Constrained Collective Movement in Human-Robot Teams

Joshua Fagan
jfagan2@vols.utk.edu

Follow this and additional works at: https://trace.tennessee.edu/utk_graddiss

Part of the Artificial Intelligence and Robotics Commons

Recommended Citation
https://trace.tennessee.edu/utk_graddiss/7728

This Dissertation is brought to you for free and open access by the Graduate School at TRACE: Tennessee Research and Creative Exchange. It has been accepted for inclusion in Doctoral Dissertations by an authorized administrator of TRACE: Tennessee Research and Creative Exchange. For more information, please contact trace@utk.edu.
To the Graduate Council:

I am submitting herewith a dissertation written by Joshua Fagan entitled "Constrained Collective Movement in Human-Robot Teams." I have examined the final electronic copy of this dissertation for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Doctor of Philosophy, with a major in Computer Science.

Lynne E Parker, Major Professor

We have read this dissertation and recommend its acceptance:

Lynne E Parker, Hairong Qi, Daniel C Rucker, Amir Sadovnik

Accepted for the Council:

Dixie L. Thompson
Vice Provost and Dean of the Graduate School

(Original signatures are on file with official student records.)
Constrained Collective Movement in Human-Robot Teams

A Dissertation Presented for the
Doctor of Philosophy
Degree
The University of Tennessee, Knoxville

Joshua Fagan
December 2022
For my wife, Kelsey Fagan, and my daughter, Audrey Fagan; the greatest positive influences on my motivation, strength, and self-discipline.
Acknowledgements

I would like to thank my advisor, Dr. Lynne Parker, for her mentorship of my academic development, meticulous feedback, and encouragement in difficult times. Thank you to my committee members, Dr. Hairong Qi, Dr. Caleb Rucker, and Dr. Amir Sadovnik, for their constructive guidance on my dissertation as well as their patience and understanding. I am grateful for the lab members I have had the pleasure of knowing over the years. Thank you, Eric Garrison, Corey Johnson, Dr. Chris Reardon, and Dr. Sudarshan Srinivasan for always being willing to bounce ideas around and sharing constructive feedback.

On a more personal level, I would like to thank members of my family whose love and support has been invaluable to me. Thank you to my mother-in-law and father-in-law, Brenda and David Donner, for your love for me and my family and your encouragement to push through adversity. Thank you also to my dad and mom, Marbury and Pattie Fagan, for your excitement and steadfast encouragement in all of my passions and endeavours. I want to thank my sister and brother-in-law, Sarah and Alex Shafran, for your love and support of my dreams. I also want to thank my brother and sister-in-law, Jonathan and Miranda, for paving the way and being a team I can share academic life experiences with. I owe an incredible debt to my wife, Kelsey Fagan. Thank you for your constant encouragement, for sharing your strength with me, and for all the hard work you have put into supporting me and making this all possible. Lastly, I want to thank my daughter, Audrey Fagan. Your excitement and love makes every day better.
Abstract

This research focuses on improving human-robot co-navigation for teams of robots and humans navigating together as a unit while accomplishing a desired task. Frequently, the team’s co-navigation is strongly influenced by a predefined Standard Operating Procedure (SOP), which acts as a high-level guide for where agents should go and what they should do. In this work, I introduce the concept of Constrained Collective Movement (CCM) of a team to describe how members of the team perform inter-team and intra-team navigation to execute a joint task while balancing environmental and application-specific constraints. This work advances robots’ abilities to participate alongside humans in applications such as urban search and rescue, firefighters searching for people in a burning building, and military teams performing a building clearing operation. Incorporating robots on such teams could reduce the number of human lives put in danger while increasing the team’s ability to conduct beneficial tasks such as carrying life-saving equipment to stranded people.

Most previous work on generating more complex collaborative navigation for human-robot teams focuses solely on using model-based methods. These methods usually suffer from the need for hard-coding the rules to follow, which can require much time and domain knowledge and can lead to unnatural behavior.

This dissertation investigates merging high-level model-based knowledge representation with low-level behavior cloning to achieve CCM of a human-robot team performing collaborative co-navigation. To evaluate the approach, experiments are performed in simulation with the detail-rich game design engine Unity. Experiments show that the designed approach can learn elements of high-level behaviors with accuracies up to 88%. Additionally, the approach is shown to learn low-level robot control behaviors with accuracies up to 89%.

To the best of my knowledge, this is the first attempt to blend classical AI methods with state-of-the-art machine learning methods for human-robot team collaborative co-navigation. This not only allows for better human-robot team co-navigation, but also has implications for improving other teamwork-based human-robot applications such as joint manufacturing and social assistive robotics.
# Table of Contents

1 Introduction .................................................. 1
   1.1 Constrained Collective Movement ......................... 1
   1.2 Problem Statement and Challenges ......................... 2
      1.2.1 Problem Statement .................................. 2
      1.2.2 Challenges ........................................ 3
   1.3 Overview of Approach ..................................... 4
   1.4 Contributions ........................................... 5
   1.5 Dissertation Overview .................................. 5

2 Literature Review ............................................ 6
   2.1 Human-Robot Co-Navigation ............................... 6
   2.2 Model-Based Approaches ................................ 8
   2.3 Learning-Based Approaches ............................... 9

3 Approach ..................................................... 11
   3.1 Preprocessing and Feature Engineering ................... 12
   3.2 Learning SOP Script Structure Representation .......... 15
      3.2.1 Script Element Extraction .......................... 16
      3.2.2 Transition Extraction ............................... 17
   3.3 Learn Control Commands ................................ 19
   3.4 Incorporate SOP Script with Control Commands ......... 20
   3.5 Summary and Discussion ................................ 20

4 Experiments and Results ..................................... 24
   4.1 Behaviors ................................................ 24
   4.2 Environments ............................................ 27
   4.3 Data Collected .......................................... 34
   4.4 Script Representation Learning Results ................. 37
      4.4.1 Phase Extraction ................................... 39
      4.4.2 Step Extraction .................................... 42
      4.4.3 Transition Structuring ............................... 48
      4.4.4 Command Control Network ........................... 51
   4.5 Summary and Discussion ................................ 71

5 Summary and Conclusions .................................... 76
Bibliography  78

A Detailed Description of Baseline Behaviors  86
  A.1 Room Exploration  .................................................. 86
  A.2 T-Intersection Navigation  ....................................... 87
  A.3 Corner Navigation  .................................................. 92

B Detailed Description of Behavior Variants  94
  B.1 Room Entry Variants  ............................................... 94
  B.2 T-Intersection Variants  ......................................... 97
    B.2.1 Corner Variants  ............................................... 97

Vita  102
Acronyms

CCM  Constrained Collective Movement. iv, 1–4, 6, 9, 11, 13, 14, 20, 22–24, 42, 45, 48, 71, 76, 77

CCN  Command Control Network. viii, ix, 12–16, 19, 20, 22–24, 34, 47, 51, 53–76

CNN  Convolutional Neural Network. 10, 19–21, 38, 51, 53–55, 58, 59, 61, 64, 66, 69

GMM  Gaussian Mixture Model. viii, 16, 20, 37–39, 42, 45, 47, 48, 76


VAE  Variational Autoencoder. 20
# List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>Overview of learning phase.</td>
<td>13</td>
</tr>
<tr>
<td>3.2</td>
<td>Overview of execution phase.</td>
<td>14</td>
</tr>
<tr>
<td>3.3</td>
<td>Example restaurant script structured as Petri net.</td>
<td>18</td>
</tr>
<tr>
<td>3.4</td>
<td>Architecture of network used to train end-to-end robot controller.</td>
<td>21</td>
</tr>
<tr>
<td>3.5</td>
<td>Architecture of network with categorical script information merged.</td>
<td>21</td>
</tr>
<tr>
<td>3.6</td>
<td>Architecture of network with visual script information merged.</td>
<td>21</td>
</tr>
<tr>
<td>4.1</td>
<td>Visual of hallway behavior.</td>
<td>26</td>
</tr>
<tr>
<td>4.2</td>
<td>Script steps of team <strong>entering a room</strong>.</td>
<td>28</td>
</tr>
<tr>
<td>4.3</td>
<td>Script steps of team <strong>traversing an intersection in their hallway</strong>.</td>
<td>29</td>
</tr>
<tr>
<td>4.4</td>
<td>Script steps of team <strong>taking an intersection in their hallway</strong>.</td>
<td>30</td>
</tr>
<tr>
<td>4.5</td>
<td>Script steps of team <strong>intersecting a hallway</strong>.</td>
<td>31</td>
</tr>
<tr>
<td>4.6</td>
<td>Script steps of team <strong>navigating a corner</strong>.</td>
<td>32</td>
</tr>
<tr>
<td>4.7</td>
<td>Unity training simulation environment.</td>
<td>35</td>
</tr>
<tr>
<td>4.8</td>
<td>Unity testing simulation environment.</td>
<td>36</td>
</tr>
<tr>
<td>4.9</td>
<td>Original and overlaid map images.</td>
<td>38</td>
</tr>
<tr>
<td>4.10</td>
<td>Script phase VaDE architecture.</td>
<td>38</td>
</tr>
<tr>
<td>4.11</td>
<td>Images augmented with rotation.</td>
<td>40</td>
</tr>
<tr>
<td>4.12</td>
<td>Metrics reported during training VaDE for script phase extraction.</td>
<td>41</td>
</tr>
<tr>
<td>4.13</td>
<td>Scatter plot of image latent space means for script phase extraction.</td>
<td>41</td>
</tr>
<tr>
<td>4.14</td>
<td>GMM clustering confusion matrix for script phase extraction.</td>
<td>43</td>
</tr>
<tr>
<td>4.15</td>
<td>Metrics reported during training VaDE for corner script steps extraction.</td>
<td>43</td>
</tr>
<tr>
<td>4.16</td>
<td>Scatter plot of image latent space means for corner script step learning.</td>
<td>44</td>
</tr>
<tr>
<td>4.17</td>
<td>GMM clustering confusion matrix for corner script step extraction.</td>
<td>44</td>
</tr>
<tr>
<td>4.18</td>
<td>Metrics reported during training VaDE for T-intersection script steps extraction.</td>
<td>46</td>
</tr>
<tr>
<td>4.19</td>
<td>Scatter plot of image latent space means for T-intersection script step learning.</td>
<td>46</td>
</tr>
<tr>
<td>4.20</td>
<td>GMM clustering confusion matrix for T-intersection script step extraction.</td>
<td>47</td>
</tr>
<tr>
<td>4.21</td>
<td>Metrics reported during training VaDE for room script steps extraction.</td>
<td>49</td>
</tr>
<tr>
<td>4.22</td>
<td>Scatter plot of image latent space means for room script step learning.</td>
<td>49</td>
</tr>
<tr>
<td>4.23</td>
<td>GMM clustering confusion matrix for room script step extraction.</td>
<td>50</td>
</tr>
<tr>
<td>4.24</td>
<td>Learned Petri Net demonstrating structure and transitions of script phases.</td>
<td>52</td>
</tr>
<tr>
<td>4.25</td>
<td>Baseline architecture of CCN network.</td>
<td>54</td>
</tr>
<tr>
<td>4.26</td>
<td>Metrics reported during training baseline regression CCN.</td>
<td>54</td>
</tr>
<tr>
<td>4.27</td>
<td>Metrics reported during training baseline classification CCN.</td>
<td>55</td>
</tr>
<tr>
<td>4.28</td>
<td>Baseline architecture of CCN with phase and step IDs.</td>
<td>55</td>
</tr>
</tbody>
</table>
4.29 Metrics reported during training baseline regression CCN with phase and step IDs. .................................................. 56
4.30 Metrics reported during training baseline classification CCN with phase and step IDs. .................................................. 57
4.31 Architecture of baseline CCN with maps and team member relative locations. .................................................. 59
4.32 Metrics reported during training baseline regression CCN with maps and relative team member locations. .............................. 60
4.33 Metrics reported during training baseline classification CCN with maps and relative team member locations. .............................. 62
4.34 Confusion matrices of baseline classification CCN with maps and relative team member locations. .............................. 63
4.35 Architecture of baseline CCN with maps, team member relative locations, and phase and step IDs. .............................. 64
4.36 Metrics reported during training baseline regression CCN with maps, relative team member locations, and one-hot phase and step IDs. .............................. 65
4.37 Metrics reported during training baseline classification CCN with maps, relative team member locations, and one-hot phase and step IDs. .............................. 67
4.38 Confusion matrices of baseline classification CCN with maps, relative team member locations, and phase and step IDs. .............................. 68
4.39 Architecture of baseline CCN with maps, team member relative locations, and pretrained encoder from phase learning VaDE. .............................. 69
4.40 Metrics reported during training baseline regression CCN with maps, relative team member locations, and pretrained weights. .............................. 70
4.41 Metrics reported during training baseline classification CCN with maps, relative team member locations, and pretrained weights. .............................. 72
4.42 Confusion matrices of baseline classification CCN with maps, relative team member locations, and pretrained weights. .............................. 73

A.1 Script steps of team entering a room. ............................................. 88
A.2 Script steps of team traversing an intersection in their hallway. .......................................................... 89
A.3 Script steps of team taking an intersection in their hallway. .......................................................... 90
A.4 Script steps of team intersecting a hallway. ............................................. 91
A.5 Script steps of team navigating a corner ............................................. 93

B.1 Room variant of bottom left corner doorway on left-hand side. ............................................. 95
B.2 Room variant of bottom left corner doorway on right-hand side. ............................................. 95
B.3 Room variant of bottom right corner doorway on left-hand side. ............................................. 96
B.4 Room variant of bottom right corner doorway on right-hand side. ............................................. 96
B.5 Traversing hallway variant of intersecting hallway being on left-hand side. ............................................. 98
B.6 Taking intersection variant of intersecting hallway being on left-hand side. ............................................. 99
B.7 Intersecting hallway variant of team taking the right-hand hallway branch. ............................................. 100
B.8 Corner variant with corner that bends 90-degrees to the left. ............................................. 101
Chapter 1

Introduction

An essential collaborative human-robot team behavior is human-robot co-navigation, in which robotic agents cooperatively move with humans in a shared space. Human-robot co-navigation has been explored in many important applications, such as manufacturing [13, 88], firefighter search and rescue [59, 60], military operations, and urban disaster relief [15, 47, 48]. The skills needed and challenges faced in human-robot co-navigation vary significantly based on the relationship between the robot and the humans in the environment. In some applications, such as robot-crowd navigation, the robot and humans are completely independent in their goals and there are no spatial requirements to fulfill aside from non-collision. Because of this independence, in these applications, the robot does not need to have insights into the plans and goals of the humans around it, aside from an estimation of the humans’ trajectories. Other applications, such as firefighter search and rescue, place a robot on a team with humans and require tight collaborative navigation as a form of teamwork. This work terms this form of team collaborative co-navigation the Constrained Collective Movement (CCM) of a team. This research focuses on learning a model of a human team’s CCM in order to incorporate a robot into the team of humans. Equipping the robot with a model of the human team’s CCM will enable the robot to infer and emulate the humans’ behavior and conform to the human team’s CCM.

Substituting a robot for a human on a team performing applications such as firefighter search and rescue and military operations could have many life-saving effects. The robot could be equipped with a suite of sensors such as LiDAR and infrared vision that would help it navigate the team in visually impaired situations. The robot could also be used as a pack bot to carry heavy equipment for the team. Both of these means of support are already accomplished by human team members, but in situations of real risk and danger, alleviating physical and mental stress from the human team members, may save lives. Additionally, the robot could be sent into high danger areas first and potentially save a human from being harmed.

1.1 Constrained Collective Movement

This work defines the CCM of a team as how the members of the team maneuver in coordinated ways through complex environments while managing trade-offs due to
environmental and application-specific constraints. These constraints may involve general environmental concerns, such as obstacle avoidance, as well as unique application-specific challenges and commonly employed specialized tactics, referred to as Standard Operating Procedure (SOP), for solving them. These specialized tactics, such as for searching a burning building for people, have been developed by decades of research and require teams of humans to undergo long, rigorous training processes to achieve success. It is important to note that while two applications may be similar, the methods employed to tackle each one may be significantly different. For example, while a firefighter search and rescue and a military building clearing operation may both be a form of search task, they each have a very different team size, team structure, and search protocol.

This research capitalizes on the human-only team’s goal to follow an explicit SOP by enabling the robot to interpret and emulate the SOP. This will give the robot a high-level understanding of the intra-team (movement within the team “unit”) and inter-team (movement of the team as a whole through the environment) behavior. This will facilitate incorporating the robot on an expert human-only team in a socially compliant manner; a manner that mimics low-level human control, as well as leverages the team’s SOP to give the robot a high-level script-based understanding of the team’s complex behavior.

While the overall problem is very challenging, this research focuses on an initial beginning step to determine whether it is possible for robots to learn basic CCM in the context of navigation. This work is not learning the entire behavior demonstrated by a human-only team. Specifically, if the team is searching a building for injured people and administering aid to them, the robot is not learning to look for, recognize, and help people. This work is exploring the first step in this problem space: can we learn the SOP scripts and combine them with a motion controller to allow the robot to join the team? Furthermore, this work is explored in the context of a high fidelity simulation environment and is not conducted with real people and robots in the physical world. The high fidelity simulation environment gives a realistic environment to explore and test the developed ideas so that in future work they can be explored in the physical world with real humans.

1.2 Problem Statement and Challenges

1.2.1 Problem Statement

As stated in the section 1.1, the CCM of a team is defined as navigating a 3-D environment in a distributed fashion, such that each team member moves to follow inter-team and intra-team motion protocols. To satisfy inter-team protocols, each team member must move so the team progresses as a unit according to the method prescribed by the SOP script (e.g., breadth-first search vs. depth-first search). Intra-team protocols guide team members on how they must move so as to conform to formation structures mandated by the SOP script. This work formulates CCM as \( P = CCM(U, (T, A, C)) \), with:

**Inputs**

- a continuous 3-D environment \( U \). The state of the environment at time \( t \) is denoted as \( U_t \).
• the tuple \((T, A, C)\) that defines the team’s characteristics.
  
  - \(T = \{m_i : 1 \leq i \leq M\}\), a team of \(M\) team members.
  - \(A\) is a set of possible actions agents can take.

• a Script \(C = (H, S, R, F)\).
  
  - \(H\) is a set of script phases. Each phase represents a class for a script step (e.g., “Room Navigation” is a class for script step “Enter Room”). Each script phase \(h_t\) is dependent on the environment \(H(h_t|U_t)\).
  - \(S\) is a set of script steps. Each step, \(s_t\), is dependent on the environment, the step’s phase, and team member poses, \(S(s_t|U_t, h_t, P_t)\).
  - Each team member, \(m_i\) has a heterogeneous role \(R_{m_i}\), which indicates the appropriate action to be taken as a probability distribution over possible actions conditional on the script step, phase, team member poses, and the environment, \(R_{m_i}(a|s_t, h_t, P_t, U_t), a \in A\). \(R = \{R_{m_i} : m_i \in T\}\).
  - \(F\) is a set of transitions from one script step to another. Each transition is conditional on the current step, the current phase, the action selected, and the environment, \(F(s_{t+1}|s_t, h_t, a_t, U_t)\).

**Outputs**

• the set of team member pose sequences \(P = \{(p_{m_i})_{t=0}^N : m_i \in T\}\)
  
  \((p_{m_i})_{t=0}^N = ((x_{m_i}, y_{m_i}, \phi_{m_i})_0, ..., (x_{m_i}, y_{m_i}, \phi_{m_i})_t, ..., (x_{m_i}, y_{m_i}, \phi_{m_i})_N)\), where \((x_{m_i}, y_{m_i}, \phi_{m_i})_t\) is a tuple indicating agent \(m_i\)’s 2-D \(x, y\) position and yaw orientation, at time \(t\). \(P_t\) is the set of all agent poses at time \(t\).

The problem of CCM for a human-robot team is to infer the script model from demonstrations of CCM being performed by a human-only team, \(T_H\). That is, given \(U\), \(T_H\), \(A\), and \(P\) construct \(\hat{C}\). It is assumed \(T_H\) is a human-only team with homogeneous capabilities, a priori training with a well-defined script \(C\), and no communication.

### 1.2.2 Challenges

Based on the problem statement, below are specific challenges that should be addressed by a full solution to the problem;

• **Step Identification** - The robot should be able to identify which step the team is at in the script. When creating the script model this is important for determining the overall structure of the script. Determining the script step when performing CCM will allow the robot to infer where each member of the team, including itself, should move.

• **Novel Step Recognition** - If the robot is encountering a step in the script that it has not seen previously, then it should recognize it as being a novel step and update its script model accordingly. Learning the new script step will involve generalizing the relative team member poses, the general environment landscape, and the previous and next steps of the script.
• **Low-Level Navigation** - Once the robot understands the current script step, the agent should use this information for low-level instructions to determine a location goal and solution path.

• **Update Step Transitions** - Each non-start step in the script is preceded by at least one step and each non-end step has at least one subsequent step. The script model should be able to adapt and update its belief of the previous and next steps in the script.

• **Close Script Loops** - Some scripts have loops in the procedures: “while my car is being repaired, sit in the waiting room”. To have an accurate model of the script, it is necessary for the robot to understand and capture the high-level looping behavior of the script.

• **Role Adaptation** - Adapt to fewer team members or current team members “failing” in their roles.

This dissertation primarily addresses the challenges of step identification and low-level navigation. This research develops models to learn the script phases, script steps, and their transitions from demonstrations of human-only teams performing CCM. Additionally, this research develops models to determine the appropriate action the robot should take if they were put on the human-only team. This work addresses elements of novel step recognition, updating step transitions, and closing script loops in the context of learning the initial script model. Retroactively updating the learned script while performing CCM in unseen environments may be a valuable addition to the work in the future. Role adaptation is not performed in any way in this research but is a promising area of future work.

1.3 Overview of Approach

This work, along with previous work on human-only teaming [33, 54, 72, 75, 85], has been influenced by research in human-only teams. Specifically, this research incorporates insights that a shared mental model between team participants increases team performance [50, 78]. Developing a representation of the humans’ shared mental model for the robot takes advantage of the fact that many expert teams utilize SOPs. The main goal of this research is to learn a model of the team’s SOP and incorporate the information into the robot’s action selection that determines the control commands. The approach taken in this work for modeling an SOP is that of an AI script.

Robots interact with the world by viewing the environment through sensors and modifying the world with effectors and actuators. The pipeline from sensing to movement can be a complex pipeline incorporating feature extraction, sensor fusion, cognitive reasoning, decision making, and instruction interpretation. This work bypasses much of the specialized and complex components in this pipeline by training neural networks to classify the appropriate low-level action from raw sensory information.

Multiple methods of incorporating the high-level SOP information learned into the low-level control commands made by the robot are explored in this work. One method explored is using the features that determine each script step as an additional input to the end-to-end
robot controller. An alternative method that is developed is sharing the weights trained in the networks used to learn the SOP with the end-to-end robot controller. These methods will be explored in more detail in Chapter 3.

1.4 Contributions

This work has two main contributions to the fields of machine learning and robotics. The first contribution is that of constructing the SOP script from sensory information. Constructing scripts has largely been performed with corpora of text for natural language processing. My research utilizes visual and other spatial features to create scripts capturing real world events.

The other major contribution is that of merging the two cognitive paradigms that are developed in this work. A large portion of human-robot collaborative co-navigation focuses on formation control, which falls short of capturing more complex behaviors than simply maintaining a certain spatial relationship. The work to incorporate script knowledge with an end-to-end controller gives rich, high-level, goal-oriented information to the decisions being made to control the robot.

1.5 Dissertation Overview

This dissertation has the following structure: Chapter 2 provides a review of related works. Chapter 3 describes the approach developed in this work. Chapter 4 summarizes experimental results. Chapter 5 concludes with summary remarks.
Chapter 2

Literature Review

The research objective is to incorporate a robot on a team of expert humans performing a task relying on collaborative navigation, a challenge that amounts to enabling the robot to infer and cooperate with the team’s CCM. This objective is highly related to the bodies of research exploring human-robot co-navigation and human-robot teaming. Relevant works in these areas are addressed in Section 2.1. The developed approach uses expert team demonstrations to create high-level model-based scripts as well as low-level learning-based policies of the team’s behavior. These scripts and policies are merged to generate the appropriate robot movement strategies. This approach necessitates an understanding of several fields of study. Research in schema theory and model-based approaches, in general, are key to the approach and are discussed in Section 2.2. Results from deep behavioral cloning and other learning-based methods are also of importance to this research and are presented in Section 2.3.

2.1 Human-Robot Co-Navigation

Human-robot co-navigation is essential for a mobile robot to operate in a shared space with humans. It is a problem that incorporates elements of social norm understanding, static and dynamic obstacle avoidance, as well as goal completion. There exists a spectrum of dependence between the robots and humans performing co-navigation. The independent side of the spectrum would be a system where each agent’s goals are independent of all the other agents and the driving interaction is social compliance while avoiding collisions. A good example of a task on the independent side of the spectrum is crowd navigation [18, 23, 83, 86]. Around the middle of the spectrum would be systems where teams of humans and robots have the same goals, but coordinating movements is not integral to accomplishing the goal. An example of this type of system is cooking a meal or cleaning a room. The agents all have the same goal of preparing several dishes at the same time and navigation is mainly focused on avoiding being in each other’s way. The dependent side of the spectrum are tasks where teams work together on a common goal and navigation coordination is critical to successfully completing that goal. Teams performing a firefighter search and rescue mission are a prime example of the dependent end of the spectrum. The applications addressed in this work reside on the dependent side of the spectrum.
There has been substantial effort in recent years to solve tasks on, and related to, the independent side of the spectrum. Pedestrian trajectory prediction is a problem that has close ties to crowd navigation and has seen a lot of attention from the computer vision and autonomous driving communities. Most commonly, observational learning has been used to train systems of Long Short-Term Memory networks (LSTMs) to predict the trajectories of pedestrians [2, 11, 25]. There are, however, results from reinforcement learning that have showed success as well. One notable work used deep reinforcement learning to enable a robot to navigate around areas of a college campus crowded with pedestrians [18]. Yet another approach used is clustering trajectory behavior for use in path planning for autonomous cars [49]. These methods hold similarities to this work’s problem, insofar as they aim to enable a robot to move around an environment with people. As such, methods for solving these problems have a direct bearing on tackling the addressed problem. However, due to the dependence of the team members on each other, as well as the inherent heterogeneous role of each team member, the related solutions cannot be directly used to solve this research problem.

At the middle of the spectrum are applications where robots and humans are on teams, so their behavior is more dependent, but co-navigation is not an inherent trait of their teamwork. A notable work comes from Breazeal where she uses crowd-sourcing to train a model for a human and robot to complete a set of tasks in a shared space to escape a Mars setting [14]. A non-team application in the middle of the spectrum is that of herding people out of a dangerous situation such as a building fire [37].

At the other end of the spectrum are applications that rely heavily on the robot and human not only navigating in a cooperative fashion, but a coordinated and collaborative fashion. Classical methods for human-robot coordinated navigation have focused on formation control [36, 52, 69, 71, 74, 87]. However, these methods tend to rely on hard coding rules and typically do not express complex behaviors that shift according to the dynamic nature of real world environments. A model-based method is used in [3, 70] for a group of robots to aid a firefighter. Again, these methods focus on having the robots maintain a set formation around the firefighter, without demonstrating a rich behavior that varies depending on the situation facing the team. In [74], a desired formation is set and the robot notifies the human team member via a vibrating band if the human is destroying the formation and the robot cannot act to avoid it. Work is done in [79] to demonstrate how formation-keeping can aid a human and robot working together to move a heavy object. Riek explores the idea of synchronous joint action on human team members and how a robot can leverage this to gain information on the team’s dynamics and coordinate its movements with a human team as they dance [36].

A body of work that closely explores the problem in this dissertation comes from generating behaviors for military training simulations. These works [39, 40, 55, 80] seek to generate behavior for a team performing a military operation, like a bounding overwatch. These works solve a similar yet simpler problem. The environments are exceedingly simple simulations and the operations themselves are composed of only one or two behaviors that do not change throughout the operation. The behaviors themselves are composed from primitives and thus susceptible to the shortcomings of limiting robot action and unnatural movements. The teams in this research perform operations that are composed of many
distinct, complex behaviors in a high fidelity simulation environment. The methods used in these works will be discussed in more detail in Section 2.3.

Human-robot teaming has been used in real world disasters to aid in search and rescue. Robots were first used in real world urban search and rescue at The World Trade Center disaster [15] and have seen continued use in scenarios such as post-earthquake search and rescue [47, 48]. Current protocol for robots used in the field primarily rely on teleoperation, which increases the stress and workload of the first responders working with the robots.

2.2 Model-Based Approaches

Model-based methods for human-robot co-navigation usually involve analyzing proxemics and constructing functions to emulate the desired behavior. Force-based models, such as potential fields [44] and the social forces model [30], apply pushing and pulling forces to the agent’s velocity to guide the agent to their goal while avoiding collisions. Social potential fields are used in [69] to ensure a small set of robot agents stay in a set formation around a human. Potential fields are further used in [3, 70] to maintain formation control with a firefighter. Since its initial publication, the social forces model has been extended for use in crowd navigation [27], for a service robot accompanying a human [26], and to incorporate various factors such as social interactive groups [86].

Due to their hand-crafted design, model-based methods have the benefit of high transparency into why the robot decides which action to take. This hand-crafted design does have two main drawbacks, however. Firstly, they require substantial time and domain knowledge to analyze the desired relationship and create the functions the robot should follow. Secondly, these methods typically generate high-level movement primitives that may limit the actions a robot can take and may result in unnatural, jagged movements. The goal would be for the robot to adopt the human’s behaviors closely so as to minimize the retraining time it would take the humans to become comfortable with a robot being interchanged with a previous human team member. This work uses a model-based approach as a high-level team mental model and a learning-based model to generate the robot’s movements. A script is used to capture the robot’s understanding of the team’s current state and this information is used in conjunction with low-level behavioral cloning to direct the robot’s actions. A comparison of required retraining time is not performed in this dissertation and is saved for future work.

Schemata are a form of knowledge representation first defined in psychology and cognitive theory that define relationships between perception and actions. The idea of using schemata for robot navigation was perhaps most notably presented in Arkin’s seminal works [7, 8] and further explored for formation control in [10]. Scripts, presented by Schank [73], are a chain of schemata that represent events and situations that an agent is so familiar with that they do not need every detail to understand the context of the situation and the appropriate action to take. A classic example of such a situation is ordering at a restaurant. Most people have been in this situation enough times that they know what actions to take when a host or hostess asks them to follow and then stops by a table. Similarly, the average person is not surprised when a random new person walks to their table and asks them if they would like something to drink. As the task this work focuses on involves a team of (simulated) expert
humans who have trained thoroughly together, the developed approach can take advantage of the second nature of how the team acts together and form the mental model as a script for the robot. Many attempts at learning script knowledge representations have come from the natural language processing community. The goal of this community is to capture or infer meaning from a corpus of text utilizing scripts [62, 63, 64]. To the best of my knowledge, learning a script from visual and other sensory information for human-robot team knowledge representation has not been performed before.

2.3 Learning-Based Approaches

Learning-based approaches follow two main threads: reinforcement learning (RL) and supervised learning-based behavioral cloning (BC). While there has been success with using RL for crowd-based human-robot co-navigation [16, 18, 29, 45, 61] there is one difference in assumptions that makes BC a more attractive approach for this research. Both approaches try to learn a policy from state and action information gathered from demonstrations. BC assumes the demonstrations are performed by an expert policy and thus try to mimic the policy directly. RL assumes the demonstrations are performed with any policy and thus tries to balance maximizing a reward and learning the policy. This can result in the RL system improving on the observed policy which may generate behaviors that are not as natural for the humans on the team. Work has been done to train RL methods for autonomous driving [17], but given the expert nature of this work’s demonstrations, and the desire to mimic the behavior for the comfort of human team members, this work adopts a BC approach.

Various forms of supervised learning from different fields have inspired this work’s solution to the problem of CCM. Observational learning has been used in generating behavior for simulated forces for games and training programs [39, 40, 55, 80]. In [40], Kamrani et al. train a collection of decision trees to determine what actions the agent should take. While these works do seek to train a team to follow specific behaviors similar to this work, their learning of the behavior is based solely on high-level action primitives. These primitives have the disadvantage that they still need hard coding to get from the primitives to the low-level velocities. This may create a system that limits the actions a robot can take while producing jagged or unnatural robot movements. This sub-optimal robot action may result in lower human confidence in the robot and higher stress and overhead load on the human. The goal of this research is for the system to directly learn these low-level velocities. As discussed in Section 2.1, observational learning has been used with promising results. LSTMs [32], a form of Recurrent Neural Network (RNN), have seen particular success given their ability to capture temporal information. Merged LSTMs are trained in [2] to predict trajectories of pedestrians given their neighbor’s trajectories. Bartoli et al. [11] build on [2] by not only incorporating other agents into the trajectory prediction, but also static objects of interest in the environment. Additionally, results from autonomous car driving have influenced this work. Both problems involve deriving controls for a vehicle from rich high-dimensional perceptions of the world. One work of particular interest is [4], wherein Amini et al. use a variational feed-forward network to learn a distribution over the appropriate turn and acceleration controls for the car given images of the scene.
Other networks that inform this research are Convolutional Neural Network (CNN), which extract useful features from visual images [46, 51, 82]. CNNs have been used in co-navigation for determining where pedestrians may be moving to [23]. Autoencoders [68], which have been around for decades as a way to extract a condensed feature vector [31, 89], have been used in creative ways in more recent work, such as paraphrase detection in texts [76] and pairing autoencoders to translate between robot actions and natural language descriptors [92].
Chapter 3

Approach

The general approach in this research is to use demonstrations of a human-only team performing examples of CCM to enable autonomous learning of a model of the team’s SOP. The learned SOP model is then provided to a robot, enabling it to mimic behaviors similar to those demonstrated by the human-only team, which I believe would allow the robot to work together with the humans in a natural, transparent manner.

When considering the procedural flow of the approach, it is helpful to think of the approach as having two distinct stages: a learning stage and an execution stage. The learning stage, as seen in Figure 3.1, takes demonstrations of a human-only team performing CCM and constructs a model of the team’s SOP. To accomplish this, a human-only team performs CCM while executing a task, leveraging their knowledge gained from extensive prior training. While the human-only team is working, sensors outfitted on the humans collect features about the environment the team is in and the actions taken by the team members. The environment features are used as inputs to train the SOP model while the team member actions are used as the ground truth for learning output values that describe the correct behavior.

It is important to note that this research does not use real world human-only teams, but instead mimics natural human motions in a realistic simulation environment. For ease of presentation, this dissertation refers to humans but in actuality, simulated humans are used in the experiments.

The execution stage, depicted in Figure 3.2, uses the learned SOP model to incorporate a robot into a human-only team. The robot will adopt the role of one of the humans from the human-only team used in the training stage, forming a new human-robot team. When this team performs an operation similar to the task used in training demonstrations, the robot will use onboard sensors to collect and process environment features. These environment features will be used as inputs to the components of the SOP model. The SOP model will predict what action the robot should take next that will result in the robot mimicking the learned human behavior.

The learned SOP model is designed to capture the high-level and low-level behaviors of the team. High-level behavioral information, such as what general structure the team should be in for different scenarios, is helpful for the robot to make complex decisions about where it should be within the team as the environment changes. Learning low-level behavioral information that describes what action is being taken by an agent, describing their linear
and angular velocity for example, lets the robot mimic the movement patterns of the humans. This means the robot can move in a manner already natural to the rest of the team members, who have trained extensively with each other performing similar movements.

To capture high-level and low-level behavior information, the SOP model has two components, the script representation and the Command Control Network (CCN). The two components can be seen as part of Figure 3.1 and Figure 3.2. The script representation uses unsupervised learning models that do not directly control the robot. As such, during training, only the environment features are used to train the script model. The CCN is a supervised learning model that is trained with the purpose of directly controlling the robot. The CCN is trained with the environment features and the learned script information as inputs to the model. The team member action responses are used during training as well, as the ground truth for the learned output of the model.

The first item that must be addressed when learning the SOP model from the (simulated) physical world is processing the sensor readings of the environment \( U \), and generating the features required for learning. This step is discussed in detail in Section 3.1. The specifics of training and utilizing the knowledge gained from the script representation are explored in Section 3.2. The details of building and training the CCN are explored in Section 3.3. There are multiple ways in which high-level script information can be incorporated into the CCN. Section 3.4 explores three such methods developed in this research. A discussion and summary remarks are made in Section 3.5.

The application domain of relevance is the navigation of agents through an indoor urban environment that has hallways, rooms, T-intersecting hallways, and corners in hallways. The approach presented in this dissertation is specific to this domain and would not directly map to learning other types of scripts such as the restaurant script. However, I believe that this research is relevant to other domains as the number, structure, and inputs of the models used in the approach can be adjusted to reflect and capture the important domain features.

### 3.1 Preprocessing and Feature Engineering

To extract a high-level understanding of the team’s SOP through the environment, features pertaining to the general structure of the environment and the team members’ positions in the environment are needed. While the individual methods used in this section are well researched and not a novel contribution of this work, their combination and the goal of using them to extract visual and spatial features for script learning is.

The features described in this section are used in learning each of the script parameters \((H, S, R, F)\). Firstly, the general structure of the environment must be known so that the script representation can capture the high-level phases, \(H\). This research accomplishes this by mapping the environment as the team is performing an experiment. At each time step, a patch of the map surrounding the team can be used as a high-level “map image” of what the current environment looks like. These map images are ideal for capturing high level information such as if the team is in a room, while also capturing details specific to the current room, such as if the door to the room opens to a corner of the room or the middle of a wall.
Figure 3.1: Overview of learning phase. Demonstrations of human-only teams performing CCM are carried out. While the (simulated) human-only team is working, sensors collect features about the environment the team is in and the actions taken by the team members. The environment features and team member actions are used to train the SOP model. The script model is an unsupervised learning model that does not directly control the robot. As such, only the environment features are used to train the script model. The CCN is a supervised learning model that is trained with the purpose of directly controlling the robot. The CCN is trained with environment features and learned script information as inputs to the model. The team member responses are used as the ground truth for the learned output of the model.
**Figure 3.2:** Overview of execution phase. Once the two components of the SOP model have been trained, they are used to control the robot while it performs CCM with the human-robot team. The human-robot team co-navigates around the environment to perform a similar task as those used in the human-only team demonstrations for the learning phase. The robot uses onboard sensors to collect environment features. Those features are passed to the script model. The script phase and step information output from the script model is used as an input, along with the environment features, to the CCN. The action command output from the CCN is then passed to the robot for execution.
While the general structure of the environment is enough to determine what phase of the script the team is in, learning the specific steps, $S$, used to navigate the script phase requires more detailed information. These mid-level script steps depend greatly on the relative location of team members as well as specific nuances of the current instance of the phase being explored. For example, when a team enters a room that has a door opening in the middle of a wall, the team members should have a different formation than if the door enters the room at a corner of the room. As such, a way for encoding the team member locations as inputs to the model in a visually descriptive method is needed. This work takes advantage of the structurally descriptive quality of the map images for step learning. At each time step, the current map image is augmented by superimposing each team member’s location on the image. These map images now contain high level information about the structure of the environment and the formation of the team within the environment and are ideal for learning the script steps.

### 3.2 Learning SOP Script Structure Representation

This section describes the approach taken to represent and learn the structure of the SOP script. Specifically, this work uses an Artificial Intelligence script to model the SOP. A script is a structure that describes appropriate sequences of events in a particular context. Scripts are used to represent specific knowledge in order to interpret and participate in events that agents have been through many times. As such, they are ideal for capturing the SOP information. As defined in Section 1.2, a script is parameterized by a set of script phases, a set of script steps, a role for each team member, and a set of transitions from one script step to another. The role for each team member is modeled with the CCN and is discussed in detail in Section 3.3. The set of script phases, script steps, and transitions between script steps all describe the structure of the script being learned. The script phases capture high-level information about a specific area of interest the team may be encountering, such as a room or a hallway. Script steps detail mid-level information about the general steps the team should perform to navigate a script phase. Examples of script steps are a team entering or leaving a room. Script phases and steps have a natural temporal flow where one script phase and step can lead to any number of other script phases and steps. These transitions between script phases and steps are vital for the robot to make inferences about what future situations current actions can lead to.

In other work, statistical scripts have been learned for the purposes of natural language processing [62, 63, 64]. However, these methods only consider words as features and do not extend to applications that deal with a physical space. Constructing scripts from real world sequences of common events using visual and spatial features is a novel contribution of this work. This contribution has implications not just in Human-Robot Interactions with expert teams, but also in areas that don’t require teaming such as manufacturing, construction, and Virtual Reality (VR) personnel training. For example, robots can use the “script” or general assembly process being used in manufacturing and construction to help out where most needed without acting on a team and without being explicitly told what to do. This same technology can be used without a robot in a VR setting to give humans multiple intelligent artificial agents to practice with. This type of VR augmentation would benefit
programs such as the New York Police Department’s usage of VR to train their police officers to respond to active shooter scenarios [42].

The creation of the script phase and script step elements is presented in Section 3.2.1. After all of the script elements are learned, they may be used along with training sequences to determine the script step and phase transitions. This final step in SOP script creation is explored in Section 3.2.2.

### 3.2.1 Script Element Extraction

After high-level visual features have been extracted from raw sensor data, they are used to train a Variational Auto Encoder (VAE) for script phase clustering. Instead of learning a compressed state of the input as is done with standard autoencoders, VAEs learn the means and standard deviations of a multi-variable probability distribution that explains where the data came from. As such, VAEs have had much success in capturing and expressing visual information for clustering [20, 43, 53, 65]. VAEs are unsupervised, generative networks; both of these qualities are beneficial for applications handled in this work. Unsupervised learning techniques support this work’s goal of limiting hard coding components of the learned models. Using a VAE for script phase and script step clustering means the model will find the natural distributions that distinguish environment and team member features. Additionally, using a generative model means we can have the network generate representative images from the clusters of phases and steps. For example, if the team is just entering a corner, the network could be used to generate a representative image of the next step in the corner phase SOP. Two possible future works could benefit from these images. Firstly, these images could be used to supply more visual information to the CCN. Supplying snapshots “into the future” may prove advantageous for the robot controller. Secondly, these images could be valuable for direct communication with the human team members. The robot would be able to tell the humans not only what phase and step the robot believes the team is on, but could also show the human team members where the robot believes each team member should move to.

Advancements have been made to VAEs to improve their ability to cluster images [24, 91, 93]. This research specifically uses a VAE augmented with a Gaussian Mixture Model, known as a Variational Deep Embedding (VaDE) [38], to extract the multimodal probability distribution of visual features over the phases of the script. During training, instead of learning the parameters of a single multi-variable probability distribution, VaDEs learn the parameters of the multiple components specified by the GMM. The GMM parameters are trained in tandem with the autoencoder’s encoder and decoder layers.

This work trains a VaDE for clustering the script phases. Additionally, a separate VaDE is trained to cluster the script steps for each script phase. During the execution phase, when the robot has joined the human expert team, the environment features undergo two VaDE passes. Firstly, the features are passed to the script phase VaDE to determine the current script phase. The features are then passed to the VaDE which was trained to determine the steps for the appropriate phase. This results in the script phase and script step identification for the current time step.
3.2.2 Transition Extraction

After learning the sets of individual script phases and script steps, the transitions between the phases and steps must be learned. This research uses Petri nets [58] to model this sequence of transitions between phases and steps. Petri nets are graphical models used to describe distributed systems. They are designed to represent multiple entities and requirements in graphical form. This is well suited for tasks that involve multiple agents and a dynamic environment.

Formally, Petri nets have three parameters: a set of places, a set of transitions, and a set of directed arcs or “flow relationships.” Petri nets are bipartite graphs with the two types of elements being places and transitions and the connections between nodes are flow relationships. Places are represented by circles and can contain any number of tokens. Transitions are represented by rectangles. A transition activates whenever each place that leads to the transition has at least one token in it. When a transition activates, one token is removed from each place that has a flow relationship to the transition, and one token is added to each place that has a flow relationship from the transition.

As an illustration, Figure 3.3 constructs a simple “dining at restaurant” script as a Petri net. Each place, represented as a circle, designates an individual event or environmental condition. Notice in Figure 3.3a, before the patron can sit down at the table, certain environmental conditions must be met. Namely, the Maitre d’ must be present and a table must be available. Dining at a restaurant is a procedure the typical person has been through enough times to know that if you enter a restaurant and there is a podium at the door with no one behind it, you typically do not just walk in and sit down at a random table. Instead, you wait for someone to come and ask how many people are in your party. Figure 3.3b demonstrates that while multiple constraints of one transition may be met, it will not activate until all constraints are met. In this example, after going through the steps of being seated and having initial courses, if the patron still wants dessert, they will place an order with the waiter, instead of asking for the check. It is important to note that each place in a Petri net, each step in the script, has an appropriate action depending on environment and application-specific constraints. For example, one of the phases in the restaurant script is the “eat food phase”, which consists of the patron going through the steps to order a course and consume the food. Each time the patron wants to perform this phase, they must go through a similar, but slightly different, procedure depending on the restaurant and what course they are having. Details such as which menu to use may change, but the general procedure remains the same.

In this work, each Petri net place represents features extracted from the environment and may have a script phase and step associated with them. Transitions represent the conditions needed for the team’s natural movement from one script phase and step pair to another phase and step pair. Constructing the Petri net is performed iteratively over all sequences of team poses in the training data set to build out the structure of the SOP script. Per the problem statement, each sequence of team poses starts and ends with a well defined script phase and step. Each time step in each team position sequence is used to infer the current script phase and step and update the transitions to previous script phases and steps. The script building method, designed for use in this research, is outlined in Algorithm 1.
(a) Script step with patron entering and waiting to speak to the Maitre d’ about getting a table. Patron has to wait for environment constraints to follow script steps to be seated.

(b) Script step with patron ordering more food from menu. Patron is ordering more food instead of paying the bill and leaving due to application specific constraints.

**Figure 3.3:** Simple restaurant script structured as Petri net.
Algorithm 1: Algorithm for building Petri Net.

**Data:** FeatureEngineering, Clusters, TrainingEpisodes

**Result:** Transitions, Places, Arcs

Transitions ← Set{TransitionID};
Places ← Dictionary{PlaceID : (PhaseID, StepID)};
Arcs ← Dictionary{TransitionID : PlaceID};
CurrentPlaceID ← 0;
CurrentTransitionID ← 0;

for PoseSequence ∈ TrainingEpisodes do
  Step PreviousStep ← StartStep;
  Step PreviousPhase ← StartPhase;
  for Sample in PoseSequence do
    Feature f ← FeatureEngineering( Sample.sensorInformation );
    (Phase p, Step s) ← Cluster( features, Clusters );
    if (p,s) \notin Places then
      Transitions.Add(CurrentTransitionID);
      Arcs.Add(CurrentPlaceID, CurrentTransitionID);
      CurrentPlaceID ← CurrentPlaceID + 1;
      Places.Add(CurrentPlaceID : (p, s));
      Arcs.Add(CurrentTransitionID, CurrentPlaceID);
      CurrentTransitionID ← CurrentTransitionID + 1;
    end
  end
end

3.3 Learn Control Commands

This research develops the Command Control Network (CCN) as an end-to-end controller to determine the appropriate commands to move the robot. An end-to-end controller is a neural network that takes in raw sensory data, and possibly higher-level features, and directly calculates movement commands. These types of controllers have seen great success in applications such as robot navigation [34, 66] and autonomous car driving [4, 9]. The general architecture of the CCN is illustrated in Figure 3.4. More specifically, the CCN is a feed forward network that takes images of the environment and produces linear and angular robot control values. Because of the widespread success of CNNs in image understanding, CNNs are incorporated into the early stages of the network to extract high-level features from the environment images. These features are then passed to fully connected layers which lead to the output of the network. This architecture is inspired by literature which shows successes in similar applications [90, 94].
3.4 Incorporate SOP Script with Control Commands

There are three methods developed to merge the high-level script information with the low-level controller.

The first method for inputting script information into the CCN would be using a unique ID associated with each script phase and step. This method is visualized in Figure 3.5. At each time step, the current script phase and step are predicted with the relevant VaDEs. The unique phase ID and step ID are encoded in the network by converting the IDs into one-hot vectors, concatenating the one-hot vectors to the flattened CNN layers, and passing all of the information through the fully connected layers. A weakness of this method is that a phase and step IDs do not inherently give a sense of relationship between this phase and step and another phase and step. Other methods may have more information on how this is a unique step and what the role of the agent should be.

The second and more intuitive approach uses the high-level images generated with the SOP script feature engineering module and uses them as input to the network along with the images of the environment. This network is detailed in Figure 3.6. The work in [4] does something similar to give driving directions to an autonomous car. The images generated for learning the current script step are used as an input to the network and will go through a similar set of CNN layers as the original visual images.

In addition to their usage in image clustering, VAEs have also seen significant success in end-to-end controllers of autonomous vehicles [4, 9, 12]. These works demonstrate that capturing the distributions of the latent space of visual features and using those as elements of the end-to-end controller have positive effects on the network’s performance. The works listed do not address the dependent relationship with humans that is a focus of this work. This type of dependence adds a challenge of addressing the nondeterministic nature of human actions. However, VAE’s success with extracting meaningful features from road environments (environments that demonstrate similarities yet also contain many significant variations), lends credibility to the networks ability to capture the nondeterministic nature of human actions in a meaningful way. This work recognizes that incorporating VAEs into an end-to-end controller has the benefit of learning a general representation of the environment that is robust to variations.

As end-to-end autonomous vehicle driving has similarities to this work, the third method for CCN augmentation is incorporating the variational component from the trained phase learning VaDE. Specifically, the pretrained encoder, variational layers, and GMM from the phase learning VaDE as the beginning layers to the CCN. The weights in these layers would not be modified in training. The network would have fully connected layers after the variational layers that would be trained to generate the appropriate movement controls. This would be a way to implicitly incorporate information captured from script learning into the end-to-end controller.

3.5 Summary and Discussion

The developed approach utilizes demonstrations from a team of (simulated) human experts performing CCM to train the parameters of a script representing the team’s SOP. Team
Figure 3.4: Architecture of network used to train end-to-end robot controller. Environment images are passed through CNN layers to extract spatial features. The output of the CNN layers is flattened. This output is passed through multiple fully connected layers. The final fully connected layer connects to two outputs to learn linear and angular values to control the robot.

Figure 3.5: Architecture of network with categorical script information merged. Script and phase information are used to construct one-hot vectors. These vectors are concatenated with the output of the CNN layers.

Figure 3.6: Architecture of network with visual script information merged. The image used to learn the current script phase and script steps is included as an input. The image is passed through a similar set of CNN layers and is concatenated with the other visual inputs.
pose sequences of the team performing CCM in different environments are used as the
demonstrations to train the script. Once the script model is trained, it can be used to enable
the robot to perform on the team with the humans. The use of script learning for visual and
spatial information, especially for expert human-robot teams, is a novel contribution of this
work. The unpredictability and interdependence on human team members adds a level of
complexity to classic robot action decision making problems, such as loading a dishwasher,
that makes the problem considerably more challenging.

The learning stage uses environment information and the human team members’ actions
to train the various components of the SOP model. Sensor information is used to construct
high-level images of the environment. These high-level environment images are used to train
multiple VaDEs. One VaDE network is used to cluster the visual features into script phases.
Another VaDE network is trained to cluster the script steps for each script phase. The
approach of extracting visual features and clustering them with VaDEs for script learning
is a novel contribution of this work. These methods for extraction and clustering have
been used in other domains, but not for an application that depends so greatly on human
interaction. The clustered script phases and steps are used to create a Petri net that captures
the transitions between phases and states. The visual features as well as the human team
members’ actions are used to train an and-to-end network that incorporates the visual
features in a number of examined ways. This network, combining the script information
and behavioral cloning, is another significant contribution of this work.

After the learning stage, the execution stage uses the learned models to perform CCM on
a human-robot team. At every time step of the team performing an exploration application,
the map images from the robot’s sensors are first passed through the feature extraction
module to get the high-level visual features. These visual features are then passed to the
trained VaDEs to determine the current script phase and script step. The current script
phase and script step are used with the Petri net to determine the next script phase and
script step. This information is passed, along with the original visual features, to the end-
to-end controller to determine what action the robot should take.

**Temporal Information**

The temporal element of a team performing CCM through an environment is an important
characteristic for both developing the SOP structure and the CCN. Currently, temporal
information is captured in the system by building out the Petri network, according to
algorithm 1. When creating the CCNs currently only single images of the current time
step are used in training and testing the networks. This was a simplifying design decision
influenced by architectures of end-to-end driving systems in related works [4, 9, 12]. However,
including a temporal element explicitly in the CCN is an interesting avenue of future work.
This information could be included by simply inputting images of the environment from
multiple time steps into the current CCN. A more advanced method would be altering the
type of network used for the CCN. Specifically, making the base network some for of recurrent
neural network may capture important temporal information. An interesting avenue of future
work is analysing how many time steps back result in the best performance.
Model Explainability

The key contribution of this work is the creation of a script model for use with behavioral cloning to aid in an expert human-robot team’s CCM during the desired application. The addition of the SOP script model to the CCN would demonstrate improved performance and increased transparency. Performance of the robot participating on the team is improved due to the robot having a more accurate high-level understanding of what the team is trying to do and how the robot should participate in the operation. Having a high-level model of the robot’s understanding of the team behavior means the robot could more transparently answer questions regarding its decision making. This map would help human team members interact with the robot, and could result in the humans having more trust in the robot’s selected actions.

This idea of increased transparency is one of the most interesting facets of future work that may stem from this research. One of the simplifying assumptions made in this work is that the members of the team do not explicitly communicate with each other. An interesting research question to explore would be: if communication were allowed between team members, would incorporating the learned script information improve the robot’s communication with the humans? This dissertation hypothesizes that including script information would greatly increase the humans’ abilities to quickly and accurately gain information from the messages sent by the robot. A common downside to end-to-end controllers is their “black box” nature. These systems are not transparent about what decisions go into making specific command controls. As such, communication depending on these systems is limited. The robot can tell the humans what it believes the appropriate action to take is, but it cannot give significant information about how and why it came to that decision. This lack of transparency into the robot’s actions may cause the humans to have reduced levels of trust that the robot is making the right decision. By incorporating high-level script information into the decision process, the robot is also able to communicate this high level information to the humans at each time step. Instead of only being able to communicate to the human “I believe I should move forward and to the right” the robot can send high-level motivation as well such as “The team is about to enter a room so I should move forward and to the right.” In addition to potentially improving understandability in communication, this work hypothesizes that increasing transparency when communicating robot actions will also increase human trust that the robot is doing the right thing for the right reasons. Illustrating this capability would be an interesting area of future research.
Chapter 4

Experiments and Results

This chapter presents the setup and results of experiments designed to test the developed approach’s ability to learn a SOP script model of a predefined team’s behavior when performing CCM, as well as use the learned model for incorporating a robot onto the team.

Experiments are performed in a high fidelity, realistic simulation environment. Within the simulation environment, a team is constructed and behavior guides are programmed into the team to generate a complex SOP for the team to follow while navigating the environment and performing an abbreviated military building clearing application. The predefined team performs CCM with the defined behaviors in two simulation environments. Data collected in one environment is used for training the models, while data from the other environment is only used to test the methods in a new environment. The training data is used to learn the SOP script models of the predefined simulation team’s CCM. The data generated from the testing environment is then used to determine how well the models have captured the predefined team’s CCM. Testing data is passed to the models and comparisons are made between the decisions made by the models and the predefined team.

Details about the building clearing application and behaviors are presented in Section 4.1. The specifics of the two simulation environments, and defining the team in the simulation, are discussed in Section 4.2. The specific details of the characteristics and quantities of the training and testing data gathered from the two environments are detailed in Section 4.3. The models constructed and results attained for learning the SOP script representation are discussed in Section 4.4. This section includes results from training the models for script phase and script step learning, learning the transitions between script phases and script steps, and learning the CCN for controlling the robot’s actions. Finally, a summary and final discussion of results is performed in Section 4.5.

4.1 Behaviors

There are three criteria for a suitable set of behaviors in the experiments to demonstrate the effectiveness of the system. Firstly, selected team behaviors should be well-defined. This will improve the ability to determine when things are working or not. Secondly, the behaviors should be commonly known, so as to facilitate understanding. Lastly, the behaviors should be complex in order to adequately test the solution’s abilities. To satisfy these criteria,
military building exploration operations are chosen as the behavior of the teams. These operations have been developed over decades of military research so they are well-defined and well-known procedures. Each operation is also sufficiently complex to handle real world situations faced by military forces.

While the performed experiments demonstrate the developed approach’s ability to handle one set of rigorously defined behaviors, the results should generalize well to other team building exploration/search applications such as firefighter search and rescue. Additionally, this work may generalize well to other applications that depend on structured spatial relationships and behavior norms between humans and robots — applications such as elderly and physically impaired assistance.

Building exploration or “building clearing” uses a multi-person team with the goal of navigating through an environment with unique protocols for moving down a hallway and navigating through distinct areas of interest, such as exploring a room connected to the hallway. This research’s implementation of building exploration supplies behaviors for navigating three areas of interest: navigating around a corner in the hallway, moving in and through an intersecting hallway, and clearing a room. This work uses a four-person team in all of its experiments. The team behaviors were inspired by a United States Army infantryman’s guide to combat from 1993 [21]. When performing a building clearing application, one of the implied assumptions is that unknown areas have increased risk and should be explored with caution. In the case of military building clearing, the risk manifests itself in the form of opponents potentially looking to do the team members harm. The SOP mitigates this risk by establishing behaviors that promote caution and maneuverability when exploring unknown areas. However, this work does not try to explicitly learn concepts of harm or learn a form of robot caution. Instead, this work constrains itself to learning the physical relationships demonstrated by a predefined team.

For the purposes of the learning phase, each environment navigation trace (sequence of team member poses) starts and ends at specified hallway locations. The team navigates down the hallway following the SOP demonstrated in Figure 4.1. The team follows the hallway and performs the appropriate behavior for each area of interest as it comes to it. Notice that for a given experiment, the team may run into any number of instances, or none, of each area of interest. Whenever the team encounters an intersection, instructions are given to the team on what intersection branch to take in order to get to the specified end location. When the team encounters a room, they will perform the behavior demonstrated in Figure 4.2. The behavior is described in detail in Appendix A.

Different environment situations result in variations to the general room SOP specified in Figure 4.2. There are two main environment variables: where the doorway is located in the room and which side of the hallway the doorway is on. The doorway can be in the center of a wall, as in Figure 4.2, or it can be in one of two corners of the room, as in Figures B.1a and B.2a of Appendix B. This type of environment variable changes the entire structure of the team’s positions in the room. Additionally, the doorway can be on the left or right side of the hallway that the team is approaching from. This does not affect the main structure of the team’s final position in the room, but it may affect which team member takes on which roles (e.g., which team member has to perform a switchback). These variants are explored in detail in Appendix B.
Figure 4.1: Visual of hallway behavior.
The team may approach and navigate a T-intersection in one of three ways. The team may traverse an intersecting hallway, as shown in Figure 4.3, the team may take an intersecting hallway, as shown in Figure 4.4, or the team may intersect another hallway, as shown in Figure 4.5. The detailed steps of each type of T-intersection navigation are discussed in Appendix A.

T-intersection traversal and taking a T-intersection each have a variant that is the result of the intersection being on the team’s left-hand side instead of their right-hand side. The variants have symmetrical goal positions, but the variants may require different team members to adopt each goal position. Additionally, intersecting a T-intersection has one variant that is the result of the team needing to go right instead of left. This variant is symmetrical as well, but again, the team member required at each goal position is potentially different.

When navigating a corner, the team performs the SOP depicted in Figure 4.6. A detailed discussion of the steps taken for corner navigation are presented in Appendix A. Corner navigation has one variant that is the result of the hallway taking a 90° turn to the team’s left instead of right. This variant is presented in Appendix B.

4.2 Environments

The gaming industry has made significant advances to game development engines that result in near photo-realistic environments with accurately simulated physics that can incorporate simulated humans and robots in urban settings. Additionally, significant advances have been made in game development engines that give the developer the ability to create complex behaviors for the agents in the environment. These advances have sparked a field of study known as serious games [1, 22, 67, 81] in which researchers use video game systems to explore serious research questions. Serious games have been used in many interesting applications relating to computer vision and robotics in recent research [5, 6, 41, 56]. Due to these advances, this research uses Unity, a game development engine, to design and run simulations. Using Unity has two main advantages for this work. Firstly, performing simulations in realistic environments means less retraining would need to be performed when transitioning the learned model from simulation to the real world. Unity’s realism is not present in its visualizations alone, but also its physics engine. Agents and objects in Unity are given physical properties such as friction and gravity which increases the realistic, nondeterministic nature of the Agents’ interactions with the world. Secondly, Unity allows for more complex behaviors to be programmed into the agents performing the experiments. Unity has built-in AI capabilities that handle path planning and can be used to direct the agents to goal positions throughout the environment. These behaviors are influenced by the physics engine as well and thus have built-in deviations. These built-in deviations in experiment traces are a desirable quality as they emulate the nondeterministic behavior of humans. A team of humans could attempt to perform the same exact trace in the same environment repeatedly and there will always be slight variations in the movements that they make. Having this type of variability manifest in the simulation environment gives this research more realistic training and testing data. Additionally, this variability also gives the models more unique
Figure 4.2: Script steps of team entering a room. Steps specific to a room with a door in the center of its wall and the team entering with the doorway on its left-hand side.

Step 1: Queue up outside of room. Step 2: Lead team member (yellow) enters doorway.
Step 3: Lead team member (yellow) continues in forward motion to most distant position.
Step 4: Next team member (blue) performs switchback, continues to most distant position.
Step 5: Next team member (red) continues in forward motion to position close to doorway.
Step 6: Next team member (green) performs switchback, continues to position close to doorway.
Step 7: All team members leave room and go to construct hallway formation.
Figure 4.3: Script steps of team traversing an intersection in their hallway. Steps specific to a hallway intersecting the team’s hallway on the team’s right-hand side. Step 1: Position on brink of intersection. Step 2: Forward (yellow) and rightmost (red) team members position to see down intersecting hallway. Step 3: Left (blue) and rearmost (green) team members move forward into intersection. Step 4: All team members pass through intersection, forming hallway formation on the other side.
Figure 4.4: Script steps of team taking an intersection in their hallway. Steps specific to a hallway intersecting the team’s hallway on the team’s right-hand side. Step 1: Position on brink of intersection. Step 2: Forward (yellow) and rightmost (red) team members position to see down intersecting hallway. Step 3: Left (blue) and rearmost (green) team members move forward into intersection. Step 4: All team members enter intersecting hallway, forming hallway formation on the other side.
Figure 4.5: Script steps of team intersecting a hallway. Steps specific to intersecting a hallway and progressing to the left. Step 1: Lead team member (yellow) positions themselves on the brink of the left path of the intersected hallway. Rightmost team member (red) positions themselves on the brink of the right path of the intersected hallway. Step 2: yellow and red team members enter hallway, staying close to the wall, and facing opposite directions. Step 3: Leftmost (blue) and rearmost (green) team members enter the hallway, facing opposite directions, and support yellow and red team members. Step 4: All team members leave intersection, forming hallway SOP and continue on their way.
Figure 4.6: Script steps of team navigating a corner. Steps specific to a corner that bends at a 90-degree angle to the team’s right-hand side. Step 1: Position so forward (yellow) and rightmost (red) team members can view hallway around corner. Step 2: All team members move into corner. Step 3: All team members move into hallway, forming hallway formation.
data which will allow the models to generalize and avoid overfitting, or memorizing, examples it has seen.

This research uses two separate simulation environments for collecting training and testing data. The training simulation environment is visualized in Figure 4.7. The training simulation environment is developed specifically to incorporate multiple areas of interest that require the different behaviors, specified in Section 4.1, to successfully navigate through them. As indicated in Figure 4.8a, the areas of interest are three rooms (each with a different door placement), a T-intersection, and a corner. Experiments performed start at one origin location and end at a different origin location. A typical experiment may start at “Origin 1” and end at “Origin 3.” For this experiment, the team would need to navigate the hallway to the room with the center entrance, then navigate through the room with the right entrance, then the T-intersection, taking the intersecting hallway to the corner, navigating through the corner, and ending at “Origin 3”. This research will refer to a team moving through the environment from one origin to another as a trace. The result of a team’s trace through the environment is a trajectory, which is the sequence of positions the team took to navigate the environment.

The simulation environment developed for collecting testing data is shown in Figure 4.8. The testing simulation environment is developed as a simpler environment than the training environment, but one that still incorporates one of each area of interest. As shown in Figure 4.8, the testing simulation environment has one T-intersection, one corner, and one room. There are two origin locations and in each trace, the team will start at one origin and end at the other. An example of one such trace will have the team starting on “Origin 1” and ending on “Origin 2.” In this example, the team will proceed down the hallway to the T-intersection, they will navigate through the intersection, branching into the intersecting hallway. The team will then move to and through the corner and then proceed to navigate through the room and finish at “Origin 2.”

The team is composed of four team members. The images showcasing the desired team behaviors from Section 4.1 are used to hard code goal points in the environment for each step of a behavior, for each team member. The behavior sequences are encoded as a behavior tree, a structure used for dictating the flow of behavior in game development [19, 35, 84, 95]. The behavior tree uses the location of the agents along with hard-coded locations in the environment to determine which behavior the team should currently be demonstrating. A Unity built-in obstacle avoiding navigator is used to move the team members to their specified goal positions.

As an example, consider if the team is moving down the hallway from “Origin 1” toward “Origin 2”. As soon as the lead team member’s position is within a certain threshold of a hard-coded position, indicating the team is approaching the room with center entrance, the behavior tree switches from issuing goal locations that enforce the hallway behavior and set goal positions, as shown in Figure 4.2, for the team to move through the steps to navigate the appropriate room. This process of setting up goal locations for each team member, for each step, of each behavior is incredibly tedious and time-consuming. It is done here for the purposes of mimicking what human team members would do and to provide realistic examples for the proposed system to learn from. However, note that the learning modules have no access to these predefined behaviors. This is a significant motivation for
the developed approach with its automated, data-driven process that would not require this
type of hard coding, given data from actual teams of interest.

4.3 Data Collected

The models trained in this work take as inputs RGB images, local patches of the map
surrounding the team, and team member relative positions. Additionally, the CCN models
produce linear and angular offsets that specify where the robot should move to. As such, the
experiments performed rely on sensors and methods that can gather all of this information;
sensors such as a camera for RGB images, LiDAR to collect laser scans for generating maps,
computer vision methods of human pose recognition, and IMU and other tracking devices
for determining distances and angles traveled.

As described in Section 4.2, experiments are performed in Unity. Unity has built-in RGB
video capture capabilities which are used in the experiments to generate the RGB images of
the environment. A LiDAR device is simulated in Unity by using transparent rays that are
cast from a source in 1° increments for 360° around the source location. The rays report the
distance from the source to the first encountered object. By default, Unity tracks the global
location of objects in the simulation environment. Relative team member locations can be
calculated by taking the difference of global locations between team members. Similarly, the
angular and linear distance traveled by an agent between time steps can be calculated by
taking the difference in global location of that agent at each time step. These linear and
angular difference values are used as ground truth values for training the CCN networks in
Section 4.4.4.

When the team navigates in the environment during each trace, sensor information
capturing environment and team member pose data are collected for building the SOP
models. Without loss of generality, in this work all data collected and models generated
are from the perspective of the rightmost (red) team member as shown in Figure 4.1. To
accomplish this, in the simulation environment, the rightmost agent is equipped with a
forward facing RGB camera and a simulated LiDAR as discussed previously, which serves
as input to the learning system. Additionally, all relative team member locations are taken
with respect to the rightmost agent.

Raw sensor data is transmitted to ROS [77] which is used to collect, process, and store
the data generated from each experiment in Unity. Map images for training the SOP models
are generated using the laser scans and team member relative pose data. The laser scans are
passed to a ROS package called “gmapping”, which is an implementation of the FastSLAM
2.0 [57] algorithm. Gmapping produces an accurate map based on the laser scans with
information about the localization of the laser producing the scans. At each time step, a
patch of the area around the team is selected from the map. Each patch is centered on
the team’s centroid position, the average location of the team members. In addition to
generating the standard map images as seen in Figure 4.9a, map images are augmented with
the team member locations overlaid onto them, as is shown in Figure 4.9b. All map and
RGB images are scaled and normalized to have pixel values between 0 and 1.

This method of image generation has the advantage of multiple laser scans being used to
generate each image. The images are, therefore, well-defined lines of the environment. This
Figure 4.7: Unity training simulation environment. The environment is designed to collect data of the team performing each behavior defined in this work. As such, the environment has three rooms, one T-intersection, and one corner.
(a) Unity testing environment overview.

(b) First person view of hallway in testing environment.

(c) First person view of room example in testing environment.

Figure 4.8: Unity testing simulation environment. The environment is designed to collect data of the team performing each behavior defined in this work. As such, the environment has one room, one T-intersection, and one corner.
method has a disadvantage that artifacts from previously explored regions may be captured in a patch and may confuse the image. An example of this type of artifact can be seen in Figure 4.9a. At this time, the team has just finished exploring a room and is back in the hallway, but artifacts from the room and the recently explored T-intersection can be seen in this map image as well.

When a trace is performed, hand-specified script phase and step labels are recorded as the team moves through the environment. The script phase and script step labels are generated by comparing the the overlaid map images to the images and descriptions of the constructed team behaviors from Section 4.1. The phase labels are 0 for Hallway, 1 for Room, 2 for T-intersection, and 3 for Corner. There is one script step in the set of Hallway data, six script steps in set of room data, five script steps in the set of T-intersection data, and four script steps in the set of corner data. Four traces were performed with the training simulation environment for each of the following start and end sets: Origin 1 to Origin 2, Origin 1 to Origin 3, Origin 2 to Origin 1, Origin 2 to Origin 3, Origin 3 to Origin 1, and Origin 3 to Origin 2. Between each trace, objects in the environment (furniture, pillars, etc.) are moved around significantly to produce discernible variations in the environment and subsequently captured images and team member positions. This results in 24 traces for a total of 2,891 data samples. Four traces were performed with the testing simulation environment for each of the following start and end sets: Origin 1 to Origin 2, and Origin 2 to Origin 1. As in the training data collection, between each trace, objects in the environment (furniture, pillars, etc.) are moved around significantly to produce discernible variations in the environment and subsequently captured images and team member positions. This results in 8 traces for a total of 808 data samples.

4.4 Script Representation Learning Results

The collected dataset is used to train four VaDE networks. One network is used to learn the phases of the script. Three VaDE networks are used to train the script steps associated with the room, T-intersection, and corner phases separately. Each VaDE has the architecture displayed in Figure 4.10. Each VaDE’s GMM is given a different number of components. The number of GMM components is equal to the number of items that specific VaDE is trying to learn. For example, the room script step learning VaDE has a GMM with six components. The network’s architectures are adopted from an architecture successful at capturing information on the benchmark MNIST [28]. The trained VaDEs are used to extract the latent space distribution parameters for each image. The mean parameters of the images are clustered according to their learned GMM component. It is important to note that the VaDEs being trained are done so in an unsupervised manner. The script phase and script step information is not being used during training and is only being used to report on the unsupervised network’s ability to cluster the map images in a way that is in line with the predefined team’s SOP.

Before training each VaDE, several augmentations may be performed to the data. The data may be re-balanced so that each class of script phase or script step has an equal number of images. This reduces the likelihood of the model being biased by the data being comprised of more samples of one script phase or script step over another less represented script phase.
Figure 4.9: Original map images and map images augmented by overlaying team member positions.

Figure 4.10: Script phase VaDE architecture. Input map image is passed through two CNN layers of size 64 and 32 kernels. After each CNN layer, the output is passed to a max pooling layer which is not shown for image clarity. The resultant layer is flattened into a fully connected layer. The output of the fully connected layer is passed to two separate, fully connected layers of size 2. These layers are responsible for capturing the mean and standard deviation of the latent space distributions. The output from these layers are passed to $z$, which is the layer responsible for extracting samples from the latent space distributions given the learned GMM clusters. Latent space samples are passed to a fully connected layer of the same size as the flattened layer. The fully connected layer is passed through two deconvolutional layers of size 32 and 64. After each deconvolutional layer, the layer is upsampled to adjust for the max pooling. The output of the final deconvolutional layer is the reconstruction of the original image.
or step. The data may be duplicated and rotated by increments of two degrees from four
degrees counter-clockwise to four degrees clockwise. Examples of rotated images can be seen
in Figure 4.11. This is performed to increase the number of images used to train the network
and thereby reduce overfitting.

Section 4.4.1 details the results of learning the phases of the SOP script. Section 4.4.2
reports on the results from learning the VaDEs for determining the script steps for each phase.
The results derived from learning the transitions between script phases and steps is discussed
in Section 4.4.3.

4.4.1 Phase Extraction

The best results for script phase learning were obtained using overlaid map images of size
75x75 pixels and augmented with balancing script phases and with rotation. Balancing and
rotating the map images resulted in the network being trained with 29,800 images (7,450
images of each script phase) and tested with 5,020 (1,255 images of each script phase). The
VaDE’s GMM is given four components, one for each script phase. The VaDE was trained
for 200 epochs with 25% of the data being held for validation. The model can classify the
script phase with an accuracy of 79% on the training set, 77% on the validation set, and
74% on the test set.

Figure 4.12 shows the accuracy and reconstruction loss being reported while the VaDE is
training. The closeness of the training and validation curves gives a good indication that the
model is not overfitting the data. Additionally, the accuracy and reconstruction loss level
out towards the end showing the model has likely finished learning.

A visualization of the learned latent space can be attained by passing each image through
the VaDE encoder and plotting the resulting latent space distribution means. This is
illustrated in Figure 4.13. Each point in the scatter plot is color-coded based on its label:
0 for Hallway, 1 for Room, 2 for T-intersection, and 3 for Corner. The latent space in the
VaDE is given two nodes and is therefore 2-dimensional. The scatter plot uses the two means
generated for each image that describe the probability distribution they are mapped to in
the latent space. Figure 4.13 shows that despite the unsupervised nature of the model, it can
clearly capture information valuable to the relationship between map images and assigned
phases. For instance, it can be seen in Figure 4.13 that the majority of points for each phase
are grouped in similar regions of the latent space. Corners are largely found clustered on
the right side of the image, rooms are clustered mostly on the bottom of the image, corners
are predominantly found at the top of the image, and hallways are clustered in the center of
the image. This makes intuitive sense as hallways not only share similar features with other
phases, but also have the largest number of samples transferring from one phase to another.
For example, when a team leaves the hallway and enters a room, those transition images
should appear in the latent space in between the main cluster of hallway samples and room
samples.

Figure 4.14 shows a confusion matrix created by clustering each data set based on the
trained GMM. From the confusion matrices, we can see that model performs very well
on recognizing rooms and T-intersections. The model does reasonably well recognizing
corners, but not quite as well as on the rooms and T-intersections. The model is weakest
on recognizing hallway images although the accuracy of almost 70% on the validation set is
Figure 4.11: Example of images augmented by rotating them four degrees counter-clockwise to four degrees clockwise in two degree increments. Only the original image, the four degree counter-clockwise rotated image, and the four degree clockwise rotated images are shown.
Figure 4.12: Accuracies and losses reported during training VaDE for script phase extraction. The maximum accuracy reached is 79% on the training set and 77% on the validation set. The network reaches a stable learning level and no signs of overfitting are indicated.

Figure 4.13: Scatter plot of image latent space means for script phase extraction. Each point in the scatter plot is color-coded based on its label: 0 for Hallway, 1 for Room, 2 for T-intersection, 3 for Corner. The key takeaway is that the VaDE does a good job clustering the script phases. Most corner samples (yellow) are clustered to the left of the figure, most T-intersection samples (green) are clustered to the top of the figure, most room samples (blue) are clustered to the bottom of the figure, and most hallway samples (purple) are clustered to the center of the figure.
still significantly higher than random selection. This reduction in hallway accuracy is likely
due to the number of samples of the team transitioning from the hallway to another area of
interest.

These results demonstrate that, given overlaid map images of the team performing CCM,
the unsupervised VaDE model can effectively learn to distinguish the script phases present
in the environment. The model was able to maintain a high accuracy when being tested on
an environment it had no prior exposure to.

4.4.2 Step Extraction

Corner Phase Step Extraction

The best results for corner script step learning were obtained using overlaid map images
of size 75x75 pixels and augmented with balancing corner script steps and with rotation.
Balancing and rotating the map images resulted in the network being trained with 1,640
images (410 images of each script step) and tested with 1,160 (290 images of each script
phase). The VaDE’s GMM is given four components, one for each corner script step. The
VaDE was trained for 200 epochs with 25% of the data being held for validation. The
model can classify the script phase with an accuracy of 88% on the training set, 87% on the
validation set, and 83% on the test set.

Figure 4.15 shows the accuracy and reconstruction loss being reported while the VaDE
is training. As can be seen, the VaDE manages to learn a high accuracy without overfitting
the corner script step data.

A visualization of the learned latent space is illustrated in Figure 4.16. Each point in the
scatter plot is color-coded based on its script step label. The latent space does show a clear
progression in clustering of script step samples. It appears as though the step zero samples
start on the right side of the figure and smoothly transition in a clockwise manner around
the outside of the figure progressing through steps two, three, and four. It makes logical
sense to see this type of progression, as well as seeing the step four and step zero clusters
being close to each other. Step four is associated with leaving the corner which is similar in
features to samples of step zero where the team is entering a corner.

Figure 4.17 shows a confusion matrix created by clustering each data set based on the
trained GMM. From the confusion matrices, we can see that model performs very well on
clustering the script steps of the corner phase. In all data sets, the step with the lowest
correctly classified samples is the third step. This step is involved with the team rotating
around the bend of the corner and as such, may have more confusion involved in the overlaid
map images than the other script step samples.

These results demonstrate that the unsupervised VaDE model can very effectively learn
to distinguish the corner script steps present in the environment, given overlaid map images
of the team performing CCM. The results for learning corner script steps is better than the
results of learning the other phase script steps. This may be due to the lower number of steps
involved in the corner phase. This reduction in performance could also be due to the fewer
number of variants and less drastic changes in environments from one variant to another.
The model was able to maintain a very high accuracy when being tested on an environment
it had no prior exposure to.
Figure 4.14: GMM clustering confusion matrix for script phase extraction. The main takeaway is that the reported individual phase accuracies are high. The lowest accuracy is for hallway phase samples. This is likely due to the large number of samples that transition from hallways to other areas of interest.

Figure 4.15: Accuracies and losses reported during training VaDE for corner script step learning. The maximum accuracy reached is 88% on the training set and 87% on the validation set.
Figure 4.16: Scatter plot of image latent space means for corner script step learning. The main point to notice is the script steps samples are clustered well together. Additionally, the script step samples have a natural flow from one script step to another.

Figure 4.17: GMM clustering confusion matrix for corner script step learning. The key takeaway is that the network has very good accuracy across most script steps. There is one weakness with clustering the third script’s step. This step is involved with the team rotating around the bend of the corner and as such, may have more confusion involved in the overlaid map images than the other script step samples.
T-Intersection Phase Step Extraction

The best results for T-intersection script step learning were obtained using overlaid map images of size 75x75 pixels and augmented with balancing T-intersection script steps and with rotation. Balancing and rotating the map images resulted in the network being trained with 3,175 images (635 images of each script step) and tested with 1,140 (228 images of each script phase). The VaDE’s GMM is given five components, one for each T-intersection script step. The VaDE was trained for 200 epochs with 25% of the data being held for validation. The model can classify the script phase with an accuracy of 86% on the training set, 83% on the validation set, and 79% on the test set.

Figure 4.18 shows the accuracy and reconstruction loss being reported while the VaDE is training. As can be seen, the VaDE manages to learn a high accuracy without showing signs of overfitting the T-intersection script step data.

A visualization of the learned latent space is illustrated in Figure 4.19. Each point in the scatter plot is color-coded based on its script step label. The latent space does show a clear progression in clusters of script step samples. It appears as though the step zero samples start on the left side of the figure and smoothly transition in a counterclockwise rotation around the outside of the figure progressing through steps two and thee. Step four continues on the counterclockwise progression, but step three interestingly branches off into the middle of the figure. This could be due to the nature of the script steps progressing through the intersection and samples being different depending on if the team is taking the branch into the intersecting hallway or not. It could also be due to the other variation of the phase where the team approaches and intersects a different hallway.

Figure 4.20 shows a confusion matrix created by clustering each data set based on the trained GMM. From the confusion matrices, we can see that model performs, as a whole, very well on clustering the script steps of the T-intersection phase. The only script step that is classified poorly is the third script step. This is the step associated with positioning fully in the hallway. The weakness with this step could be attributed to the variants of the T-intersection phase having dramatic differences.

These results demonstrate that, given overlaid map images of the team performing CCM, the unsupervised VaDE model can accurately learn to distinguish the T-intersection script steps present in the environment. Despite the nature of T-intersection variants to be very different from each other, the model does very well on clustering the majority of script steps. The model was able to maintain a high accuracy when being tested on an environment it had no prior exposure to.

Room Phase Step Extraction

The best results for room script step learning were obtained using overlaid map images of size 75x75 pixels and augmented with balancing room script steps and with rotation. Balancing and rotating the map images resulted in the network being trained with 17,340 images (2,890 images of each script step) and tested with 2,940 (490 images of each script phase). The VaDE’s GMM is given six components, one for each room script step. The VaDE was trained for 200 epochs with 25% of the data being held for validation. The model can classify the
Figure 4.18: Accuracies and losses reported during training VaDE for T-intersection script step learning. The maximum accuracy attained is 86% on the training set and 83% on the validation set.

Figure 4.19: Scatter plot of image latent space means for T-intersection script step learning. The main point to notice is the script step samples are clustered well together. Additionally, the script step samples have a natural flow from one script step to another.
Figure 4.20: GMM clustering confusion matrix for T-intersection script step learning. The key takeaway is that the CCN has good accuracy with one weak spot that is likely due to a drastic difference in the team starting in the intersected hallway or the intersecting hallway.
script phase with an accuracy of 54% on the training set, 51% on the validation set, and 48% on the test set.

Figure 4.21 shows the accuracy and reconstruction loss being reported while the VaDE is training. As in the training of the phases, the closeness of the training and validation curves gives a good indication that the model is not overfitting the data. Additionally, the accuracy and reconstruction loss level out towards the end showing the model has likely finished learning.

A visualization of the learned latent space is illustrated in Figure 4.22. Each point in the scatter plot is color-coded based on its script step label. The latent space does show a natural progression of sample clusters that start with step zero (on the left of the figure) and smoothly transition through clusters of samples for steps one, two, three, and four (through the middle and lower portion of the figure). The most difficult step to cluster appears to be the last step when the team is leaving the room. These samples are scattered relatively evenly across the latent space with a possible heavier distribution on the left side. This may be due to the nature of the team leaving the room being a little more chaotic than how they enter a room.

Figure 4.23 shows a confusion matrix created by clustering each data set based on the trained GMM. From the confusion matrices, we can see that model performs adequately on recognizing most of the steps. As to be expected based on Figure 4.22, the one step that caused a low accuracy is the last step.

These results demonstrate that, given overlaid map images of the team performing CCM, the unsupervised VaDE model can effectively learn to distinguish the room script steps present in the environment. The results for learning room scripts steps is poorer than the results of learning the other phase script steps. This may be due to the higher number of steps involved in the room phase. This reduction in performance could also be due to the less structure involved in the later steps involving leaving the room and reentering the hallway. The model was able to maintain a reasonable accuracy when being tested on an environment it had no prior exposure to.

### 4.4.3 Transition Structuring

Each sample of each trace in the simulation environment has its phase classified based on passing the sample through the phase learning VaDE and using the GMM component as the phase label. Each sample is then passed through the VaDE associated with its predicted phase label to classify its script step. The samples are taken in sequence order and their predicted script phases and steps are used by Algorithm 1 to build the Petri Net structure that captures the transitions from one script phase and step to another. To adjust for inaccuracies in classifying script phases and steps, pruning of the Petri net is performed on connections between script steps that appeared less than ten times. Figure 4.24 depicts the resulting Petri Net. For simplicity purposes, only the script phases displayed.

The general structure of the Petri net is similar to the flow of the SOP defined in Figure 4.1. There are no major disruptions to the Petri net such as a script step in the room phase leading directly to a script step in the corner phase. This work attributes the similar flow of the Petri net in part to the smooth, continuous transforming of map images from one phase and step into another. For example, if a map image should be clustered as a hallway but is
Figure 4.21: Accuracies and losses reported during training VaDE for room script step learning. The maximum accuracy attained is 54% on the training set and 51% on the validation set. This is the lowest accuracy attained for script step clustering. It is helpful to note however, that the room phase has the largest number of steps and randomly guessing script step classifications would result in an accuracy of 17%. Additionally, the lower accuracy is likely due to the unstructured nature of the team exiting a room.

Figure 4.22: Scatter plot of image latent space means for room script step learning. The key takeaway is that script steps zero through four have reasonably well-defined clusters with script step five being almost evenly distributed among all other step clusters. This supports the intuition that the poor accuracy is largely due to struggling to capture the behavior of the team leaving the room.
Figure 4.23: GMM clustering confusion matrix for room script step learning. The main takeaway is that the network has reasonable accuracy across most script steps except the fifth script step. Again, this is likely due to the network struggling to capture the behavior of the team leaving the room.
instead clustered as a room, it is most likely that the sample will be clustered as a step one sample of the room phase. Even though the label associated with them may not match the ground truth associated with them, the relationship this sample has to the flow of the SOP is equivalent. This adds significant robustness to the Petri net’s ability to supply helpful information to the CCN.

4.4.4 Command Control Network

The collected dataset is used to train five CCNs for the purpose of exploring the alternative methods for incorporating SOP scripts with CCNs as described in Section 3.4. All networks have the same baseline structure as shown in Figure 4.25. There are four methods of incorporating script information that are used as augmentations to this baseline method. This section discusses the architecture and performance of each CCN trained.

In each trained model, the output of the network has two components. One component determines how much the robot should move linearly, and the other component determines how much the robot should move angularly. Details on collecting linear and angular values are discussed in Section 4.3.

For analysis purposes, regression and classification are performed for each of the five CCNs trained. For regression, each network is trained to directly learn the appropriate angular and linear offset of the next time step. The output of regression is a single node for each component. For classification, the continuous angular and linear values are discretized via binning. Bounds for the bins are selected according to the training data in order to ensure an equal number of samples per bin. For classification, the output for each component is a vector of five values indicating the predicted confidences for the discrete bin the sample belongs to. In each case, the model is trained with the 2,891 samples from the training environment with 25% of the samples being held out of training for validation.

Some augmentations to the CCN involve incorporating the script phase and step ID. To attain that information, each CCN input sample first has its overlaid image passed to the phase learning VaDE to get the predicted phase. The overlaid image is then passed to the VaDE associated with the predicted phase to get the predicted step. The predicted script phase and script step are passed to the CCN as one-hot vectors.

Baseline CCN

The architecture of the baseline CCN can be seen in Figure 4.25. Input RGB images are passed through two CNN layers of size 64 and 32 kernels. After each CNN layer, the output is passed to a max pooling layer. The resultant layer is flattened into a fully connected layer. The output of the fully connected layer is passed to one fully connected layer of 100 nodes and then to another fully connected layer of 50 nodes. The output of this fully connected layer is passed to the two separate outputs that represent the linear and angular offset the robot should travel.

As is illustrated in Figure 4.26, during regression, the network does not appear to learn anything meaningful from the training data. This is not surprising as the behaviors and environment are very complicated and this baseline is not supplied with enough information or complexity to learn anything. The minimum angular MAE attained is 21.97 on the
Figure 4.24: Learned Petri Net demonstrating structure and transitions of script phases. Note the labels have been added by hand as descriptives for the reader to understand what the Petri net is showing.
training set, 22.13 on the validation set, and 24.21 on the testing set. This equates to an average deviation of 24° between the appropriate angular offset and the predicted angular offset on the test set. The minimum linear MAE attained is 0.844 on the training set, 0.827 on the validation set, and 0.873 on the testing set. This equates to an average deviation of 0.873 feet (≈10.5 inches) between the appropriate linear offset and the predicted linear offset on the test set.

As can be seen in Figure 4.27, during classification, the network failed to learn anything meaningful from the training data. As with regression, this is not surprising as the behaviors and environment are very complicated and this baseline is not supplied with enough information or complexity to learn anything. The final accuracy for both the linear and angular outputs was 20% which does not beat randomly guessing. Confusion matrices showed the network guessing one linear and angular value for every sample and are not displayed for simplicity. Similarly, performance on the testing environment is not performed since the network did not learn anything.

**CCN with Script Phase and Script Step IDs**

The first modification to the baseline CCN is the approach of incorporating script phase and script step IDs as inputs to the network. The architecture of this augmented baseline CCN can be seen in Figure 4.28. Input RGB images are passed through two CNN layers of size 64 and 32 kernels. After each CNN layer, the output is passed to a max pooling layer. The resultant layer is flattened into a fully connected layer. The output of the fully connected layer is concatenated with the one-hot vectors encoding the script phase and script step. This concatenated vector is passed to one fully connected layer of 100 nodes and then another fully connected layer of 50 nodes. The output of this fully connected layer is passed to the two separate outputs that represent the linear and angular offset the robot should travel.

As is illustrated in Figure 4.29, during regression, the network does not appear to learn anything meaningful from the training data. Again, this is not surprising as the behaviors and environment are very complicated and this form of incorporating script information does not supply the network with rich details on the relationships between the team members and the environment. The minimum angular MAE attained is 19.85 on the training set, 19.75 on the validation set, and 23.17 on the testing set. This equates to an average deviation of 23° between the appropriate angular offset and the predicted angular offset on the test set. The minimum linear MAE attained is 0.824 on the training set, 0.835 on the validation set, and 0.862 on the testing set. This equates to an average deviation of 0.862 feet (≈10.3 inches) between the appropriate linear offset and the predicted linear offset.

As can be seen in Figure 4.30, during classification, the network failed to learn anything meaningful from the training data, just as in the case of the baseline. As with regression, this is not surprising as the behaviors and environment are very complicated and this form of incorporating script information does not supply the network with rich details on the relationships between the team members and the environment. The final accuracy for both the linear and angular outputs was 20% which does not beat randomly guessing. Confusion matrices showed the network guessing one linear and angular value for every sample and are not displayed for simplicity. Similarly, performance on testing environment is not performed since the network did not learn anything.
Figure 4.25: Baseline architecture of CCN network. Input RGB image is passed through two CNN layers of size 64 and 32 kernels. After each CNN layer, the output is passed to a max pooling layer, which is not shown for image clarity. The resultant layer is flattened into a fully connected layer. The output of the fully connected layer is passed to one fully connected layer of 100 nodes and then another fully connected layer of 50 nodes. The output of this fully connected layer is passed to two separate outputs that represent the linear and angular offset the robot should travel.

(a) Angular MAE.  
(b) Angular MSE.  
(c) Linear MAE.  
(d) Linear MSE.  

Figure 4.26: Metrics reported during training baseline regression CCN. The key takeaway is that the network failed to learn anything meaningful.
Figure 4.27: Metrics reported during training baseline classification CCN. The key takeaway is that the network failed to learn anything meaningful.

Figure 4.28: Architecture of baseline CCN with phase and step IDs. Input RGB image is passed through two CNN layers of size 64 and 32 kernels. After each CNN layer, the output is passed to a max pooling layer, which is not shown for image clarity. The resultant layer is flattened into a fully connected layer. The flattened layer is concatenated with the vectors of one-hot encoded script phase and script step IDs. The output of the concatenated layer is passed to one fully connected layer of 100 nodes and then another fully connected layer of 50 nodes. The output of this fully connected layer is passed to two separate outputs that represent the linear and angular offset the robot should travel.
Figure 4.29: Metrics reported during training baseline regression CCN with phase and step IDs. The key takeaway is that the network failed to learn anything meaningful.
Figure 4.30: Metrics reported during training baseline classification CCN with phase and step IDs. The key takeaway is that the network failed to learn anything meaningful.
CCN with Map Images

The next method of augmenting the baseline CCN is including information that has shown success in training the script phase and step learning VaDEs. Specifically, the overlaid map images are included as inputs to the CCN. Additionally, the team member locations relative to the rightmost agent are formed into a vector and are included as an input to the network. The structure of the network is shown in Figure 4.31. Input RGB images are passed through two CNN layers of size 64 and 32 kernels. Overlaid map images are also passed through two CNN layers of size 64 and 32 kernels. After each CNN layer, the output of that layer is passed to a max pooling layer. The resultant layer from each chain of CNN and max pooling layers is flattened. The flattened CNN layers are concatenated along with the vector of team member relative positions. The output of the concatenated layer is passed to one fully connected layer of 100 nodes and then another fully connected layer of 50 nodes. The output of this fully connected layer is passed to the two separate outputs that represent the linear and angular offset the robot should travel.

As is illustrated in Figure 4.32, during regression, the network was able to learn a model of the appropriate linear and angular offsets. The minimum angular MAE attained is 10.77 on the training set, 11.45 on the validation set, and 13.11 on the testing set. This equates to an average deviation of 13° between the appropriate angular offset and the predicted angular offset on the test set. The minimum linear MAE attained is 0.503 on the training set, 0.503 on the validation set, and 0.542 on the testing set. This equates to an average deviation of 0.542 feet (≈6.5 inches) between the appropriate linear offset and the predicted linear offset on the test set.

As can be seen in Figure 4.33, as with regression, classification saw significant improvement in learning. The network attained an angular accuracy of 81% on the training data, 73% on the validation data, and 68% on the testing data. The network attained a linear accuracy of 68% on the training data, 58% on the validation data and 56% on the testing data. This is indicative of the value supplied by high-level spatial information present in the overlaid maps.
Figure 4.31: Architecture of baseline CCN with maps and team member relative locations. Input RGB images are passed through two CNN layers of size 64 and 32 kernels. Overlaid map images are also passed through two CNN layers of size 64 and 32 kernels. After each CNN layer, the output of that layer is passed to a max pooling layer, which is not shown for image clarity. The resultant layer from each chain of CNN and max pooling layers is flattened. The flattened CNN layers are concatenated along with the vector of team member relative positions. The output of the concatenated layer is passed to one fully connected layer of 100 nodes and then another fully connected layer of 50 nodes. The output of this fully connected layer is passed to two separate outputs that represent the linear and angular offset the robot should travel.
Figure 4.32: Metrics reported during training baseline regression CCN with maps and relative team member locations. The key takeaway is the networks ability to learn to mimic the demonstrated movements with an average angular difference of $13^\circ$ and an average linear difference of 6.5 inches on the testing set.
The confusion matrices generated by the trained model, as shown in Figure 4.34, give valuable insights into where the network’s strengths and weaknesses are. The angular confusion matrices show that the network has the best accuracy on the label associated with no angular movement. This is likely due to the similarity of the environments where this occurs. For example, whenever the team is moving down a hall, there is very little angular movement performed by the team members. The network seems to have the most difficulty knowing when to select a small angle clockwise or counter clockwise. This is understandable as they involve more fine motor control situations. For example, they would be the commands needed to position the team member to look around a corner. These scenes are likely more varied than those of the team walking down the hallway. The linear confusion tells a little different story. Based on the linear confusion matrices the network has a harder time understanding when the robot should move forward larger amounts of distance. This is likely due to situations where the agent moves a lot after having been standing still. This scenario happens frequently when the team is leaving a room.

**CCN with Map Images and Script Phase and Script Step IDs**

This augmentation method attempts to determine if script phase and script step IDs can be beneficial to training, given more suitable companion data. As such, the baseline CCN is modified to have as inputs the overlaid map images, team member relative locations, and script phase and script step IDs. The structure of the network is shown in Figure 4.35. Input RGB images are passed through two CNN layers of size 64 and 32 kernels. Overlaid map images are also passed through two CNN layers of size 64 and 32 kernels. After each CNN layer, the output of that layer is passed to a max pooling layer. The resultant layer from each chain of CNN and max pooling layers is flattened. The flattened CNN layers are concatenated along with the vector of team member relative positions and the vectors of one-hot encoded script phase and script step IDs. The output of the concatenated layer is passed to one fully connected layer of 100 nodes and then another fully connected layer of 50 nodes. The output of this fully connected layer is passed to the two separate outputs that represent the linear and angular offset the robot should travel.

As is illustrated in Figure 4.36, during regression, the network again is able to learn a model of the appropriate angular and linear offset. The minimum angular MAE attained is 8.66 on the training set, 10.10 on the validation set, and 12.41 on the testing set. This equates to an average deviation of 12° between the appropriate angular offset and the predicted angular offset on the test set. The minimum linear MAE attained is 0.474 on the training set, 0.471 on the validation set, and 0.501 on the testing set. This equates to an average deviation of 0.5 feet (≈6 inches) between the appropriate linear offset and the predicted linear offset on the test set. The network attained similar performances to the network with map features and no phase and step IDs. The exact performance improvement is 1° on the angular component and 6 inches on the linear component. This indicates that the phase and step IDs are not necessarily crucial to learning, but do have a slight advantage when being used, especially with the linear component.

As can be seen in Figure 4.37, during classification, there is a small but definable improvement in learning when compared with the baseline classification CCN augmented with overlaid maps and team member relative locations. The network attained an angular
Figure 4.33: Metrics reported during training baseline classification CCN with maps and relative team member locations. The maximum angular accuracy attained is 81% on the training set and 73% on the validation set. The maximum linear accuracy attained is 68% on the training set and 58% on the validation set. The main takeaway is that this is a massive increase in learning from the baseline method.
Figure 4.34: Confusion matrices of baseline classification CCN with maps and relative team member locations. The main takeaway is that the angular accuracies are very good with slight weaknesses in classifying fine angular controls. Additionally, the network does very well on classifying small linear movements but has a harder time classifying larger linear movements.
Figure 4.35: Architecture of baseline CCN with maps, team member relative locations, and phase and step IDs. Input RGB images are passed through two CNN layers of size 64 and 32 kernels. Overlaid map images are also passed through two CNN layers of size 64 and 32 kernels. After each CNN layer, the output of that layer is passed to a max pooling layer, which is not shown for image clarity. The resultant layer from each chain of CNN and max pooling layers is flattened. The flattened CNN layers are concatenated along with the vector of team member relative positions and the vectors of one-hot encoded script phase and script step IDs. The output of the concatenated layer is passed to one fully connected layer of 100 nodes and then another fully connected layer of 50 nodes. The output of this fully connected layer is passed to two separate outputs that represent the linear and angular offset the robot should travel.
Figure 4.36: Metrics reported during training baseline regression CCN with maps, relative team member locations, and one-hot phase and step IDs. The key takeaway is the networks ability to learn to mimic the demonstrated movements with an average angular difference of $12^\circ$ and an average linear difference of 6 inches on the testing set.
accuracy of 83% on the training data, 80% on the validation data, and 73% on the testing data. The network attained a linear accuracy of 74% on the training data, 70% on the validation data and 68% on the testing data. This indicates that while the script phase and step IDs are not a powerful enough feature to facilitate learning on their own, they do have the ability to help the network make more accurate robot decisions.

The confusion matrices generated by the trained model are shown in Figure 4.38. The angular confusion matrices show that the network has good accuracy across all angular movements. The linear confusion matrices show that this network appears to have the same strength and weaknesses as the baseline CCN augmented with overlaid maps and relative team member locations. Specifically, the network does well instructing the robot on when stay still and when to move small distances, but the network has a harder time understanding how far the robot should move when it needs to move larger distances.

**CCN with Map Images and Pretrained VaDE Weights**

The final augmentation explored has the most substantial modification to the baseline network. In this modification, the encoding portion of the VaDE used in script phase learning is added to the architecture of the network. The augmented CCN architecture can be seen in Figure 4.39. The overlaid map images are passed to the VaDE encoder. The output of the encoder is passed to a fully connected layer that is of the same size as the flattened layer that went into the variational layers. RGB images and team member location vectors are passed to the network in the same way as during the baseline augmentation with maps and relative team member locations. The output from the RGB image CNN chain, the flattened output from the variational layers, and the relative team member locations are all concatenated together and passed to a fully connected layer of 100 nodes. The output of that layer is passed to a fully connected layer of 50 nodes. The output of this fully connected layer is passed to the two separate outputs that represent the linear and angular offset the robot should travel. It is important to note that the weights of the phase learning VaDE that are incorporated in the CCN are not adjusted while training the augmented CCN.

As is illustrated in Figure 4.40, during regression, the network is again able to learn to mimic the demonstrated angular and linear actions. The minimum angular MAE attained is 2.33 on the training set, 3.117 on the validation set, and 5.34 on the testing set. This equates to an average deviation of 5° between the appropriate angular offset and the predicted angular offset on the testing set. The minimum linear MAE attained is 0.298 on the training set, 0.344 on the validation set, and 0.381 on the testing set. This equates to an average deviation of 0.38 feet (≈4.5 inches) between the appropriate linear offset and the predicted linear offset. This shows significant improvement over the network with map features and no pretrained autoencoder, especially in the angular component. The difference in performance is a reduction of 7° in the angular offset and 2 inches in the linear component.

As can be seen in Figure 4.42, during classification, there is a significant improvement in learning when compared with the baseline CCN augmented with overlaid maps and team member relative locations. The network attained an angular accuracy of 89% on the training data, 85% on the validation data, and 81% on the testing data. The network attained a linear accuracy of 82% on the training data, 78% on the validation data, and 75% on the testing data. These results support this research’s hypothesis that incorporating the weights learned
Figure 4.37: Metrics reported during training baseline classification CCN with maps, relative team member locations, and one-hot phase and step IDs. The maximum angular accuracy attained is 83% on the training data and 80% on the validation data. The maximum linear accuracy attained is 74% on the training data and 70% on the validation data. The main takeaway is that this script step and phase IDs gives a slight boost in performance as compared to the same CCN without the script phase and step IDs.
Figure 4.38: Confusion matrices of baseline classification CCN with maps, relative team member locations, and phase and step IDs. The main takeaway is that the linear component still struggles to accurately classify larger linear movements.
Figure 4.39: Architecture of baseline CCN with maps, team member relative locations, and pretrained encoder from phase learning VaDE. The overlaid map images are passed to the VaDE encoder. The output of the encoder is passed to a fully connected layer that is of the same size as the flattened layer that went into the variational layers. RGB images and team member location vectors are passed to the network in the same way as during the baseline augmentation with maps and relative team member locations. The output from the RGB image CNN chain, the flattened output from the variational layers, and the relative team member locations are all concatenated together and passed to a fully connected layer of 100 nodes. The output of that layer is passed to a fully connected layer of 50 nodes. The output of this fully connected layer is passed to two separate outputs that represent the linear and angular offset the robot should travel. It is important to note that the weights of the phase learning VaDE that are incorporated in the CCN are not adjusted while training the augmented CCN.
Figure 4.40: Metrics reported during training baseline regression CCN with maps, relative team member locations, and pretrained weights. The key takeaway is the networks ability to learn to mimic the demonstrated movements with an average angular difference of 5° and an average linear difference of 4.5 inches on the testing set.
from the script phase learning VaDE would prove to be the most helpful form of augmenting the CCN.

The confusion matrices generated by the trained model are shown in Figure 4.42. The angular confusion matrices show more accurate, but similar results to those observed in the previous augmentation experiments. Specifically, the network does well instructing the robot on when to stay still and when to move small distances, but the network has a harder time understanding how far the robot should move when it needs to move larger distances.

4.5 Summary and Discussion

This research has shown the steps taken to develop a predefined team performing complex behaviors in multiple realistic simulation environments. Experiments are performed in simulation, data is collected from the team performing CCM, and models to capture the team’s SOP are trained and evaluated.

This dissertation has shown that the implemented VaDEs can successfully capture the structure and important high-level information demonstrated by a team performing CCM. The phase learning VaDE is shown to be able to cluster script phases with an accuracy of 79% on training data, 77% on validation data, and 74% on testing data. Additionally, the three script step learning VaDEs are shown to successfully cluster their corresponding script steps. Script steps for corner phases are clustered with an accuracy of 87% on validation data and 83% on test data, T-intersection scripts steps are clustered with an accuracy of 83% on validation data and 79% on test data, and room script steps are clustered with an accuracy of 51% on validation data and 48% on testing data. These metrics are presented succinctly Table 4.1. Moreover, the structure and transitions are learned by construction of a Petri net.

The VaDE networks perform well, but there is room for improvement, specifically with learning the room script steps. It is possible this phase has the most trouble due to the larger number of script steps, but more likely it is due to the disorder that is present in the team member’s formation, and subsequent map images, when the team exits the room. This could be mitigated by improving the information captured in the overlaid map images. Currently, only the location of each team member is overlaid on the map images as circles. Incorporating a shape that gives information about the orientation of each team member may prove useful.

Additionally, something that could improve learning for both methods is collecting and using more data samples. The models trained may take advantage of augmentation methods to increase the size of data sets, but the number of samples used to train and test methods is still small by historical standard. Attempting this solution has two main downsides, however. Firstly, a considerable amount of time is required to perform the simulation experiments and manually assign a script phase and label to each sample. Secondly, the time required to train the models, especially the VaDEs, becomes significantly longer. In this work, models are trained using an Apple iMac with 3.2 GHz Quad-Core Intel Core i5 and 8 GB of memory. The longest model to train is the phase learning VaDE trained with 29,800 rotated and balanced overlaid map images. Each epoch of training takes, on average, five minutes so training for 200 epochs takes about 17 hours. Other rotation augmentation methods were
Figure 4.41: Metrics reported during training baseline classification CCN with maps, relative team member locations, and pretrained weights. The maximum angular accuracy attained is 89% on the training data and 85% on the validation data. The linear angular accuracy attained is 82% on the training data and 79% on the validation data. The main takeaway is that using the pretrained weights results in the best performance.

Table 4.1: Performance comparison of script phase and script step learning VaDEs.

<table>
<thead>
<tr>
<th></th>
<th>Training</th>
<th>Validation</th>
<th>Testing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Phase</td>
<td>79%</td>
<td>77%</td>
<td>74%</td>
</tr>
<tr>
<td>Room</td>
<td>54%</td>
<td>51%</td>
<td>48%</td>
</tr>
<tr>
<td>T-intersection</td>
<td>86%</td>
<td>83%</td>
<td>79%</td>
</tr>
<tr>
<td>Corner</td>
<td>88%</td>
<td>87%</td>
<td>83%</td>
</tr>
</tbody>
</table>
Figure 4.42: Confusion matrices of baseline classification CCN with maps, relative team member locations, and pretrained weights. The main takeaway is that the network does really well at predicting angular values across the board. Additionally, the network does well at predicting linear values, but still has a harder time with predicting when larger linear values are appropriate.
investigated for overlaid map images that resulted in 65,560 images for training. The script
phase learning VaDE trained with 65,560 images was scheduled to take approximately two
days to train. This turned out to be too prohibitive and the smaller rotation augmentation
is used. Future work investigating the benefits of using larger data sets would likely benefit
from more specialized hardware that incorporated GPUs for training the networks.

This work has also shown that provided with inputs rich in high-level features, such as
the overlaid map images, the CCN can learn to accurately predict appropriate angular and
linear commands that conform to a predefined team’s CCN. Additionally, this research has
explored multiple ways of augmenting a CCN with learned script information and has found
significant increases in performances linked with some of those methods. A comparison of
the augmentation methods and their resulting MAEs is shown in Table 4.2. Additionally, a
comparison of the augmentation methods and their resulting accuracies is shown in Table
4.3. The tables show that there is a benefit to be had to using the script phase and step
IDs as inputs, but they in and of themselves are not enough to achieve meaningful learning.
The tables also show that augmenting the CCN with the weights and architecture of the
encoder learned from the script phase VaDE shows the greatest performance improvement in
general over the baseline CCN. This indicates that the unsupervised VaDE not only learns a
representation of the high-level map images that is beneficial for learning the script structure,
but it also learns important features that help guide the CCN’s learning process.

One point of interest in each CCN is the network’s ability to classify angular movements
more accurately than the linear counterparts. A potential improvement to help the network
improve on predicting all ranges of linear values would be to increase the computational
power of the linear output branch. This can be done by supplying more layers in the
network between where the fully connected layer of 50 nodes branches and the output for
the linear component. Another possible way to force the network to focus is to train two
separate networks, one for learning the angular component, and one for learning the linear
component. I believe developing two separate networks is not the best way to go. This route
could lead to overfitting of one or both outputs and could also become a practical problem
with compute power when trying to implement the approach on a robot in the real world.

When considering translating the developed approach to a real world scenario, it is helpful
to look at an example. Using the approach to develop a robotic agent to assist the previously
mentioned team of NYPD officers performing building clearing [42] is a perfect example. In
such a case, mapping and location data would need to be collected from the NYPD officers
performing the building clearing operation. This would likely not be too difficult as the
officers were already outfitted with VR headsets. These headsets have the capabilities to
track the orientation and location of the officers. Additionally, the virtual environment the
officers are immersed in would supply all software for mapping needs. Data could be collected
from officers training in VR and used to train the SOP models outlined in this work. After
sufficient training, the model could be equipped on a real world robot that would be able to
navigate with the officers that performed the training in VR.
Table 4.2: MAE performance comparison of methodologies for merging script information with regression CCN. All CCNs are trained using RGB images from the environment. Possible CCN inputs and augmentations are RGB Images (RGB), one-hot encodings of the phases and scripts (1H), map images used for training the script steps (map), team member relative \([x, y, \theta]\) positions (pos), and using network weights pretrained on the script phase learning VaDE (pre). The key takeaway is that merging the high-level script phase and step information has significant improvements to the CCN’s linear and angular performance.

<table>
<thead>
<tr>
<th></th>
<th>Angular (MAE)</th>
<th>Linear (MAE)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Training</td>
<td>Validation</td>
</tr>
<tr>
<td>Only RGB</td>
<td>22</td>
<td>22</td>
</tr>
<tr>
<td>+ 1H</td>
<td>20</td>
<td>20</td>
</tr>
<tr>
<td>+ map + pos</td>
<td>11</td>
<td>11</td>
</tr>
<tr>
<td>+ map + pos + 1H</td>
<td>9</td>
<td>10</td>
</tr>
<tr>
<td>+ map + pos + pre</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>

Table 4.3: Performance comparison of methodologies for merging script information with classification CCN. All CCNs are trained using RGB images from the environment. Possible CCN inputs and augmentations are RGB Images (RGB), one-hot encodings of the phases and scripts (1H), map images used for training the script steps (map), team member relative \([x, y, \theta]\) positions (pos), and using network weights pretrained on the script phase learning VaDE (pre). The key takeaway is that merging the high-level script phase and step information has significant improvements to the CCN’s accuracy.

<table>
<thead>
<tr>
<th></th>
<th>Angular (Accuracy)</th>
<th>Linear (Accuracy)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Training</td>
<td>Validation</td>
</tr>
<tr>
<td>Only RGB</td>
<td>20%</td>
<td>20%</td>
</tr>
<tr>
<td>+ 1H</td>
<td>20%</td>
<td>20%</td>
</tr>
<tr>
<td>+ map + pos</td>
<td>81%</td>
<td>73%</td>
</tr>
<tr>
<td>+ map + pos + 1H</td>
<td>83%</td>
<td>80%</td>
</tr>
<tr>
<td>+ map + pos + pre</td>
<td>89%</td>
<td>85%</td>
</tr>
</tbody>
</table>
Chapter 5

Summary and Conclusions

In this research I address the challenges of CCM for expert teams with robots by fusing model paradigms typically used independently for various human-robot navigation applications. The methods being fused are high-level, top-down AI methods and low-level, bottom-up behavioral cloning techniques. These methods have significant advantages when used individually and I demonstrate in this dissertation how they are able to be merged.

This dissertation addresses, to various degrees, the challenges described in Section 1.2. Step identification and low-level navigation are addressed fully in this work. This research develops VaDE networks to learn clusters of the script phases and script steps for identification purposes. Low-level navigation is accomplished in this work by training CCNs to determine the appropriate action the robot should take if it were put on the human-only team. This work addresses elements of novel step recognition, updating step transitions, and closing script loops by learning a Petri net that captures the transitions between script phases and steps. It is important to note that once the Petri net model is learned, it is not updated when being exposed to a new environment. This type of retroactive updating of the model is saved as a challenge for future work. Role adaptation is not performed in any way in this research but is a promising area of future work.

While the developed approach has shown success on learning the SOP of the predefined team in the experiments section, limitations do exist. Currently, the largest limitation is the need to specify how many script phases exist in the SOP as well as how many script steps exist per script phase. This was a simplifying assumption made with the understanding that it is common to train humans in a similar manner. Humans are not typically put on a team tabular rasa and told to figure out the team’s SOP. Instead, they would likely first be shown visual material similar to the graphics shown in Section 4.1 or a video of the desired behavior. I think it would be interesting to remove this simplifying assumption and let the VaDE networks have full control over learning the structure of the SOP. This could be accomplished using the “elbow method” wherein a different network would be trained on varying numbers of script phases or steps. Metrics revealing how well the learned GMM components cluster the data would be used to select the appropriate number of script phases and script steps.

An interesting area of future work could include exploring communication on the team performing CCM. Currently, the team members do not communicate with each other during experiments. Relaxing this simplifying assumption would allow an interesting study in
machine learning explainability to take place. The developed approach has the unique advantage that, due to its merging of high-level script information with low-level behavior cloning, it can give high-level, abstract descriptions of what its beliefs are about where the team is in its SOP and what it should be doing.

A second possible line of future work is increasing the complexity of the experiment environment by incorporating pedestrians. Currently, there are no other humans in the experiment environments apart from the team members. Incorporating some form of social element to the constraints being balanced in CCM would produce some interesting challenges. Adding and addressing this element would increase the scope of addressable problems and would have a serious impact in pedestrian-heavy applications such as security for college campuses and malls.
Bibliography


82
[9]

[9, 15]

[9, 15]

[9, 15]

[16]

[19]

[27]

[10]

[7, 8]

[7, 8]

[7]

[4]


Appendix A

Detailed Description of Baseline Behaviors

This Appendix gives specific details of team’s SOP when navigating a typical example of each area of interest. The images used in this Appendix are the same images used in Section 4.1; they are simply repeated here for the reader’s convenience.

A.1 Room Exploration

When the team approaches a room, as shown in Figure A.1a, they will perform the behaviors displayed in Figure A.1. The first step, as depicted in Figure A.1b, is to queue up outside the room along the wall leading to the door. This gets every team member set up to quickly enter the room one after another in the event the team members in the room need assistance. As each team member enters the room, they will stay close to the wall that the door is on or closest to. The next step, as shown in Figure A.1c, is for the lead team member to enter the room. While the lead team member is moving into the doorway, each successive team member moves up in the queue. The next step, Figure A.1d, is for the lead team member to move into the room to their required position. The lead team member will follow the wall that requires the least amount of directional change from their position queued up outside of the wall. This results in less restriction of maneuverability as they enter the room. While the lead team member moves to their position, each successive team member will move up in the queue or to the doorway. The next step, Figure A.1e, is for the second team member (blue) in the queue to take their position in the room. They will follow the alternative wall that the lead team member did not follow. This has the advantage of increasing the exploration capabilities of the team while keeping the team members separate and giving them more flexibility. The blue team member is at a disadvantage, though, in that they may need to perform a switchback when entering the room. This may limit maneuverability and increase their risk of danger. They do have the advantage that the lead team member has already entered the room and can supply support if needed. The next step, Figure A.1f, has the next team member in line (red) going to their specified location, which is to follow the path taken by the lead team member but not going so far into the room. The red team member will stay closer to the door, again, to increase the team’s overall ability to observe the room,
while also keeping them spread out. The next step, Figure A.1g, is for the last team member to enter the room. They follow the path of the second team member to enter (blue) and position themselves nearer to the door, similar to red’s position. The last step, Figure A.1h, is for all of the team members to file out of the room, taking up their previous positions for hallway exploration, and continue on the path to the specified end location.

A.2 T-Intersection Navigation

When the team approaches a hallway that is intersecting their hallway, as in Figure A.2a, and they wish to traverse the intersection, continuing on the path of their hallway, they will perform the SOP that is visualized in Figure A.2. In step 1, Figure A.2b, the team approaches the intersection and positions itself so the rightmost team member (red) is just about to look down the hallway. The other team members are backing this team member up. Figure A.2c shows that for step 2, the red team member and the lead team member (yellow) each enter the intersection, facing down the intersecting hallway. The next step, step 3 in Figure A.2d, yellow and red position themselves at the entrance of the intersecting hallway while the remaining team members enter the hallway. Finally, in step 4 shown in Figure A.2e, the team members regain hallway formation on the other side of the intersection and carry on with their hallway navigation.

When the team approaches a hallway that is intersecting their hallway, as in Figure A.3a, and they wish to take the intersecting hallway, they will perform the SOP that is visualized in Figure A.3. The team approaches the intersection in step 1 and positions itself so the rightmost team member (red) is just about to look down the hallway, visualized in Figure A.3b. The other team members are backing this team member up. Step 2, Figure A.3c, shows the red team member and the lead team member (yellow) each enter the intersection, facing down the intersecting hallway. Step 3, represented by Figure A.3d, has yellow and red position themselves at the entrance of the intersecting hallway while the remaining team members enter the hallway. Figure A.3e shows the last step, step 4, where the team members enter the new hallway, taking up their hallway formation and continue on their way.

When the team approaches a hallway that their hallway is intersecting, as in Figure A.4a, and they wish to take the left hallway path, they will perform the SOP that is visualized in Figure A.4. Step 1, depicted in Figure A.4b, shows how the team approaches the intersection and positions itself so the lead team member (yellow) can view a small portion of the left hallway while the rightmost team member (red) can view a small portion of the right hallway. This gives the team as much visibility to the unknown areas without exposing the forward team members too much. Step 2, Figure A.4c, has the yellow team member enter the left hallway, staying close to the wall while the red team member enters the right hallway while staying close to the wall. The rearmost (green) and rightmost (blue) team members move up in position to assist the team members in the hallway, if need be. Figure A.4d depicts step 3, in which the green and blue team members enter the hallway, facing opposite directions to support the yellow and red team members. Lastly, Figure A.4e details step 4, where all team members come together to form the hallway navigating behavior and continue on their way.
Figure A.1: Script steps of team entering a room. Steps specific to a room with a door in the center of its wall and the team entering with the doorway on its left-hand side.

Step 1: Queue up outside of room. Step 2: Lead team member (yellow) enters doorway. Step 3: Lead team member (yellow) continues in forward motion to most distant position. Step 4: Next team member (blue) performs switchback, continues to most distant position. Step 5: Next team member (red) continues in forward motion to position close to doorway. Step 6: Next team member (green) performs switchback, continues to position close to doorway. Step 7: All team members leave room and go to construct hallway formation.
Figure A.2: Script steps of team traversing an intersection in their hallway. Steps specific to a hallway intersecting the team’s hallway on the team’s right-hand side. Step 1: Position on brink of intersection. Step 2: Forward (yellow) and rightmost (red) team members position to see down intersecting hallway. Step 3: Left (blue) and rearmost (green) team members move forward into intersection. Step 4: All team members pass through intersection, forming hallway formation on the other side.
Figure A.3: Script steps of team **taking an intersection in their hallway**. Steps specific to a hallway intersecting the team’s hallway on the team’s right-hand side. Step 1: Position on brink of intersection. Step 2: Forward (yellow) and rightmost (red) team members position to see down intersecting hallway. Step 3: Left (blue) and rearmost (green) team members move forward into intersection. Step 4: All team members enter intersecting hallway, forming hallway formation on the other side.
Figure A.4: Script steps of team intersecting a hallway. Steps specific to intersecting a hallway and progressing to the left. Step 1: Lead team member (yellow) positions themselves on the brink of the left path of the intersected hallway. Rightmost team member (red) positions themselves on the brink of the right path of the intersected hallway. Step 2: yellow and red team members enter hallway, staying close to the wall, and facing opposite directions. Step 3: Leftmost (blue) and rearmost (green) team members enter the hallway, facing opposite directions, and support yellow and red team members. Step 4: All team members leave intersection, forming hallway SOP and continue on their way.
A.3 Corner Navigation

When the team approaches a 90-degree bend or corner in their hallway, as in Figure A.5a, they will perform the SOP that is visualized in Figure A.5. Step 1, depicted in Figure A.5b, shows how the team approaches the corner and positions itself so the rightmost team member (red) can view a small portion of the right hallway past the corner. The lead team member (yellow) is directly backing up the red team member and can view a small portion of the hallway as well. This gives the team as much visibility to the unknown areas without exposing the forward team members too much. Step 2, shown in Figure A.5c, has all team members fully entering the corner. The red team member stays close to the wall they were just peeking around, the yellow team member makes room for the rightmost team member (blue) and the rearmost team member (green). The blue and green team members back up the red and yellow team members. The final step, step 3 as shown in Figure A.5d, shows the team re-entering the hallway, forming back into their hallway SOP and continuing on their way.
Figure A.5: Script steps of team navigating a corner. Steps specific to a corner that bends at a 90-degree angle to the team’s right-hand side. Step 1: Position so forward (yellow) and rightmost (red) team members can view hallway around corner. Step 2: All team members move into corner. Step 3: All team members move into hallway, forming hallway formation.
Appendix B

Detailed Description of Behavior Variants

B.1 Room Entry Variants

A significant variant to the room with a door in the center of a wall is a room with a door tucked into one of its corners. Figure B.1 shows such a room where the door is located in the room’s bottom left corner. This modifies where the agents can go, as two of them must now position themselves perpendicular to the other two instead of all agents being in the same line. In this example, the agents are approaching the room with the door on their left-hand side. This means, in order to increase mobility, the first team member through the door is going to maintain their forward momentum and go to the far corner as depicted in Figure B.1a. The following team members proceed in the same fashion as before, alternating positioning on the opposite wall to the lead team member and positioning on the same wall as the lead team member.

A similar variant to this last example is if the team were to approach the same room, having the door in the bottom left corner, but with the entrance being on their right, as displayed in Figure B.2a. In this case, the lead team member’s forward momentum would take them through the door and up the side of the wall directly in front of them. The following team members would continue the standard of alternating switchbacks and following the lead team member.

The last two variants involve rooms with doorways in their bottom right corners. These variants demonstrate very similar behaviors to B.1 and B.2. If the room is being approached on the team’s left, as in Figure B.3, the lead team member’s forward momentum is going to take them up the wall directly in front of them, instead of taken them deeper into the room. The following team members alternate switchbacks and follow the lead team member.

If however, the room is being approached on the team’s right, as in Figure B.4, then the lead team member’s forward momentum is going to take them forward, deeper into the room. The following team members alternate going along the perpendicular wall directly in front of them and following the lead team member.
Figure B.1: Script steps of team entering a room. Steps specific to a room with a door in the leftmost corner of its wall and the team entering with the doorway on its left-hand side. Behavior is similar to that demonstrated in Figure A.1 of Appendix A with the primary differences being the left (blue) and rearmost (green) team members are not performing a full switchback due to the location of the doorway in the room.

Figure B.2: Script steps of team entering a room. Steps specific to a room with a door in the leftmost corner of its wall and the team entering with the doorway on its right-hand side. Behavior is similar to that demonstrated in Figure A.1 of Appendix A with the main difference being the final team member positions in Step 7. Note the left (blue) and rearmost (green) team members are performing a full switchback due to entry angle to room.
Figure B.3: Script steps of team entering a room. Steps specific to a room with a door in the rightmost corner of its wall and the team entering with the doorway on its left-hand side. Behavior is similar to that demonstrated in Figure B.2 of Appendix B.

Figure B.4: Script steps of team entering a room. Steps specific to a room with a door in the rightmost corner of its wall and the team entering with the doorway on its right-hand side. Behavior is similar to that demonstrated in Figure B.1 of Appendix B.
B.2 T-Intersection Variants

The variant behavior for the team traversing an intersection comes when the intersection is on the team’s left-hand side instead of its right-hand side. When this happens, the behavior used to traverse the intersection is very similar to that detailed in 4.3. The main difference is that due to the intersection being on the left instead of the right, the leftmost agent (blue) will be responsible to scout out the intersecting hallway with the lead team member, instead of the rightmost team member performing that role.

The variant behavior for the team taking an intersection comes when the intersection is on the team’s left-hand side instead of its right-hand side. When this happens, a similar effect happens, as described in Figure B.5. In this case, given a circumstance depicted in Figure B.6, it is again the leftmost team member’s responsibility to cover the intersecting hallway along with the lead team member, while the rightmost and rear team members supply backup.

The variant for intersecting a hallway comes from the team needing to take the right-hand hallway instead of the left-hand hallway as shown in Figure B.7. In this case, the lead team member (yellow) covers the right hallway while the leftmost team member (blue) covers the left hallway. The following steps proceed in the same manner as Figure A.4 with the leftmost team member being the lead scout with the yellow team member, while the leftmost and rearmost team members are primarily backup.

B.2.1 Corner Variants
Figure B.5: Script steps of team traversing an intersection in their hallway. Steps specific to a hallway intersecting the team’s hallway on the team’s left-hand side. Step 1: Position on brink of intersection. Step 2: Forward (yellow) and leftmost (blue) team members position to see down intersecting hallway. Step 3: Right (red) and rearmost (green) agents move forward into intersection. Step 4: All team members pass through intersection, forming hallway formation on the other side.
Figure B.6: Script steps of team taking an intersection in their hallway. Steps specific to a hallway intersecting the team’s hallway on the team’s left-hand side. Step 1: Position on brink of intersection. Step 2: Forward (yellow) and leftmost (blue) team members position to see down intersecting hallway. Step 3: Right (red) and rearmost (green) agents move forward into intersection. Step 4: All team members enter intersecting hallway, forming hallway formation on the other side.
Figure B.7: Script steps of team intersecting a hallway. Steps specific to intersecting a hallway and progressing to the right. Step 1: Lead team member (yellow) positions themselves on the brink of the right path of the intersected hallway. Leftmost team member (blue) positions themselves on the brink of the left path of the intersected hallway. Step 2: yellow and blue team members enter hallway, staying close to the wall, and facing opposite directions. Step 3: Rightmost (red) and rearmost (green) team members enter the hallway, facing opposite directions, and support yellow and blue team members. Step 4: All team members leave intersection, forming hallway SOP and continue on their way.
Figure B.8: Script steps of team navigating a corner. Steps specific to a corner that bends at a 90-degree angle to the team’s left-hand side. Step 1: Approach corner. Step 2: Position so forward and rightmost team members (yellow and blue) can view hallway around corner. Step 3: All team members move into corner. Step 5: All team members move into hallway, forming hallway formation.
Vita

Joshua Fagan attended the University of Richmond (UR), in Richmond Virginia for his undergraduate degree. Joshua received a Bachelor of Science degree from UR, double majoring in Computer Science and Mathematics. After graduating, Joshua pursued a PhD in Computer Science at the University of Tennessee (UTK), in Knoxville Tennessee. While working towards his doctorate, Joshua took internships at the National Transportation Research Center in Tennessee, and the Army Research Laboratory in Maryland. Additionally, Joshua earned a Masters of Science in computer science in the early stages of his time at UTK. Joshua has a love of teaching and has taught classes in the Computer Science department and the Engineering Fundamentals department at UTK.