner to execute the first command and allowed the robot to successfully complete the second command autonomously. In 5 of the 39 trials, the robot's sequence of commands did not correspond to the environment either due to misinterpretation in dialogue or a mistake on the volunteer's behalf. In all 5 of these trials, the robot correctly drove forward until eventually predicting the *failure* feedback status at the end of a hallway to indicate that the command could not be executed.

2.8 Comparison Experiments

We compared our method to three prior VLN approaches [11]–[13] on the task reported in Section 2.7. While we evaluated using the same kind of live trials reported in Section 2.7, we needed a training set with the same structure as R2R, where all navigation is performed on a fixed graph with images available for all orientations at each vertex, Thus we collected an additional training set, in the style of R2R, using our robot in the same three buildings (PHYS, MSEE, and EE) where we collected the training sets reported in Section 2.4.1 and Section 2.6.2, by placing the robot at fixed waypoints, approximately 3 m apart, and collected 36 images at each waypoint at different camera pan and tilt positions. We then paired the natural-language utterances reported in Section 2.4.1 with the corresponding sequences of these waypoints to create the new training set. This allowed us to construct 2,814 training samples. We trained SF [12], RCM [13] and Babywalk [11] on R2R, using the code base provided by [11], then fine tuned on this new training set.

Evaluating with live trials in novel environments in a different building, however, required us to map the small finite action space output by models trained with the prior VLN approaches to the continuous action space needed to perform our task. During the trials, upon arrival at a waypoint, the robot collected 36 images at the same pan and tilt angles described above, extracted their features as described in [11], and decided whether to drive to an adjacent waypoint or indicate completion. We determined adjacent waypoints by searching the robot's SLAM map for positions in free space that corresponded to positions in front of, behind, left of, and right of the robot's current position.

A strict evaluation of success of a purely autonomous method that cascades the prior models that output discrete navigation actions designed to navigate in a symbolic graph with our code that instead navigates to physical waypoints from those discrete actions leads to very low success rate. Thus we relaxed our success criterion to manually intervene if the robot got too near an obstacle. We moved the robot away from the obstacle to allow the trial to continue.

We conducted eight trials with each of the three methods in KNOY, where our approach had its highest success rate. We positioned the robot in the same starting location and

Table 2.8. Comparison of our system with prior VLN systems. Successful trials were those where the robot stopped in the hallway with the goal location.

	Successful number of trials	Total number of trials	Success rate
SF [12]	0	8	0.0%
RCM [13]	0	8	0.0%
Babywalk [11]	0	8	0.0%
Our System	10	14	**71.4%**

Table 2.9. Navigation statistics from the results reported in Table 2.8. Correct Waypoints is the average number of waypoints the system correctly predicted and navigated to before heading off in a wrong direction. Total Waypoints is the average number of waypoints navigated to before the stop condition is met. Navigation Error is the average driving distance from the ending location to the beginning of the hallway with the goal location.

	Correct Waypoints	Total Waypoints	Navigation Error
SF [12]	3.9	26.8	25.6 m
RCM [13]	3.5	15.8	38.9 m
Babywalk [11]	0.1	8.9	29.4 m
Our System	n/a	n/a	**3.6** m

orientation used in eight of our trials from Section 2.7 and provided the same corresponding text instructions. A trial was concluded when the method predicted the stop condition. If the robot was in the hallway with the goal location, we considered it a success. We then manually drove the robot to the beginning of the hallway with the goal location. Using odometry, we measured the navigation error, which is the driving distance from the ending location to the beginning of the hallway with the goal location. Table 2.8 shows that two of the three methods failed to correctly execute the instructions specified by the input utterances in all eight of their respective trials and one method succeeded only once. Table 2.9 shows a breakdown of these trials, including the number of waypoints traversed and how many waypoints the robot successfully navigated before heading off in a wrong direction. In some, the robot would simply start driving in the opposite direction from that indicated by the instructions. In others, it would miss the turn it was supposed to take or start driving back to where it came from. Our system has a much higher success rate and far lower navigation error.

2.9 Conclusion

We have demonstrated an end-to-end system, deployed on a real robot in a real environment, that can interpret instructions through spoken dialogue as a sequence of commands and then execute those commands. We have also shown that our approach outperforms prior VLN methods when applied to the task of a real robot understanding instructions in a real unknown environment.

REFERENCES

[1] M. Hanheide, M. Göbelbecker, G. S. Horn, A. Pronobis, K. Sjöö, A. Aydemir, P. Jensfelt, C. Gretton, R. Dearden, M. Janicek, *et al.*, "Robot task planning and explanation in open and uncertain worlds," *Artificial Intelligence*, vol. 247, pp. 119–150, 2017.

[2] A. Bauer, K. Klasing, G. Lidoris, Q. Mühlbauer, F. Rohrmüller, S. Sosnowski, T. Xu, K. Kühnlenz, D. Wollherr, and M. Buss, "The autonomous city explorer: Towards natural human-robot interaction in urban environments," *International Journal of Social Robotics*, vol. 1, no. 2, pp. 127–140, 2009.

[3] C. Matuszek, D. Fox, and K. Koscher, "Following directions using statistical machine translation," in *International Conference on Human-Robot Interaction*, 2010, pp. 251–258.

[4] J. H. Oh, A. Suppé, F. Duvallet, A. Boularias, L. Navarro-Serment, M. Hebert, A. Stentz, J. Vinokurov, O. Romero, C. Lebiere, *et al.*, "Toward mobile robots reasoning like humans," in *Conference on Artificial Intelligence*, 2015, pp. 1371–1379.

[5] T. Kollar, S. Tellex, D. Roy, and N. Roy, "Toward understanding natural language directions," in *International Conference on Human-Robot Interaction*, 2010, pp. 259–266.

[6] S. Hemachandra, F. Duvallet, T. M. Howard, N. Roy, A. Stentz, and M. R. Walter, "Learning models for following natural language directions in unknown environments," in *International Conference on Robotics and Automation*, 2015, pp. 5608–5615.

[7] J. Thomason, S. Zhang, R. J. Mooney, and P. Stone, "Learning to interpret natural language commands through human-robot dialog," in *International Joint Conference on Artificial Intelligence*, 2015, pp. 1923–1929.

[8] D. P. Barrett, S. A. Bronikowski, H. Yu, and J. M. Siskind, "Driving under the influence (of language)," *Transactions on Neural Networks and Learning Systems*, vol. 29, no. 7, pp. 2668–2683, 2018.

[9] P. Anderson, Q. Wu, D. Teney, J. Bruce, M. Johnson, N. Sünderhauf, I. Reid, S. Gould, and A. Van Den Hengel, "Vision-and-language navigation: Interpreting visually-grounded navigation instructions in real environments," in *Computer Vision and Pattern Recognition*, 2018, pp. 3674–3683.

[10] J. Thomason, M. Murray, M. Cakmak, and L. Zettlemoyer, "Vision-and-dialog navigation," in *Conference on Robot Learning*, 2020, pp. 394–406.

[11] H. Zhu Wang and- Hu, J. Chen, Z. Deng, V. Jain, E. Ie, and F. Sha, "Babywalk: Going farther in vision-and-language navigation by taking baby steps," in *Annual Meeting of the Association for Computational Linguistics*, 2020, pp. 2539–2556.

[12] D. Fried, R. Hu, V. Cirik, A. Rohrbach, J. Andreas, L.-P. Morency, T. Berg-Kirkpatrick, K. Saenko, D. Klein, and T. Darrell, "Speaker-follower models for vision-and-language navigation," in *Conference on Neural Information Processing Systems*, 2018, pp. 3318–3329.

[13] X. Wang, Q. Huang, A. Celikyilmaz, J. Gao, D. Shen, Y.-F. Wang, W. Y. Wang, and L. Zhang, "Reinforced cross-modal matching and self-supervised imitation learning for vision-language navigation," in *Computer Vision and Pattern Recognition*, 2019, pp. 6629–6638.

[14] W. Hess, D. Kohler, H. Rapp, and D. Andor, "Real-time loop closure in 2D LIDAR SLAM," in *International Conference on Robotics and Automation*, 2016, pp. 1271–1278.

[15] Google, *Cloud speech-to-text*, Retrieved Mar 1, 2019 from `https://cloud.google.com/speech-to-text`, 2019.

[16] D. Chen and C. Manning, "A fast and accurate dependency parser using neural networks," in *Empirical Methods in Natural Language Processing*, 2014, pp. 740–750.

[17] N. M. Bhat, *Python text to speech library*, Retrieved Dec 1, 2018 from `https://pypi.org/project/pyttsx3/`, 2017.

[18] J. Redmon and A. Farhadi, "YOLOv3: An incremental improvement," *arXiv:1804.02767*, 2018.

[19] R. Grompone Von Gioi, J. Jakubowicz, J. M. Morel, and G. Randall, "LSD: A fast line segment detector with a false detection control," *Transactions on Pattern Analysis and Machine Intelligence*, vol. 32, no. 4, pp. 722–732, 2010.

[20] Google, *Detect text in images*, Retrieved Apr 1, 2019 from `https://cloud.google.com/vision/docs/ocr`, 2019.

[21] D. Agarwal, *LIDAR-camera-calibration in ROS*, Retrieved Jun 1, 2018, from `https://github.com/agarwa65/lidar_camera_calibration`, 2018.

[22] H. W. Kuhn, "The hungarian method for the assignment problem," *Naval research logistics quarterly*, vol. 2, no. 1-2, pp. 83–97, 1955.

[23] D. H. Douglas and T. K. Peucker, “Algorithms for the reduction of the number of points required to represent a digitized line or its caricature,” *Cartographica: The International Journal for Geographic Information and Geovisualization*, vol. 10, no. 2, pp. 112–122, 1973.

[24] W. Shi and J. Samarabandu, “Investigating the performance of corridor and door detection algorithms in different environments,” in *International Conference on Information and Automation*, 2006, pp. 206–211.

[25] L. Del Pero, J. Bowdish, D. Fried, B. Kermgard, E. Hartley, and K. Barnard, “Bayesian geometric modeling of indoor scenes,” in *Computer Vision and Pattern Recognition*, 2012, pp. 2719–2726.

[26] S. Ren, K. He, R. Girshick, and J. Sun, “Faster R-CNN: Towards real-time object detection with region proposal networks,” in *Advances in Neural Information Processing Systems*, 2015, pp. 91–99.

[27] M. Arduengo, C. Torras, and L. Sentis, *Robust and adaptive door operation with a mobile manipulator robot*, 2019. arXiv: 1902.09051 [cs.RO].

[28] A. Weiss, J. Igelsböck, M. Tscheligi, A. Bauer, K. Kühnlenz, D. Wollherr, and M. Buss, “Robots asking for directions - the willingness of passers-by to support robots,” in *International Conference on Human-Robot Interaction*, 2010, pp. 23–30.

[29] A. Robicquet, A. Sadeghian, A. Alahi, and S. Savarese, “Learning social etiquette: Human trajectory understanding in crowded scenes,” in *European Conference on Computer Vision*, 2016, pp. 549–565.

[30] A. Alahi, K. Goel, V. Ramanathan, A. Robicquet, L. Fei-Fei, and S. Savarese, “Social LSTM: Human trajectory prediction in crowded spaces,” in *Computer Vision and Pattern Recognition*, 2016, pp. 961–971.

[31] A. Bauer, B. Gonsior, D. Wollherr, and M. Buss, “Heuristic rules for human-robot interaction based on principles from linguistics-asking for directions,” in *New Frontiers in Human-Robot Interaction*, 2009, pp. 24–30.

[32] A. Bauer, D. Wollherr, and M. Buss, “Information retrieval system for human-robot communication-asking for directions,” in *International Conference on Robotics and Automation*, 2009, pp. 4150–4155.

[33] A. Boularias, F. Duvallet, J. Oh, and A. Stentz, “Grounding spatial relations for outdoor robot navigation,” in *International Conference on Robotics and Automation*, 2015, pp. 1976–1982.

[34] J. Thomason, A. Padmakumar, J. Sinapov, N. Walker, Y. Jiang, H. Yedidsion, J. Hart, P. Stone, and R. Mooney, "Jointly improving parsing and perception for natural language commands through human-robot dialog," *Journal of Artificial Intelligence Research*, vol. 67, pp. 327–374, 2020.

[35] S. Lukin, F. Gervits, C. Hayes, P. Moolchandani, A. Leuski, J. G. Rogers III, C. S. Amaro, M. Marge, C. Voss, and D. Traum, "Scoutbot: A dialogue system for collaborative navigation," in *Annual Meeting of the Association for Computational Linguistics*, 2018, pp. 93–98.

[36] V. Blukis, Y. Terme, E. Niklasson, R. A. Knepper, and Y. Artzi, "Learning to map natural language instructions to physical quadcopter control using simulated flight," in *Conference on Robot Learning*, 2020, pp. 1415–1438.

[37] S. Banerjee, J. Thomason, and J. J. Corso, "The RobotSlang benchmark: Dialog-guided robot localization and navigation," in *Conference on Robot Learning*, 2020, pp. 1–10.

[38] N. Sünderhauf, F. Dayoub, S. McMahon, B. Talbot, R. Schulz, P. Corke, G. Wyeth, B. Upcroft, and M. Milford, "Place categorization and semantic mapping on a mobile robot," in *International Conference on Robotics and Automation*, 2016, pp. 5729–5736.

[39] D. Pangercic, B. Pitzer, M. Tenorth, and M. Beetz, "Semantic object maps for robotic housework-representation, acquisition and use," in *International Conference on Intelligent Robots and Systems*, 2012, pp. 4644–4651.

[40] S. A. Stoeter, F. Le Mauff, and N. P. Papanikolopoulos, "Real-time door detection in cluttered environments," in *International Symposium on Intelligent Control*, 2000, pp. 187–192.

[41] D. Kim and R. Nevatia, "Recognition and localization of generic objects for indoor navigation using functionality," *Image and Vision Computing*, vol. 16, no. 11, pp. 729–743, 1998.

[42] G. Olmschenk and Z. Zhu, "3D hallway modeling using a single image," in *Computer Vision and Pattern Recognition Workshops*, 2014, pp. 158–164.

[43] P. Anderson, A. Shrivastava, J. Truong, A. Majumdar, D. Parikh, D. Batra, and S. Lee, "Sim-to-real transfer for vision-and-language navigation," in *Conference on Robot Learning*, 2020, pp. 1–11.

[44] J. Roh, C. Paxton, A. Pronobis, A. Farhadi, and D. Fox, "Conditional driving from natural language instructions," in *Conference on Robot Learning*, 2020, pp. 540–551.

[45] N. Sriram, T. Maniar, J. Kalyanasundaram, V. Gandhi, B. Bhowmick, and K. M. Krishna, "Talk to the vehicle: Language conditioned autonomous navigation of self driving cars," in *International Conference on Intelligent Robots and Systems*, 2019, pp. 5284–5290.

[46] M. Deitke, W. Han, A. Herrasti, A. Kembhavi, E. Kolve, R. Mottaghi, J. Salvador, D. Schwenk, E. VanderBilt, M. Wallingford, L. Weihls, M. Yatskar, and A. Farhadi, "RoboTHOR: An open simulation-to-real embodied AI platform," in *Computer Vision and Pattern Recognition*, 2020, pp. 3164–3174.

[47] Y. Zhu, R. Mottaghi, E. Kolve, J. J. Lim, A. Gupta, L. Fei-Fei, and A. Farhadi, "Target-driven visual navigation in indoor scenes using deep reinforcement learning," in *International Conference on Robotics and Automation*, 2017, pp. 3357–3364.

[48] H. Chen, A. Suhr, D. Misra, N. Snavely, and Y. Artzi, "Touchdown: Natural language navigation and spatial reasoning in visual street environments," in *Computer Vision and Pattern Recognition*, 2019, pp. 12 538–12 547.

[49] H. de Vries, K. Shuster, D. Batra, D. Parikh, J. Weston, and D. Kiela, "Talk the walk: Navigating New York City through grounded dialogue," *arXiv:1807.03367*, 2018.

[50] C. Theobalt, J. Bos, T. Chapman, A. Espinosa-Romero, M. Fraser, G. Hayes, E. Klein, T. Oka, and R. Reeve, "Talking to Godot: Dialogue with a mobile robot," in *International Conference on Intelligent Robots and Systems*, 2002, pp. 1338–1343.

[51] D. L. Chen and R. J. Mooney, "Learning to interpret natural language navigation instructions from observations," in *Conference on Artificial Intelligence*, 2011, pp. 859–865.

[52] H. Mei, M. Bansal, and M. Walter, "Listen, attend, and walk: Neural mapping of navigational instructions to action sequences," in *Conference on Artificial Intelligence*, 2016, pp. 2772–2778.

[53] T. M. Howard, S. Tellex, and N. Roy, "A natural language planner interface for mobile manipulators," in *International Conference on Robotics and Automation*, 2014, pp. 6652–6659.

[54] Z. Hu, J. Pan, T. Fan, R. Yang, and D. Manocha, "Safe navigation with human instructions in complex scenes," *Robotics and Automation Letters*, vol. 4, no. 2, pp. 753–760, 2019.

[55] G. Sepulveda, J. C. Niebles, and A. Soto, "A deep learning based behavioral approach to indoor autonomous navigation," in *International Conference on Robotics and Automation*, 2018, pp. 4646–4653.

[56] X. Zang, A. Pokle, M. Vázquez, K. Chen, J. C. Niebles, A. Soto, and S. Savarese, “Translating navigation instructions in natural language to a high-level plan for behavioral robot navigation,” *arXiv:1810.00663*, 2018.

[57] R. Meena, G. Skantze, and J. Gustafson, “A data-driven approach to understanding spoken route directions in human-robot dialogue,” in *Conference of the International Speech Communication Association*, 2012, pp. 226–229.

[58] Microsoft, *Azure speech-to-text*, Retrieved Apr 1, 2020 from https://azure.microsoft.com/en-us/services/cognitive-services/speech-to-text/, 2020.

[59] C. Schäfer and D. Tran, *Headliner*, Retrieved Mar 14, 2020 from https://github.com/as-ideas/headliner, 2020.

[60] Y. Liu and M. Lapata, “Text summarization with pretrained encoders,” in *Empirical Methods in Natural Language Processing*, 2019, pp. 3730–3740.

[61] E. Marder-Eppstein, *Move_base*, Retrieved Sep 1, 2018 from http://wiki.ros.org/move_base, 2018.

A. BUILDING FLOOR PLANS

Areas of the map that are green represent areas that the robot can drive in. Areas of the map that are red represent areas that the robot cannot drive in. These include carpeted areas and hallways that are narrower than the robot's circumscribed radius.

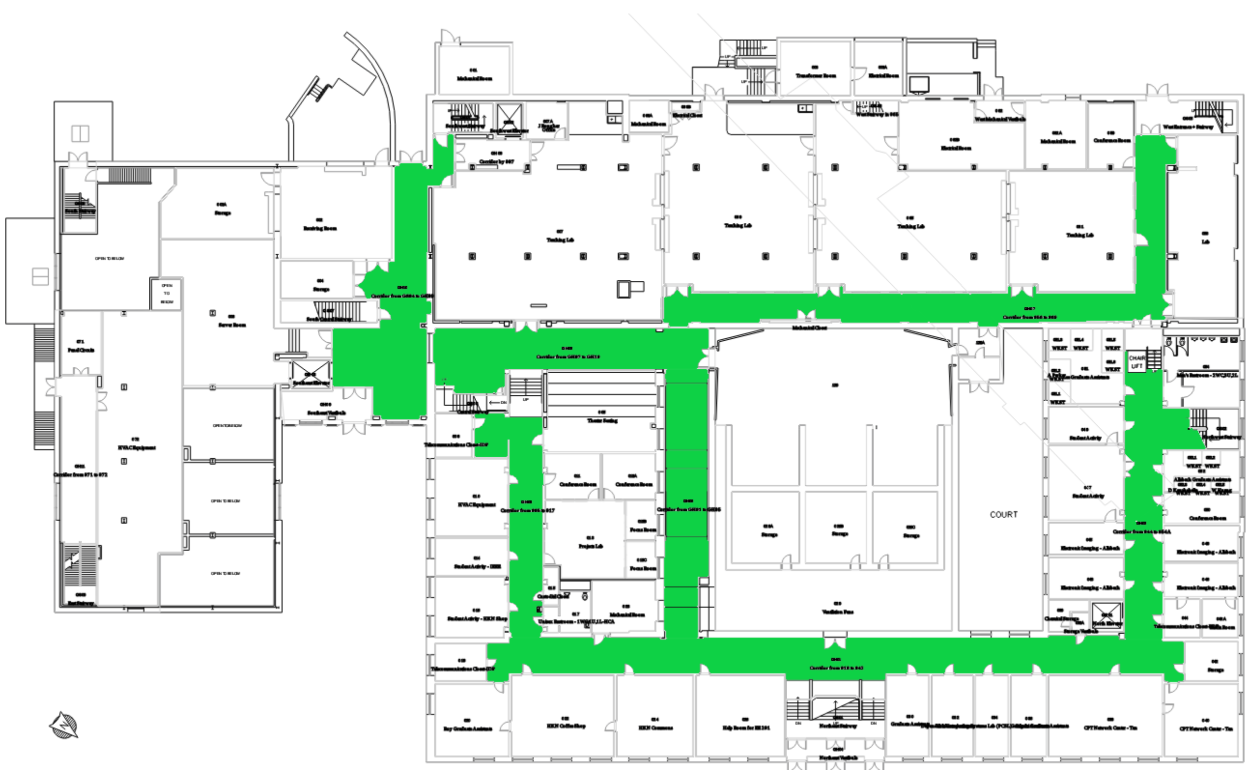


Figure A.1. Building Name: EE. Floor Number: 0.

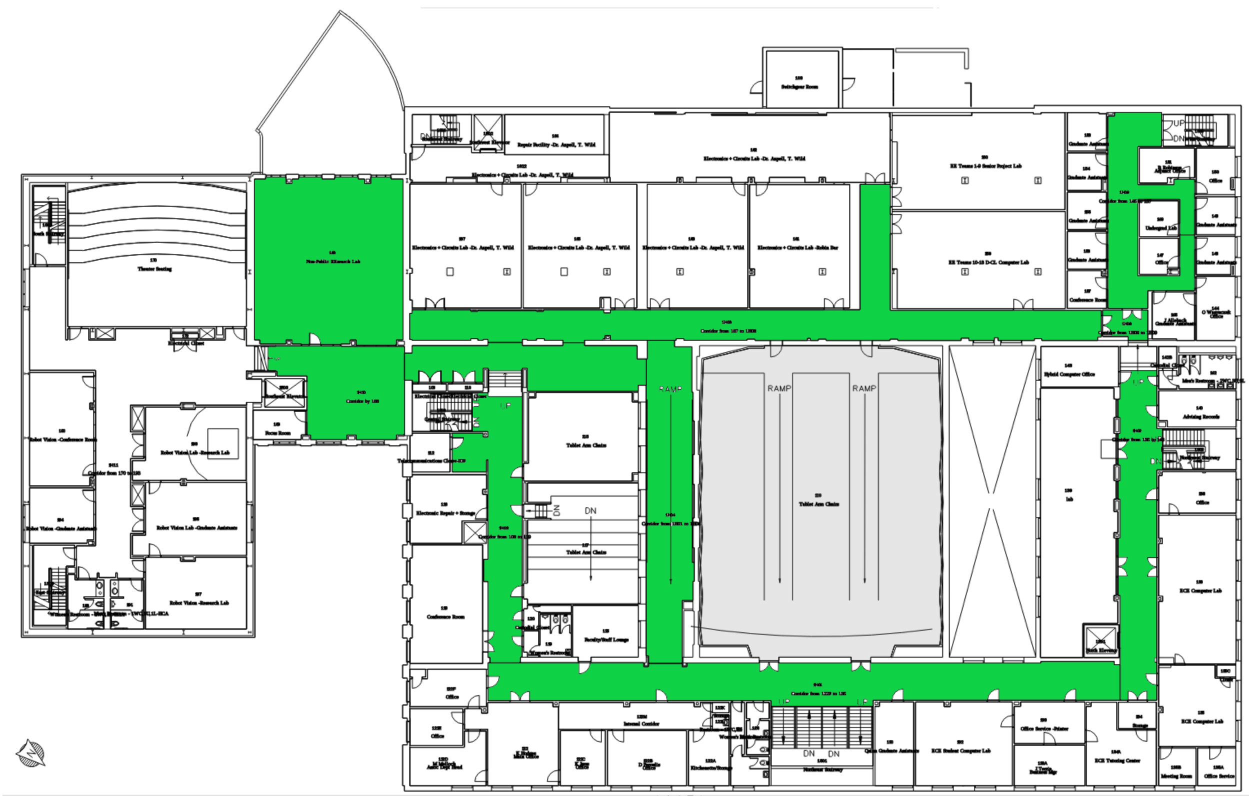


Figure A.2. Building Name: EE. Floor Number: 1.

Figure A.3. Building Name: EE. Floor Number: 2.

Figure A.4. Building Name: EE. Floor Number: 3.

Figure A.5. Building Name: MSEE. Floor Number: 0.

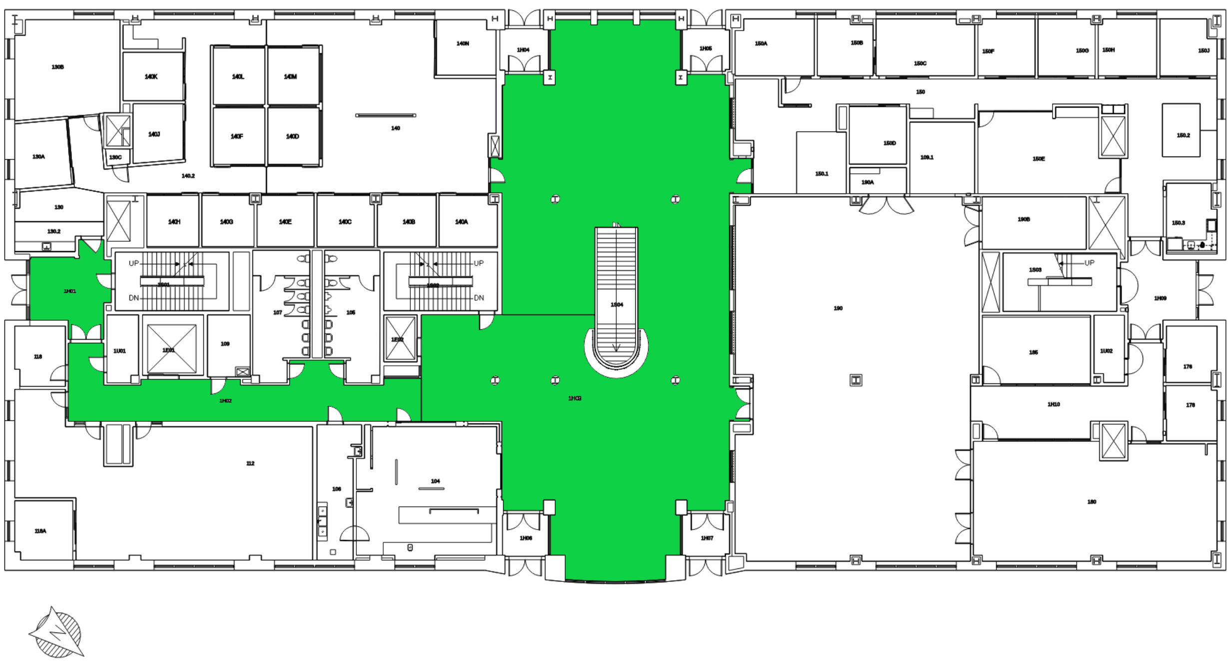


Figure A.6. Building Name: MSEE. Floor Number: 1.

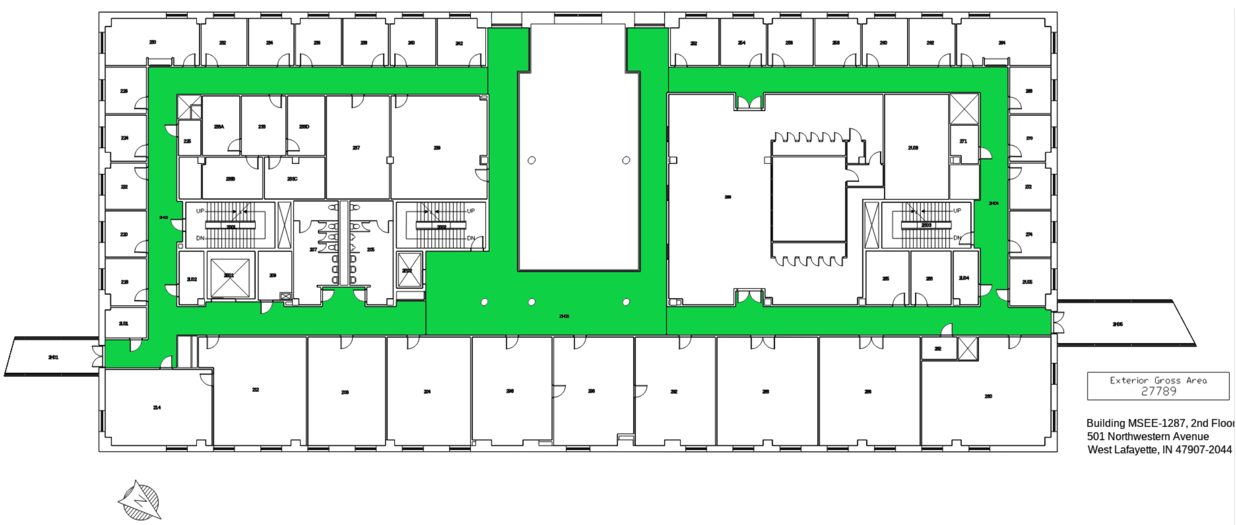


Figure A.7. Building Name: MSEE. Floor Number: 2.

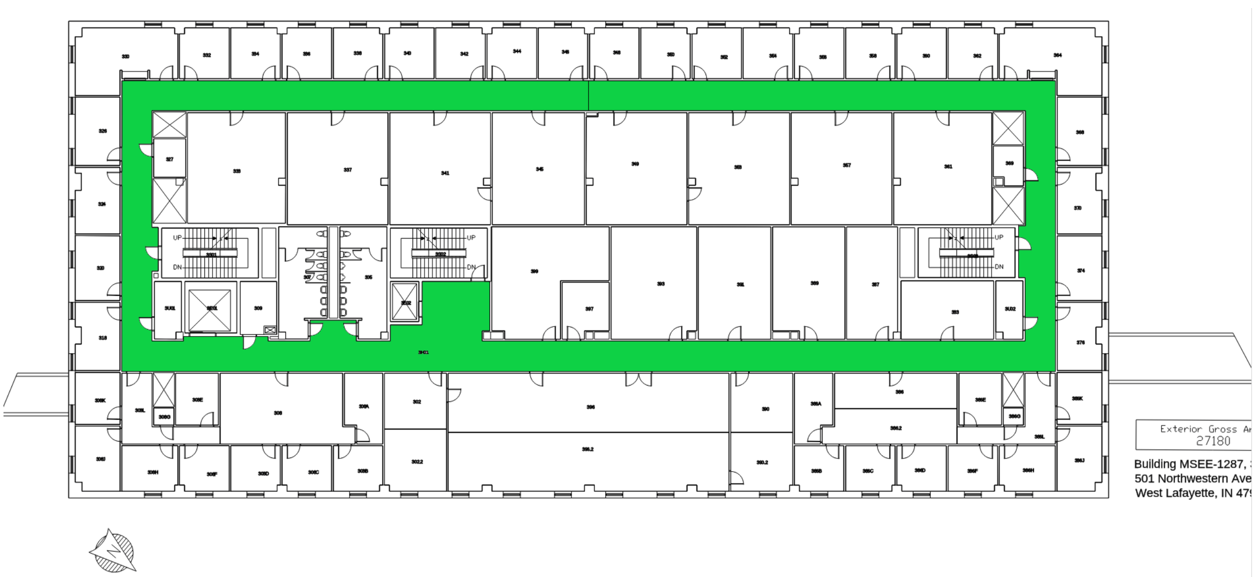


Figure A.8. Building Name: MSEE. Floor Number: 3.

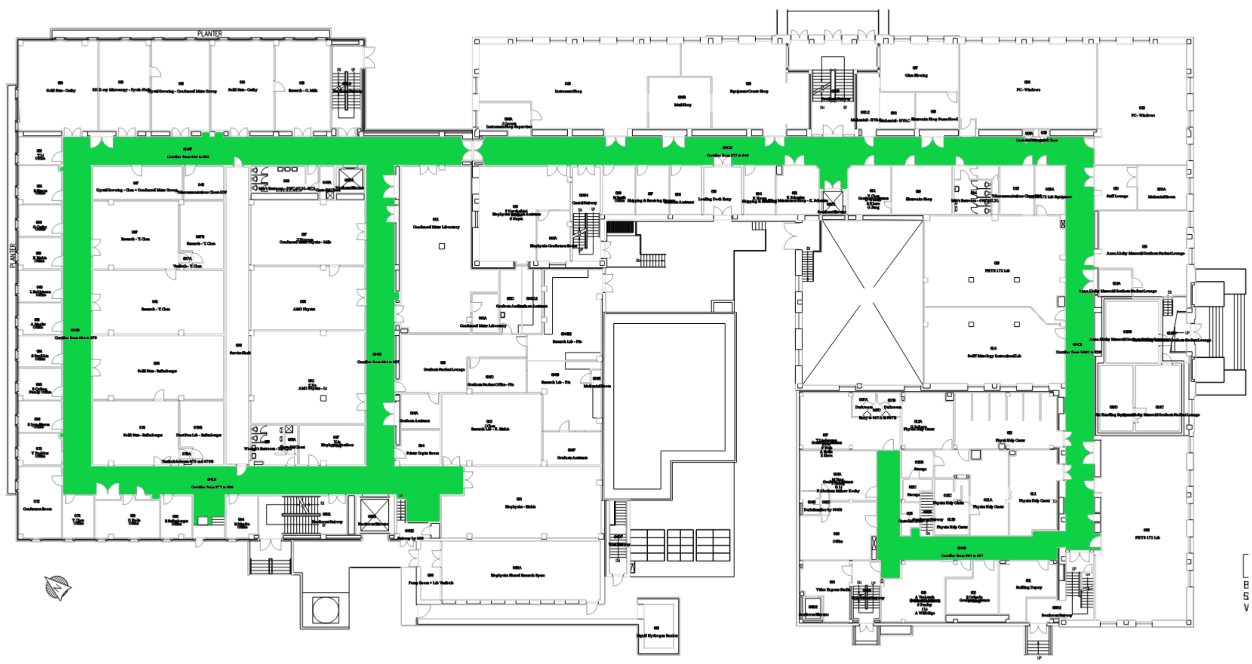


Figure A.9. Building Name: PHYS. Floor Number: 0.

Figure A.10. Building Name: PHYS. Floor Number: 1.

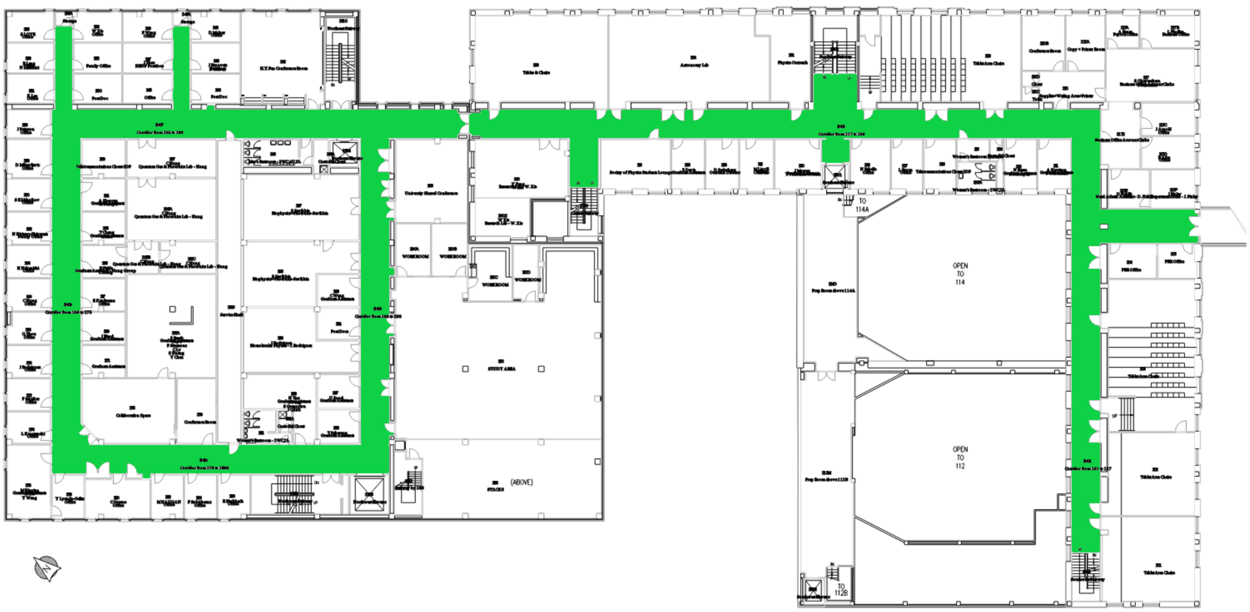


Figure A.11. Building Name: PHYS. Floor Number: 2.

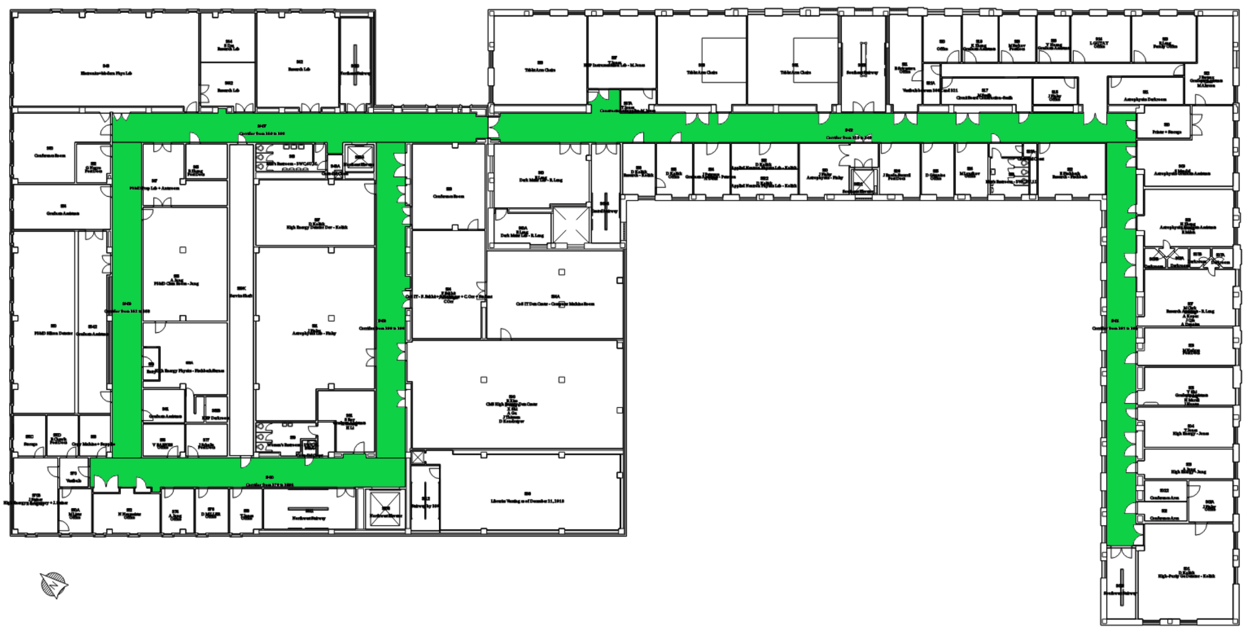


Figure A.12. Building Name: PHYS. Floor Number: 3.

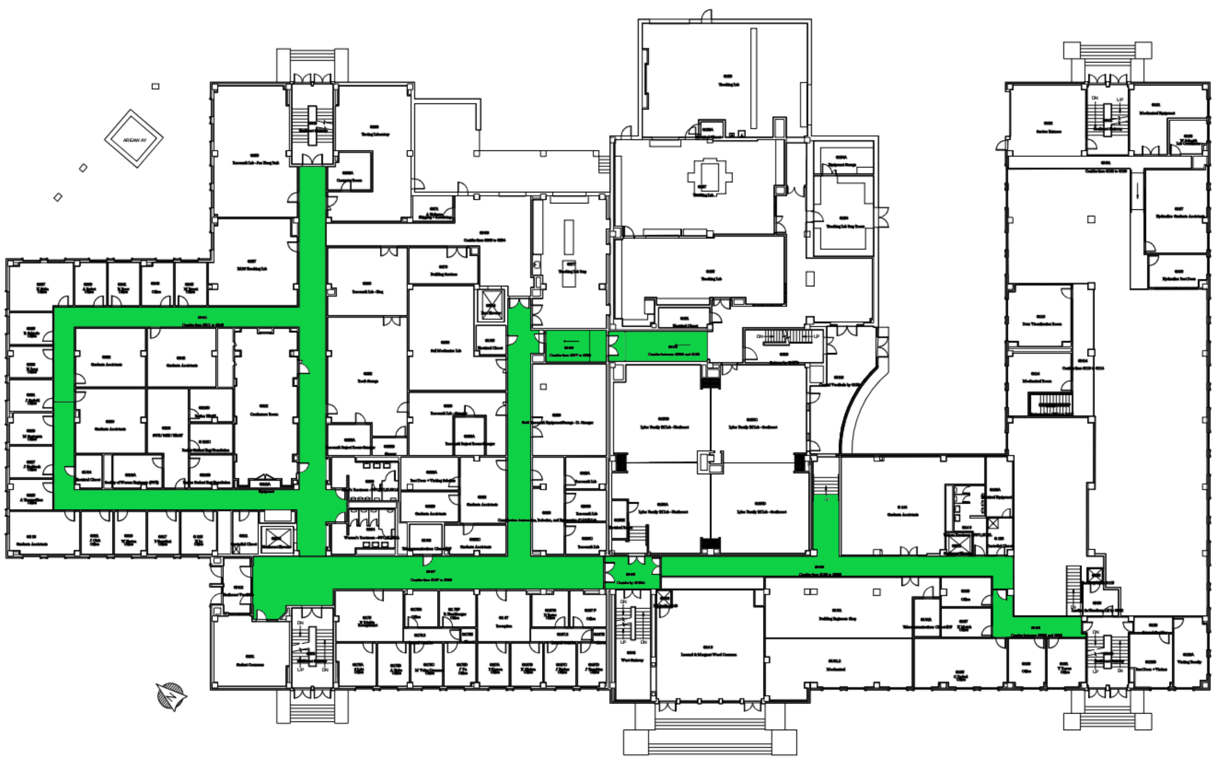


Figure A.13. Building Name: HAMP. Floor Number: 0.

Figure A.14. Building Name: HAMP. Floor Number: 1.

Figure A.15. Building Name: HAMP. Floor Number: 2.

Figure A.16. Building Name: HAMP. Floor Number: 3.

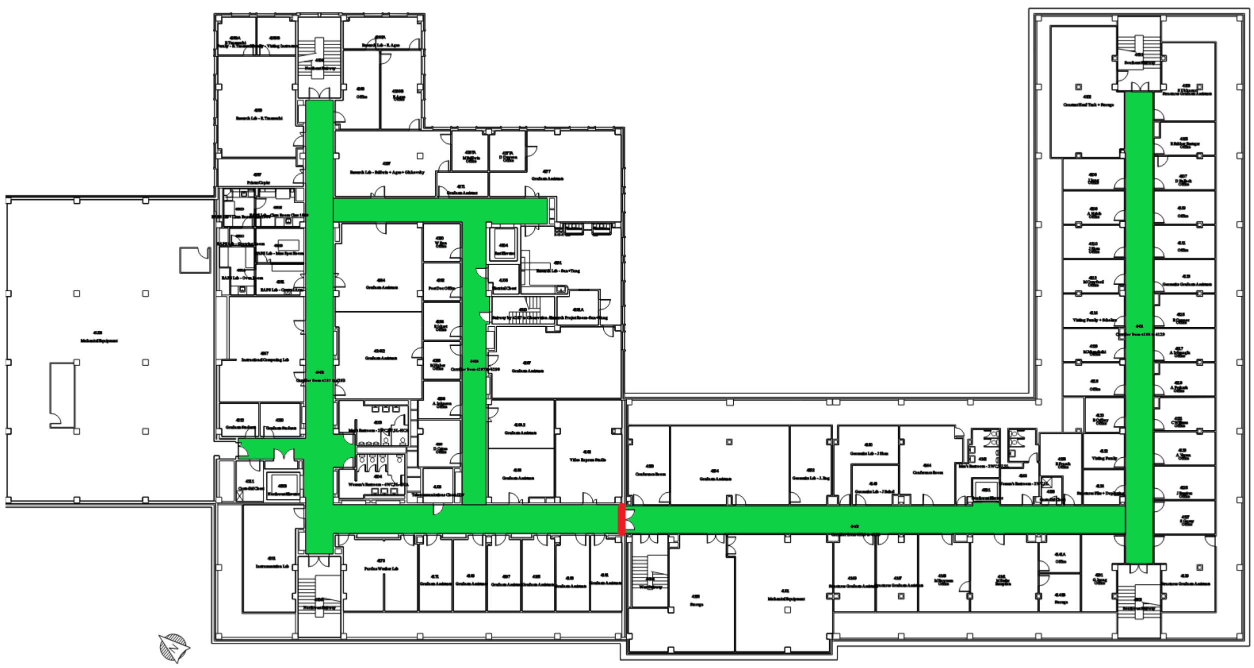


Figure A.17. Building Name: HAMP. Floor Number: 4.

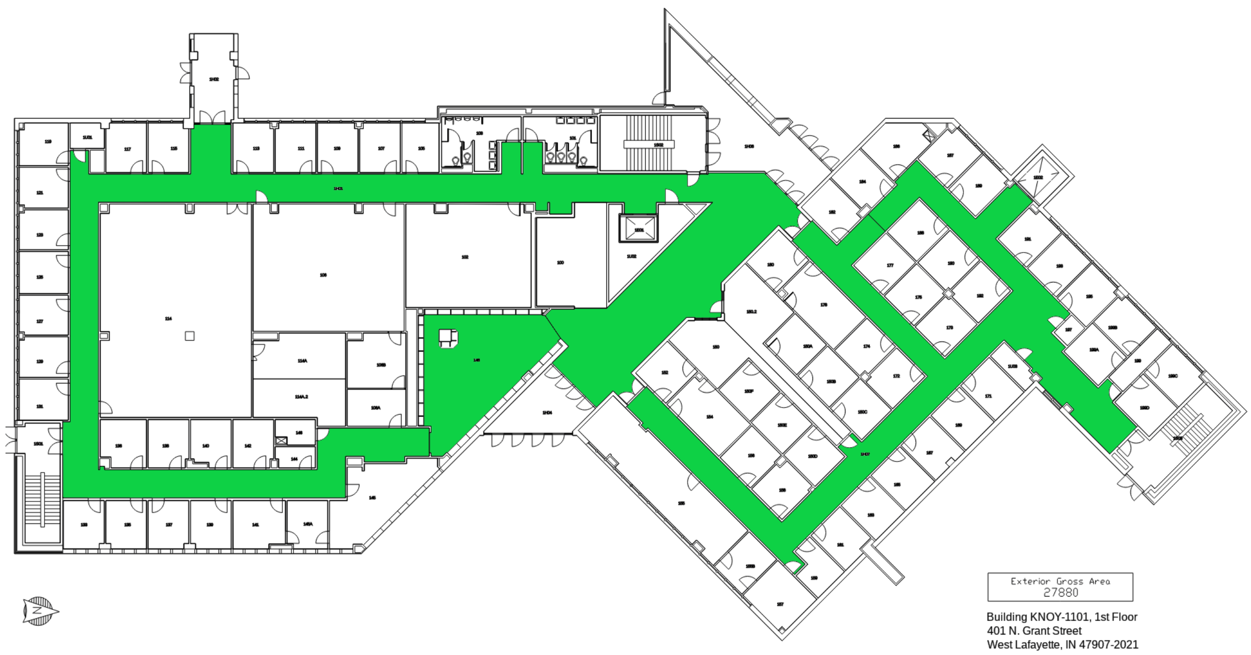


Figure A.18. Building Name: KNOY. Floor Number: 1.

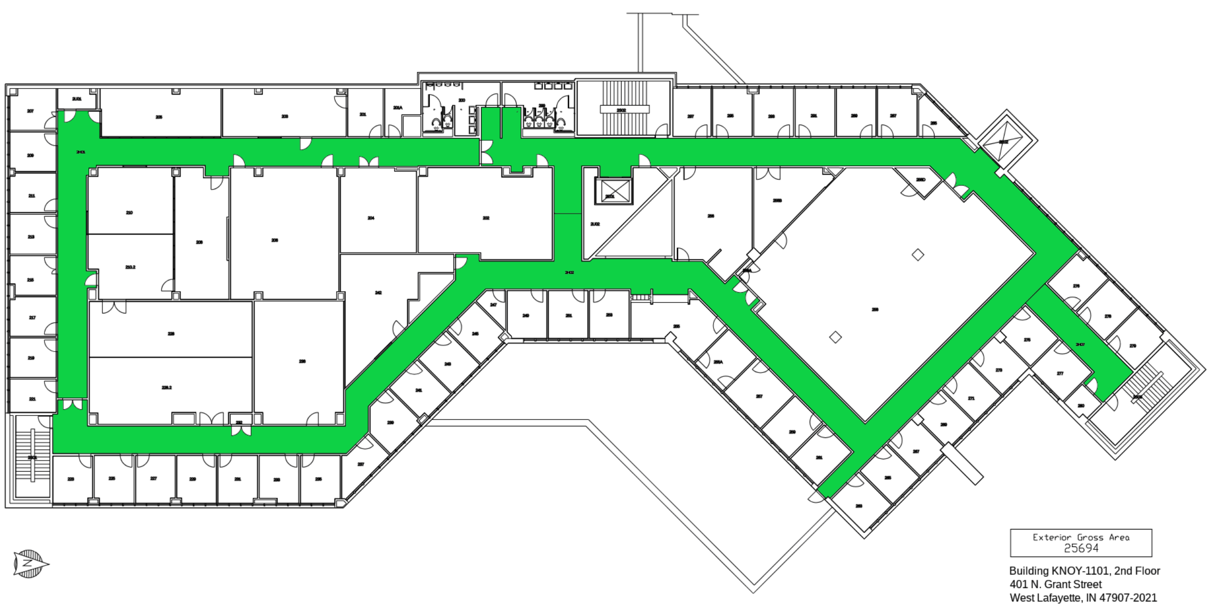


Figure A.19. Building Name: KNOY. Floor Number: 2.

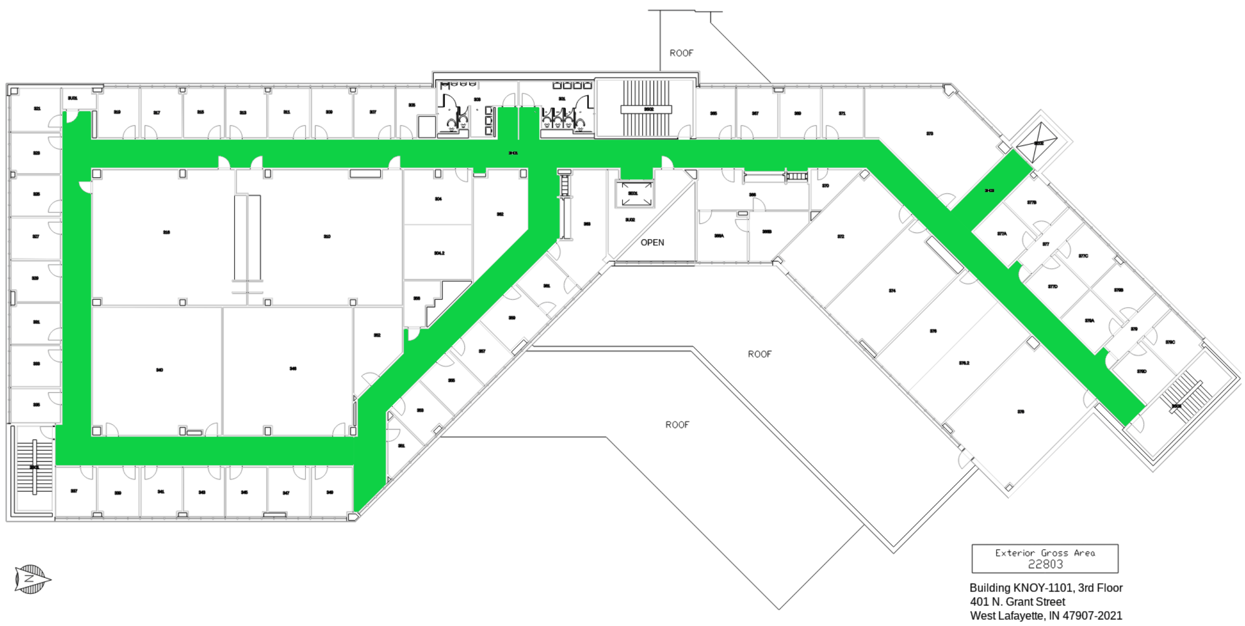


Figure A.20. Building Name: KNOY. Floor Number: 3.

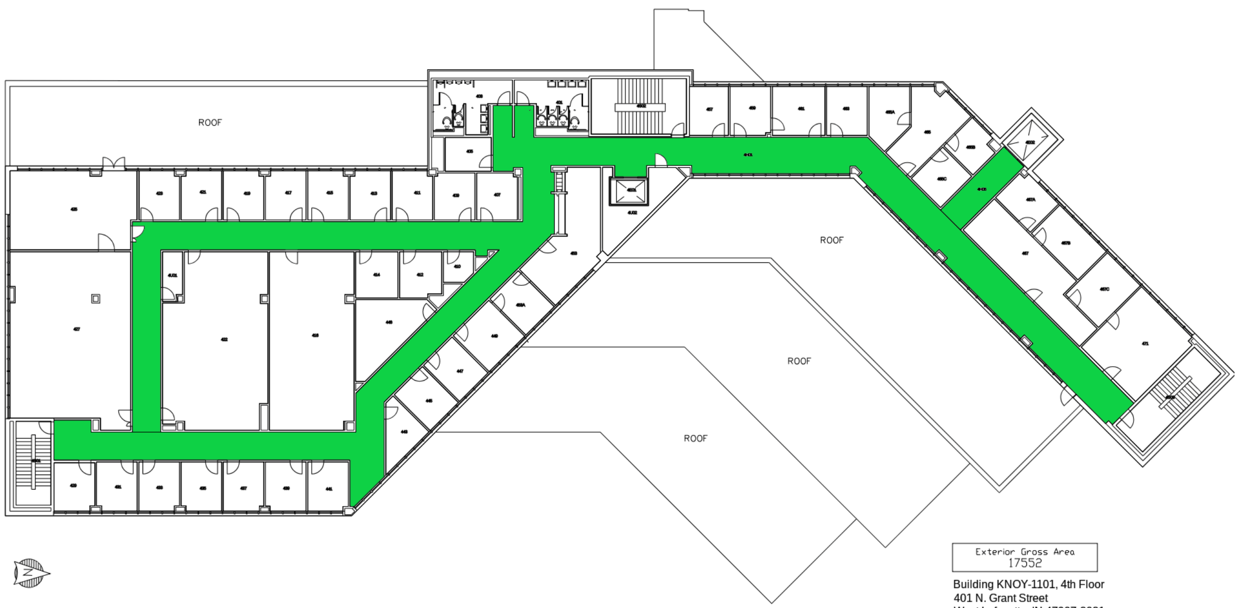


Figure A.21. Building Name: KNOY. Floor Number: 4.

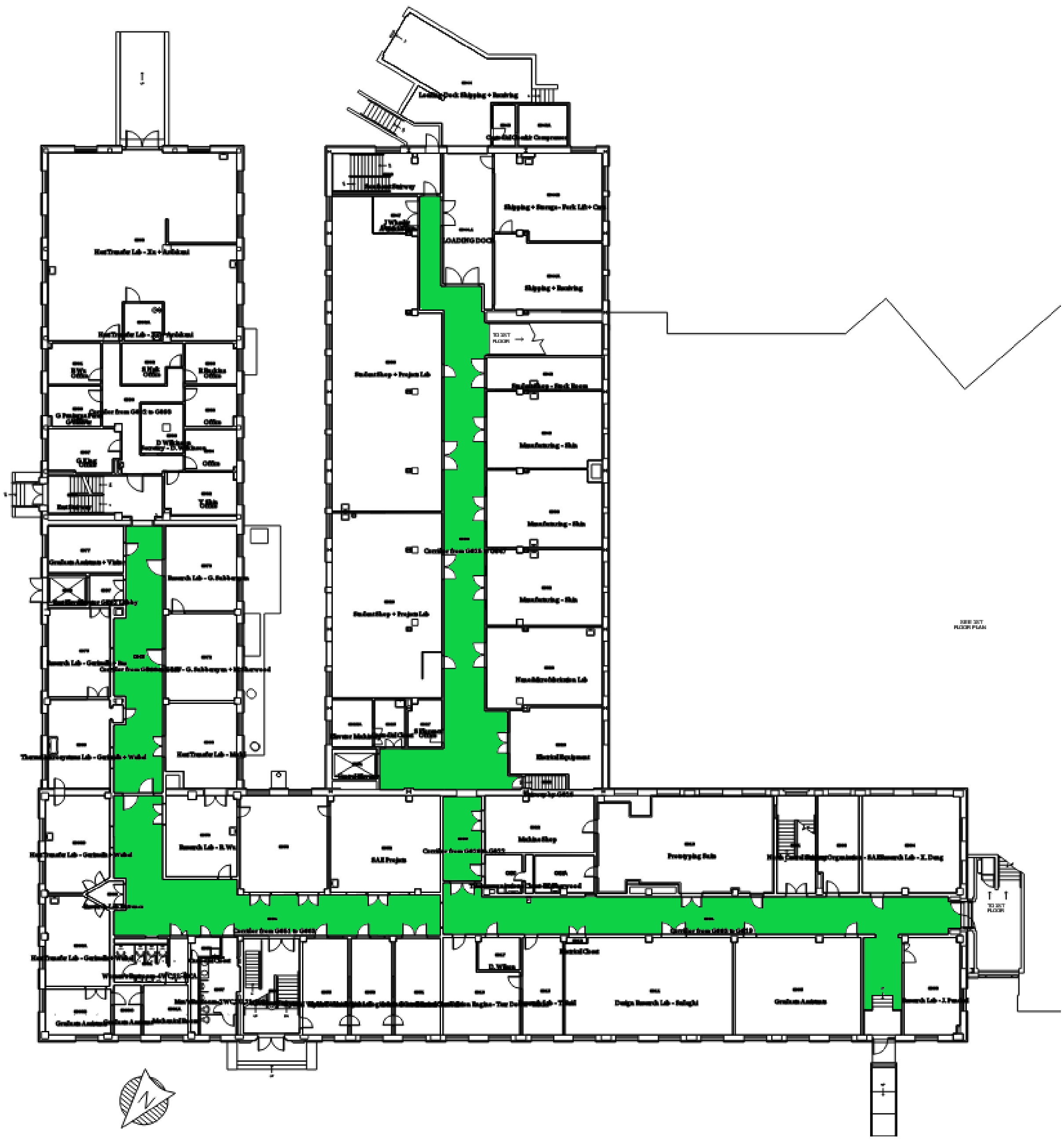


Figure A.22. Building Name: ME. Floor Number: 0.

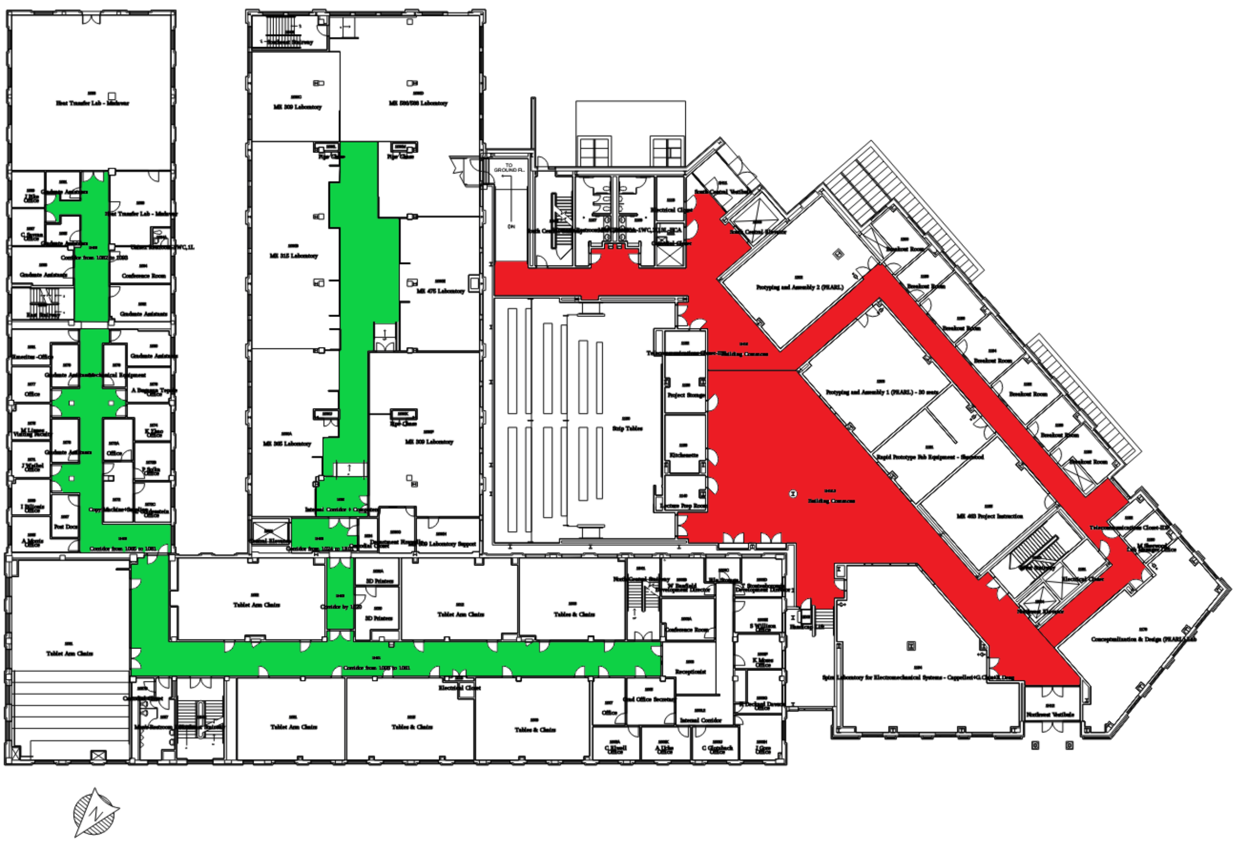


Figure A.23. Building Name: ME. Floor Number: 1.

Figure A.24. Building Name: ME. Floor Number: 2.

Figure A.25. Building Name: ME. Floor Number: 3.

VITA

Thomas Victor Ilyevsky received his B.S. in Electrical and Computer Engineering from Cornell University in 2016. He is currently pursuing a Ph.D. in Electrical and Computer Engineering at Purdue University. His research primarily focuses on artificial intelligence, computer vision, and robotics.