ApproachFinder: Real-time Perception of Potential Docking Locations for Smart Wheelchairs

by

Shivam Thukral

B. Tech, Indraprastha Institute of Information Technology Delhi, 2017

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

Master of Science

in

THE FACULTY OF GRADUATE AND POSTDOCTORAL STUDIES
(Computer Science)

The University of British Columbia
(Vancouver)

February 2022

© Shivam Thukral, 2022
The following individuals certify that they have read, and recommend to the Faculty of Graduate and Postdoctoral Studies for acceptance, the thesis entitled:

**ApproachFinder: Real-time Perception of Potential Docking Locations for Smart Wheelchairs**

submitted by *Shivam Thukral* in partial fulfillment of the requirements for the degree of *Master of Science* in *Computer Science*.

**Examing Committee:**

Ian M. Mitchell, Professor, Department of Computer Science, UBC  
*Supervisor*

Jim Little, Professor, Department of Computer Science, UBC  
*Supervisory Committee Member*
Abstract

A smart wheelchair improves the quality of life for older adults by supporting their mobility independence. Some critical maneuvering tasks, like table docking and doorway passage, can be challenging for older adults in wheelchairs, especially those with additional impairment of cognition, perception or fine motor skills. Supporting such functions in a shared manner with robot control seems to be an ideal solution. Considering this, we propose to augment smart wheelchair perception with the capability to identify potential docking locations in indoor scenes. ApproachFinder-CV is a computer vision pipeline that detects safe docking poses and estimates their desirability weight based on hand-selected geometric relationships and visibility. Although robust, this pipeline is computationally intensive. We leverage this vision pipeline to generate ground truth labels used to train an end-to-end differentiable neural net that is 15x faster. ApproachFinder-NN is a point-based method that draws motivation from Hough voting and uses deep point cloud features to vote for potential docking locations. Both approaches rely on just geometric information, making them invariant to image distortions. A large-scale indoor object detection dataset, SUN RGB-D, is used to design, train and evaluate the two pipelines.

Potential docking locations are encoded as a 3D temporal desirability cost map that can be integrated into any real-time path planner. As a proof of concept, we use a model predictive controller that consumes this 3D costmap with efficiently designed task-driven cost functions to share human intent. This controller outputs a nominal path that is safe, goal-oriented and jerk-free for wheelchair navigation.
Lay Summary

According to the Canadian Survey on Disability 2017, 22% of the Canadian population suffers from some form of disability, and among older adults specifically, mobility impairment affects 24%. This thesis aims to assist the development of smart wheelchairs that can seamlessly collaborate with people to accomplish their mobility goals. To achieve this, we developed two algorithms that can find safe locations to park (or “dock”) a wheelchair near indoor objects.

First, we developed a vision pipeline using standard computer vision techniques and hand-tuned geometric properties. Next, we leveraged this vision pipeline to train a neural network that provides similar results with far fewer computational resource requirements. Finally, we present a way to integrate these safe locations into any shared controller. The larger goal of this work is to develop a semi-autonomous docking assistance system that provides a collision-free path for wheelchair navigation.
Preface

This thesis presents the original work done by the author, Shivam Thukral, conducted in Vision-Control-Robotics Laboratory at the University of British Columbia under the supervision of Dr Ian M. Mitchell.

The author adopted code for object detection from Votenet[44]. Votenet also inspires the proposed neural network used to find safe docking locations. The skeleton code for the shared controller and robot dynamics model was adopted from Autorally[63]. Apart from the adopted code, the design, implementation, experiments, and results were completed by the author, with feedback from his supervisor.
Table of Contents

Abstract . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . iii
Lay Summary . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . iv
Preface . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . v
Table of Contents . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . vi
List of Tables . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . ix
List of Figures . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . x
Glossary . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . xiii
Acknowledgments . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . xiv

1 Introduction . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 1
  1.1 Motivation . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 3
  1.2 Contributions . . . . . . . . . . . . . . . . . . . . . . . . . . . . 3
  1.3 Thesis Overview . . . . . . . . . . . . . . . . . . . . . . . . . . . 4

2 Related Work . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 5
  2.1 Indoor Object Localisation . . . . . . . . . . . . . . . . . . . . . . . 5
  2.1.1 3D Deep Learning on Point Clouds . . . . . . . . . . . . . . . . 6
  2.2 Potential Goal Identification . . . . . . . . . . . . . . . . . . . . . . 8
  2.3 Shared-Control Approaches for Wheelchair Navigation . . . . . . . . 9
3 ApproachFinder-CV .................................................. 11
  3.1 Approach ...................................................... 12
    3.1.1 Candidate Object Extraction .......................... 13
    3.1.2 Edge Extraction for Candidate Objects .............. 15
    3.1.3 Sample and Weight Potential Docking Locations .... 16
  3.2 Desirability Costmaps ........................................ 18
  3.3 Implementation Details ..................................... 19
  3.4 Experimental Results ...................................... 22

4 ApproachFinder-NN ................................................. 25
  4.1 Approach ...................................................... 26
    4.1.1 Hough Voting .......................................... 27
    4.1.2 Backbone Network: PointNet++ ......................... 28
    4.1.3 Voting Module ........................................ 29
    4.1.4 Vote Aggregation ...................................... 30
    4.1.5 Proposal Module ...................................... 31
  4.2 Implementation Details ...................................... 32
  4.3 Results ........................................................ 33
  4.4 Analysis Experiments ....................................... 35

5 Shared Control for Wheelchairs ................................. 41
  5.1 Properties for a shared-control wheelchair system ........ 42
  5.2 Model Predictive Controller Design ......................... 42
    5.2.1 Dynamics Model ....................................... 43
    5.2.2 Cost Functions ....................................... 45
  5.3 Implementation Details ...................................... 46
    5.3.1 Mapping ............................................... 46
    5.3.2 Dynamic Obstacle Avoidance ......................... 46
    5.3.3 3D Desirability Costmap .............................. 46
    5.3.4 Navigation ........................................... 47
  5.4 Experimental Results ....................................... 48

6 Conclusions ........................................................ 51
  6.1 Limitations .................................................. 52
6.2 Future Work ........................................ 52

Bibliography .............................................. 53
### List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table 4.1</td>
<td>PointNet++ layer specifications used in implementation</td>
<td>33</td>
</tr>
<tr>
<td>Table 4.2</td>
<td>ApproachFinder-NN vs BApproachFinderNN: Baseline mode proposes directly form seed points. Average precision (AP) and Recall are at 0.25 IoU.</td>
<td>38</td>
</tr>
<tr>
<td>Table 4.3</td>
<td>Results of different symmetric pooling functions</td>
<td>39</td>
</tr>
<tr>
<td>Table 4.4</td>
<td>Effect of different sampling strategy</td>
<td>40</td>
</tr>
<tr>
<td>Table 4.5</td>
<td>Effect of varying input features</td>
<td>40</td>
</tr>
</tbody>
</table>
# List of Figures

| Figure 1.1 | Pointcloud data from a depth sensor is passed through a 3D object detection network to narrow down the search space to candidate objects. Using ApproachFinder-CV or ApproachFinder-NN, we generate proposed docking locations that are then encoded as a desirability costmap. This costmap is used by a shared controller to generate robot control commands. | 2 |
| Figure 3.1 | ApproachFinder-CV: The input point cloud (a) is processed by a pre-trained network, Votenet, to find candidate objects using bounding boxes (b). The point cloud inside these bounding boxes is used to find planar surfaces. Next, the vision pipeline approximates a convex polygon to cover all the edges of such planar surfaces (c). We sample potential docking locations around these convex polygon edges (d) and filter them based on their visibility and position (e). Finally, these filtered locations are combined to generate a desirability costmap that can be integrated with any real-time planner (f). | 12 |
| Figure 3.2 | (a) Votenet bounding box detections in simulation; (b) Votenet detections on samples taken from SUN RGB-D dataset. | 14 |
| Figure 3.3 | Detected planar surface and approximated convex polygon. Red points are plane inliers and green spheres represent the vertices of the convex polygon. | 16 |
| Figure 3.4 | Docking locations generated by ApproachFinder-CV with corresponding headings. | 17 |
Figure 3.5  (a) Large and (b) small office environment in Simulation  
Figure 3.6  Robot in the Gazebo simulation with image and point cloud captured by the camera.  
Figure 3.7  Summary for ApproachFinder-CV algorithm. Each module (in blue) is further elaborated with the associated steps. On top, we provide the message information that is passed through Robot Operating System (ROS).  
Figure 3.8  ApproachFinder-CV results on SUN RGB-D dataset. Cyan spheres represent the docking locations and grey arrows represent the ideal orientation.  
Figure 4.1  ApproachFinder-NN: The input object point cloud is sub-sampled using PointNet++. These sub-sampled points are called seed points. Each seed point votes for its nearest docking location. These votes are aggregated into clusters and are further processed by the proposal module to generate docking locations.  
Figure 4.2  Comparison between ApproachFinder-CV (left column) with ApproachFinder-NN (right column)  
Figure 4.3  Additional qualitative results for the two pipelines.  
Figure 4.4  Effect of increasing seed layers in PointNet++  
Figure 4.5  Effect of increasing vote aggregation radius  
Figure 5.1  An Model Predictive Control (MPC) based method that uses a system dynamics model and task oriented cost functions to generate control plans. These control plans are optimised in real-time on a Graphical Processing Unit (GPU).  
Figure 5.2  Navigation stack for the proposed approach using an MPC controller. Each node in the system is accompanied by a small description that highlights its importance.  
Figure 5.3  An example Gazebo simulation environment.  
Figure 5.4  Robot model used as a rough substitute for a wheelchair.  
Figure 5.5  Results from gmapping and costmap_2d ROS packages.  

xi
Figure 5.6 Finding docking locations around a table using ApproachFinder-CV. A similar set of results were observed with ApproachFinder-NN.  
Figure 5.7 Navigation using the shared controller.
Glossary

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Full Form</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADL</td>
<td>Activities of Daily Living</td>
</tr>
<tr>
<td>CNN</td>
<td>Convolutional Neural Network</td>
</tr>
<tr>
<td>FPS</td>
<td>Farthest Point Sampling</td>
</tr>
<tr>
<td>FOV</td>
<td>Field of View</td>
</tr>
<tr>
<td>GPU</td>
<td>Graphical Processing Unit</td>
</tr>
<tr>
<td>MPC</td>
<td>Model Predictive Control</td>
</tr>
<tr>
<td>MLP</td>
<td>Multi-Layer Perceptron</td>
</tr>
<tr>
<td>ROS</td>
<td>Robot Operating System</td>
</tr>
</tbody>
</table>
Acknowledgments

This work could not have been possible without the immeasurable support and guidance of my supervisor, Dr Ian M. Mitchell. I would also like to thank him for allowing me to work on this project at the University of British Columbia. I am thankful to my lab-mates for their helpful feedback and discussions.

I would also like to thank my second reader, Dr Jim Little, for taking out the time to read my work and provide insightful comments.

A special thank you to my family and friends for their unconditional love and support. I would especially like to thank my sister, Shubham Thukral, for her moral support and for proofreading my thesis.
Chapter 1

Introduction

Individuals who suffer from significant motor impairment depend on powered wheelchairs for their daily mobility needs. To accommodate the population of users who find powered wheelchairs challenging to operate, several researchers have contributed towards the automation and safety of such maneuvering tasks. A smart wheelchair is a solution that is cognitively and perceptually aware of its environment and can incorporate user intentions while executing safe and smooth control paths in complex environments. A key step in the development of such shared control paradigms for smart wheelchairs is to enhance the 3D perception capabilities of the vision system. For a smart wheelchair, as with a human, goal identification plays a vital role in enhancing system capabilities. Two well-researched themes in this domain are passage through an open doorway and docking to locations. These docking locations are user identified goals parameterised by position and heading. In the literature, the majority of the work is done towards perception and safe passage through doorways [10, 14] but identifying safe docking locations in indoor environments is still an open problem.

Finding goals in an indoor environment is a challenging task. Issues include object segmentation, small object clutter, and occlusions. To handle such scenarios, smart wheelchairs fuse data from multiple sensors to perceive safe docking locations. Color images provide semantic cues that facilitate object detection but are susceptible to changes in illumination and contrast. Compared to images, depth data provides accurate information about object geometry from active depth sen-
sors and is robust to changes in lighting conditions. However, depth point clouds are challenging data due to their unordered structure and varying density. Typical convolutional neural networks (CNNs) require regular data to perform weight sharing and kernel optimisations. Volumetric-based methods convert a point cloud into a 3D grid representation so that CNNs can be used. By comparison, our method is a point-based method that does not introduce explicit information loss due to such voxelisation.

Figure 1.1 provides an overview of the pipelines presented in this thesis: Two approaches to find potential docking locations with proper pose alignment from just geometric 3D point cloud information. Each location is accompanied by a desirability weight that is based on the location’s relative position and visibility. The first approach is a computer vision pipeline, ApproachFinder-CV, which uses a 3D object detection network to find candidate objects (tables/toilets) and then sequentially searches for potential docking locations along edges of these objects. Although robust, this approach is computationally slow and resource-intensive due to sequential sampling. The second approach is inspired by Hough voting which uses 3D deep features to directly generate docking proposals. We leverage our computer vision pipeline to generate training data for ApproachFinder-NN. Our deep neural network is real-time and computationally efficient, which makes it suitable for online robot implementation.

![Diagram](image)

**Figure 1.1:** Pointcloud data from a depth sensor is passed through a 3D object detection network to narrow down the search space to candidate objects. Using ApproachFinder-CV or ApproachFinder-NN, we generate proposed docking locations that are then encoded as a desirability costmap. This costmap is used by a shared controller to generate robot control commands.

As a proof-of-concept, we encode output from either docking location detector as a desirability costmap, and integrate it with an existing sample-based model
predictive controller. This shared control system samples millions of trajectories per second and evaluates each trajectory to generate a nominal path which is similar to the user’s input and moves toward identified docking location(s).

1.1 Motivation

In Canada, one-third of adults aged 65 or greater suffer from mobility impairment [8]. Almost 26% of the eligible individuals are denied access to a powered wheelchair because of their mental and physical disabilities [17]. A majority of the population in this group suffers from cognitive impairment and this number is expected to rise to 1.4 million by the end of 2030 [37]. Individuals lacking independent mobility often suffer from low self-esteem due to limited social interactions and minimal leisure activities [23]. Due to high dependence on caregivers, such individuals develop a sense of burden which results in reduced quality of life [18].

To restore their quality of life, they must regain their mobility. For this, a smart wheelchair that is visually aware can play a vital role. A survey estimates about two million individuals could benefit from such systems [54]. Such systems are truly beneficially if they can provide additional assistive cues that can facilitate Activities of Daily Living (ADL).

Table docking falls under the category of challenging tasks due to constrained indoor space and accurate maneuvering. In such cases, wheelchair users can easily collide with the docking objects and injure themselves. The task of docking is also included in many wheelchair driving assessments like PIDA [13] and PM-CDA [28]. The focus of this thesis lies in facilitating the task of table docking by (i) identifying candidate docking objects; (ii) detecting safe docking locations around such objects; (iii) providing a shared control system which gives preference to user intent while abiding by the safety constraints.

1.2 Contributions

The broader aim of this work is to develop an autonomous docking assistance system for smart wheelchairs. This system should not only find potential docking locations but should also generate safe and smooth trajectories according to user intentions. To achieve this, we make the following contributions:
• Present a novel 3D vision-based algorithm, ApproachFinder-CV, to detect feasible goal locations around tables and toilets using just geometric information from point clouds. Each potential location is accompanied by its desirability weight based on its visibility, relative position and heading.

• Present a real-time deep network that is inspired by Hough voting to predict docking spots and their desirability weight around objects of interest. ApproachFinder-NN is an end-to-end differentiable architecture that is 15x faster than the computer vision pipeline.

• Evaluate the performance of ApproachFinder-NN on a large state-of-the-art indoor dataset, SUNRGB-D, along with a description of how elements of the architecture and certain hyperparameters of the neural network were determined experimentally.

• Propose a way to integrate the algorithms’ output as a 3D temporal desirability cost map using a log-likelihood formulation for shared control path planning of smart wheelchairs.

• Present a Model Predictive Control (MPC) based shared controller with task driven cost functions that generate user-intent based, safe, goal-oriented and jerk-free trajectories for wheelchair navigation in real-time.

1.3 Thesis Overview

The rest of the thesis is organized as follows: Chapter 2 provides a summary of the literature on goal identification and docking location detection in indoor environments for wheelchairs. This chapter also covers the shared control techniques that are being used to capture human intent. Chapter 3 introduces our computer vision pipeline to find potential docking locations in indoor scenes. In Chapter 4, we describe how we used this computer vision pipeline to train a deep network that generates similar results. The shared MPC based controller prototype is presented in Chapter 5. A summary and future avenues of this research are discussed in Chapter 6.
Chapter 2

Related Work

This chapter is divided into three parts. The first section summarises state-of-the-art deep learning approaches to find objects in indoor environments (Sec. 2.1). The next section highlights major contributions done towards identifying possible furniture docking locations around these indoor objects (Sec. 2.2). Finally, the third section discusses different approaches to deal with wheelchair navigation while considering human intent (Sec. 2.3).

2.1 Indoor Object Localisation

The task of object localisation is extensively studied in the computer vision literature with both 2D and 3D data [2, 24]. Some approaches focus on combining semantic cues from images with geometric information from point clouds while others work on different data streams separately. This section summarises the major contributions done towards 3D object localisation for indoor scenes.

Early work in this domain [11] involved the basic steps of feature extraction, clustering and applying simple machine learning techniques for object classification or segmentation. Feature extraction methods like 3D shape contexts [6] subdivide a spherical region into bins while other methods utilised geometric properties of point clouds, such as point normals, to extract local feature regions. These extracted features are consumed by machine learning techniques, like support vector machines, to classify local regions [40]. These extracted features were used for
keypoint selection making them specific to a certain object. These methods captured local features and ignored global features making them unsuitable for large scale dense segmentation tasks.

The success of deep learning techniques in the 2D domain has driven a recent shift towards applying similar technology on 3D point clouds. This shift is supported by the availability and affordability of RGB-D cameras and GPU compute[34]. The following subsection provides insights into such deep learning algorithms.

2.1.1 3D Deep Learning on Point Clouds

We will focus on object detectors that take a point cloud as input and produce oriented 3D bounding boxes that localise different objects in the scene. In some cases, these results are further refined by predicting per point segmentation labels inside these bounding boxes. Earlier work in this domain involved handcrafted features for task specific problems. Later, they were replaced with deep point features that encode both local and global properties of point clouds. These deep features are highly generalisable for multiple tasks while being invariant to affine transformations.

3D object detection can be divided into two categories: single shot and proposal based methods. Single shot methods use a single stage network to predict the semantic class and oriented bounding box of candidate objects. They lack complex post-processing steps which make them much faster than proposal based methods. Single shot methods can be further divided into discretization-based or point-based methods. Discretization-based methods convert point cloud data into discrete voxel grids which make them suitable for traditional Convolutional Neural Network (CNN)s. Li et al. [33] transformed a point cloud into a 2D point map and predicted bounding boxes using 2D FCN. VoxelNet [73] divides the point cloud into voxels that are encoded as tensors. These tensors are used by the region proposal network to produce bounding boxes. Sindagi et al. [55] extended VoxelNet by fusing non-empty voxels with corresponding image features given by a pre-trained network. This set of concatenated features produced 3D box proposals. However, such volumetric representations are constrained by point cloud resolu-
tion and 3D convolution cost. Point based-methods work directly with the input point clouds and thereby avoid the cost of and geometric information loss from voxelisation. Yang et al. [69] proposed 3DSDD that uses a fusion sampling strategy to make detection on less representative points feasible. This approach also uses a Feature-FPS to remove time-consuming feature propagation layers.

Proposal based methods produce several proposals and use region wise features to predict the labels. Proposal-based methods are intuitive but require multi-stage training and removal of the redundant proposals as a post-processing step. Yang et al. [68] used a backbone 2D segmentation network to distinguish background and foreground pixels, and then used this labeling to project foreground points to generate proposals. Shi et al. [51] proposed PointRCNN in which they obtain foreground points directly from segmented 3D point clouds and fuse semantic and local features to produce 3D bounding boxes. F-PointNets [43] is a frustum based method that uses a 2D object detector to generate 2D candidate regions, and applies PointNet [41] to learn 3D features of each frustum for 3D box estimation. Following this, Zhou et al. [70] proposed a method that was invariant to scale. Specifically, they integrated a Point-SENNet and a PointSIFT module into the network to capture scale and orientation information, respectively, of the point clouds. Xu et al. [66] fused image encoding with corresponding frustum encoding to propose multiple bounding boxes.

Votenet [44] is a Hough voting based method that votes for virtual object centers, and groups these votes using a vote aggregation strategy to produce 3D object proposals. Votenet significantly outperforms previous approaches using just geometric information, achieving cutting-edge results on two large indoor datasets (ScanNet [12] and SUN RGB-D [56]). Recently, Qi et al. [45] proposed ImVotenet which fuses 2D votes/features from images and 3D votes from point clouds using a multi-tower training scheme to produce 3D proposals. This fusion of 2D and 3D data increased the overall performance of the network.

In some cases, networks go one step further and predict pixel-level segmentation masks inside these localised objects. Yang et al. [67] proposed an end-to-end single stage network, 3D-BoNet, which achieves instance level segmentation on point clouds. This method directly regresses the 3D bounding boxes for all the objects in the scene, and uses a per-point classifier to get class labels. This method
is computationally efficient as it doesn’t require any post processing step.

2.2 Potential Goal Identification

A key step in the development of a shared control paradigm for assistive wheelchairs is to enhance the 3D perception capabilities of the system.

One challenging perceptual task is to find navigation goals. For an indoor environment, challenges include object segmentation, frequently occurring small objects, and heavy occlusions; all are aggravated by the randomness and disorder which is common in indoor scenes. Previous work in this domain relied on a priori knowledge of the environment to identify goal locations for humans. This information combined with human intent is used for navigation in [9]. Our goal is to facilitate operations in novel environments, thus lifting the requirement of a known map and specialised sensing.

With the advent of low-cost RGB-D sensors and the use of the additional depth information provided, detection of such regions became more practical for real-world applications [27]. Deep learning techniques for object detection and segmentation play a crucial role in finding salient locations in indoor scenes; for example, RetinaNet [1] is a deep network for assistance navigation by visually impaired persons in indoor environments.

In the literature, two goal locations for wheelchairs have been extensively studied: doorways and docking locations. The phrase “docking location” is used to refer to a spot at which the user may wish to stop the wheelchair near a stationary object in order to interact with that object; for example, stopping next to a toilet to facilitate a transfer onto the toilet seat. It does not necessarily imply that the location is pre-specified (contrast to a charging dock for a robot vacuum cleaner) and usually avoids physical contact or connection with the object (contrast to mooring a boat to a dock).

Initial work in doorway passage involved finding door frames [5, 71], by first projecting and transforming the image onto a 2D gradient height map, and then searching for horizontal lines to find door frames. Derry et al. [14] proposed an approach to automatically detect open doorways using point cloud data. They used the location and orientation of the detected door to plan custom navigation tra-
jectories. Jain et al. [25] presented an algorithm for the automated detection of safe docking locations at rectangular and circular structures (like tables or desks) with proper alignment using 3D point cloud data. Additionally, template matching algorithms are also used to detect frequently occurring indoor objects like chairs.

Many approaches in the literature make use of fiducial markers or landmarks to ease the recognition task. Weber et al. [62] used orange fruit as a fiducial mark for their neural network algorithm, while [35] uses orange tape along table edges as its marker. Other approaches simplify the problem by focusing on customised docking stations. Ren et al. [49] used a fiducial T-shaped marker along with a specific U-shaped bed docking. Furthermore, approaches based on images are susceptible to vision distortions (such as lighting, colors, and contrast) which can limit the complete system. Some approaches rely on human perception to identify goal locations [53]. Similar to our work, there are approaches that utilise point clouds to find supporting planes on the goal objects; for example, Williamson et al. [64] used point cloud data to find plane-like structures using EM algorithm, but they required a priori knowledge of the number of clusters.

To the best of our knowledge, no work has attempted to solve the goal identification problem for smart wheelchairs for multiple indoor object classes with different geometries. Our work is focused on identifying parking locations around tables and toilets using point cloud data with no prior knowledge of the scene. We selected tables and toilets because they are two of the most common objects to which a powered wheelchair user may want to dock [60, 61] and for which we had suitably labeled sensor training data [56].

2.3 Shared-Control Approaches for Wheelchair Navigation

Although a growing number of tasks can be performed autonomously by robots, there are still many tasks where human judgement plays an important role or is preferred [36]. Combining human intent with semi-autonomous behaviour has been extensively explored for powered wheelchairs. Vanhoonydonck et al. [59] presented a framework where the path is chosen by user intention. A path planner provides navigation orders to the wheelchair’s shared controller, and user intention
is expressed through joystick commands. In the case of conflict, the planner may modify the user’s commands [39] or re-plan the path [9]. NavChair [32] is one of the pioneer works in this domain that is based on a commercial wheelchair with additional sensors to assist with wheelchair navigation. It employs three operating modes: obstacle avoidance, door passage and wall following. Goil et al. [19] presented a machine learning technique to assist individuals with impairments in challenging driving scenarios.

In [3], the authors develop an MPC based framework that blends human intent and obstacle avoidance based on the threat of collision. Later, they extended this work to include stability. Anderson et al. [4] proposed a semi-autonomous obstacle avoidance control where they considered safe regions (homotopies) rather than a single optimal path. In [15] authors presented a shared control framework for obstacle avoidance and stability using safe driving envelopes. At each timestep, an MPC controller determines if the current user control command permits a safe vehicle trajectory within these envelopes. Authors in [57] suggest an MPC based framework which allows the user to control at a higher level without comprising the safety added by semi-autonomous behaviors.

The previous paragraph should make clear that MPC is a popular approach to achieve the blending of user control input, obstacle avoidance, and possibly navigation support in shared-control wheelchairs. We also adopt an MPC approach, building from the AutoRally implementation by Williams et al. [63] that uses a sampling-based, derivative-free optimization algorithm. At each time step, importance sampling is used to generate potential sequences of control inputs. Trajectories are approximated using these sequences and the system dynamics model, and each trajectory is evaluated according to a set of cost functions. The estimate of the optimal control sequence is then constructed through a cost-weighted average over the sampled trajectories. The algorithm is implemented on a GPU, whose massive parallelism permits thousands of trajectories to be sampled and evaluated in milliseconds and thereby enables real-time closed-loop control at a moderate frequency (20-30 Hz).
Chapter 3

ApproachFinder-CV

In this chapter, we first discuss our computer vision pipeline, ApproachFinder-CV, to detect docking locations in indoor scenes (Sec. 3.1). Next, we talk about a way in which we can encode docking locations as temporal desirability costmaps for any real-time path planner (Sec. 3.2). This is followed by implementation specific details (Sec. 3.3). Finally, we provide experimental results on a scenes taken from a state-of-the-art indoor dataset (Sec. 3.4).

**Problem Statement** The objective of this research is to find furniture docking locations in indoor environments which will serve as potential wheelchair goals. To achieve this, we use as input to the algorithm point cloud data, which is invariant to visual distortions and provides geometric information about the scene. A point cloud is a set of 3D points \( \{ p_i | i = 1, \ldots, n \} \), where each point \( p_i \) is a \((3+C)\) dimensional vector of its coordinates \((X,Y,Z)\) and a feature channel \(C\) (color, normal, etc). The outputs are docking proposals \( \{ P_j | j = 1, \ldots, k \} \), where each \( P_j \) is a 4-dimensional vector parameterised by its location \((x,y,z)\), orientation \(\theta\) and a weight \(w_j\).

We have made some environmental assumptions for this work. The ApproachFinder and the shared controller are designed for indoor environments with flat ground planes and have not been tested in environments with split-level floors or ramps. We also inherit assumptions from our training dataset [56]: the environments are well-lit and the point clouds are acquired from roughly 1.6 meters above the ground plane.
3.1 Approach

Figure 3.1 summarises the different steps involved in this vision pipeline. The following subsections elaborate on these steps in detail.

Figure 3.1: ApproachFinder-CV: The input point cloud (a) is processed by a pre-trained network, Votenet, to find candidate objects using bounding boxes (b). The point cloud inside these bounding boxes is used to find planar surfaces. Next, the vision pipeline approximates a convex polygon to cover all the edges of such planar surfaces (c). We sample potential docking locations around these convex polygon edges (d) and filter them based on their visibility and position (e). Finally, these filtered locations are combined to generate a desirability costmap that can be integrated with any real-time planner (f).
3.1.1 Candidate Object Extraction

To identify candidate objects, we use a deep network that consumes a point cloud and estimates semantic classes, objectness scores, and 3D oriented bounding boxes for the objects in the scene. Specifically, we use Votenet[44] to detect tables and toilets in real-time owing to its simple design, compact model size, and high efficiency in detecting indoor objects with partial scans. Votenet performs fast inferences by using just geometric information without relying on color images, which makes it invariant to image distortions.

Votenet is an end-to-end trainable network that draws inspiration from Hough voting by shifting the points on the object surface to its center through a voting module. Votenet uses PointNet++ [42] for subsampling the object points and for feature learning. Then the voting module generates “votes” in the form of a vector offset for each sub-sampled point that goes from the point to the predicted center of the corresponding object. By voting in this way, vote clusters form near object centers and can be aggregated to give 3D bounding box proposals. In a post-processing step, non-maximal suppression is applied to remove redundant proposals. Figure 3.2 shows the detections generated by Votenet on simulated and SUN RGB-D[58] point clouds. Only confident predictions based on objectness score above a threshold are passed to the next step in the pipeline. After preliminary experiments with our target semantic classes of tables and toilets, we selected 0.85 as the threshold which would not yield too many overlapping detections.

Algorithm 1: Merge Overlapping Bounding Boxes

Result: \([PCD_k] = \text{List of extracted and merged point clouds for } k \text{ objects in a scene}\)

1. \([PCD_i] = \text{extractPointCloud}(BB_i)\);
2. for each \(BB_i\) do
   3. \([BB_j] \leftarrow \text{Find overlapping bounding for } BB_i\);
4. end
5. \(G(V,E) \leftarrow \text{CreateGraph}(BB_i, [BB_j])\);
6. \([CC_k] \leftarrow \text{FindConnectedComponents}(G(V,E))\);
7. \([PCD_k] \leftarrow \text{MergeConnectedComponents}([CC_k], [PCD_i])\)

For tables, Votenet often predicts multiple smaller bounding boxes covering...
Figure 3.2: (a) Votenet bounding box detections in simulation; (b) Votenet detections on samples taken from SUN RGB-D dataset

different parts of a single table (see Fig. 3.2a). For this reason, multiple overlapping bounding boxes are merged to capture the entire object. Algorithm 1 summarizes this merge procedure. Line 1 returns the point cloud of the objects that are detected by Votenet. Here we use a convex hull filter to check if the point lies inside a predicted 3D bounding box. Line 2-4 returns a list of overlapping bounding boxes by finding the number of common points from two different bounding boxes. Line 5 generates a graph where each vertex is a bounding box index and there exists a bi-directional edge if two bounding boxes overlap. Line 6 finds the connected components of this undirected graph. Finally, overlapping bounding boxes are merged and returned in line 7.

For toilets the situation was slightly different: Votenet often generated multiple overlapping bounding boxes, but the largest generally captures all of the points in the toilet. In this case we selected the single bounding box with a largest area in the XY plane rather than merging.
3.1.2 Edge Extraction for Candidate Objects

Docking locations are to be sampled around the table top or the toilet seat. For this, the first task is to find planar surfaces belonging to these object parts. Algorithm 2 consumes a point cloud of a candidate object to find planar surfaces and uses plane inlier points to extract edges around the candidate object.

Algorithm 2: Edge Extraction for Candidate Objects

Result: \([e_i] = \text{list of edges belonging to candidate planer surface}\)

\[
\begin{align*}
1 & \text{ for each } PCD \text{ in } \{PCD_k\} \text{ do} \\
2 & \quad [\text{inliers}] \leftarrow \text{PlaneSegmentation}(PCD); \\
3 & \quad [\text{inliers}'] \leftarrow \text{NoiseAndOutlierRemoval}([\text{inliers}]); \\
4 & \quad [\text{clusters}] \leftarrow \text{RegionGrowing}([\text{inliers}']); \\
5 & \quad [\text{obj\_top}] \leftarrow \text{MergeClusters}([\text{clusters}']); \\
6 & \quad [e_i] \leftarrow \text{MinimumAreaConvexPolygon}([\text{obj\_top}]); \\
7 & \text{end}
\end{align*}
\]

In Algorithm 2, line 2 uses RANSAC to search for all planar surfaces that satisfy the following conditions: (i) they are parallel to the ground plane; (ii) their height is at least 0.3 m above the ground plane; and (iii) their point cloud size is larger than 0.5K points. Plane inliers are processed for noise using a statistical outlier filter in line 3. This step cleans the point cloud data that is favourable for the next step in the pipeline. In some cases, Votenet fails to fully capture the entire surface of the object (refer Fig 3.2b). Region growing segmentation is used to expand and capture all the points on the candidate object’s surface that might be missed by Votenet’s bounding box. The purpose of this segmentation algorithm is to merge the points that are close enough in terms of a smoothness constraint. The output of this algorithm is a set of clusters, where each cluster is a set of points that are considered to be a part of the same smooth surface. In step 5, we combine these clusters to capture all the points of the candidate object. Finally, all of the points are projected onto a 2D plane and a minimum area convex polygon is approximated over these projected points. Unlike previous approaches our method is not limited to rectangular or circular objects, but can handle any convex shaped polygonal structure. The edges of this approximated polygon are returned in line 6. Figure 3.3 shows approximated convex polygon corner points and the segmented...
planar surface of tables in simulation.

Figure 3.3: Detected planar surface and approximated convex polygon. Red points are plane inliers and green spheres represent the vertices of the convex polygon.

3.1.3 Sample and Weight Potential Docking Locations

Algorithm 3 summarizes the procedure to find safe docking locations along each edge of the approximated convex polygon with their respective weights. Potential docking locations are sampled at regular intervals along each edge of the approximated convex polygon at line 2 (see Fig 3.4a). The sequential sampling distance is kept at 10 cm and the perpendicular distance away from each polygon edge is slightly more than the wheelchair width. For corners, we interpolate points between the end-point of one edge and the start-point of the next edge of the new polygon. Sampled locations outside the field of view of the camera should not be processed further. Line 3 filters out locations that are not in the camera frame and passes those that are in the frame to the next step. In line 4, a wheelchair sized 3D oriented bounding box is placed at each sampled point and checked for collisions with nearby objects in the scene. The size of the box is based on a Permobil M300 wheelchair whose frame is $41 \times 24 \times 46$ inches. Figure 3.4b, shows the locations that are in the field-of-view and are collision free. For alignment, we compute the ideal docking orientation as the direction perpendicular to the corresponding convex polygon edge. This approach handles both circular and rectangular objects.

The weight calculation (measuring desirability) for each remaining location is divided into two parts: positional weight and visibility weight. Line 6 calculates the positional weight for each filtered sampled location. The weight is kept low if the
sampled location is near to a corner or an object (like chair). The weight gradually increases as we move away from corners or other objects. Visibility weights are based on the volume of the wheelchair bounding box from the filtering step that is visible. Placements that are behind the table are given a lower weight value (less visible) and placements where the wheelchair is completely visible (e.g., along a table’s front edge) are given higher weights. Line 7 calculates the visibility weights and line 8 combines positional and visibility weights to give a desirability weight for each filtered location. A similar strategy is followed for the weight calculation for docking locations near toilets.

**Figure 3.4:** Docking locations generated by ApproachFinder-CV with corresponding headings.
Algorithm 3: Find docking locations and corresponding weights

Result: \([loc_j, W_j, \theta_j] = \text{List of desirable locations with weights and heading}\)

\begin{enumerate}
\item \[\text{for } e \in [e_i] \text{ do}\]
\item \[\text{SampleLocationsAlongEdge}(e)\];
\item \[\text{FoVFilter}(loc)\];
\item \[\text{WheelchairFit}(loc)\];
\item \[\text{for } l_j \in [loc] \text{ do}\]
\item \[W_{pos} \leftarrow \text{PositionalWeights}(l_j)\];
\item \[W_{vis} \leftarrow \text{VisibilityWeights}(l_j)\];
\item \[W_j \leftarrow W_{pos} \ast W_{vis}\];
\item \[\theta_j \leftarrow \text{direction} \perp e;\]
\item \end{enumerate}

3.2 Desirability Costmaps

An elegant way to process these docking locations is to encode them as a costmap that can be easily integrated with a real-time path planner. For this, every point on the ground plane should have a desirability value which gradually increases as we move towards a docking location. For each docking spot, parameterized by location and orientation - \((x_i, y_i, z_i, \theta_i)\), we fit a tri-variate Gaussian distribution that is centred at \((x_i, y_i, \theta_i)\). The co-variance matrix in each case is a diagonal matrix, \(\text{diag}(\sigma_{xx}, \sigma_{yy}, \sigma_{\theta\theta})\), that measures the spread of this distribution. In the implementation, the XY spread is fixed to 4 meters in both directions and the heading spread is kept at 30 degrees.

This distribution is continuous in the XY \(\theta\)-plane so that each conceivable interval on the ground plane has some desirability other than zero. The properties of normal distribution ensure most of the desirability is close to the mean (docking location) and it gradually decreases as we move further away. Also, self-selection of co-variance matrix enables us to decide the spread of this desirability curve. It should be noted that we made no attempt to optimize the basis functions used to construct these costmaps, and other Gaussian parameters or other functions—for example, something with bounded support—could be substituted.
For a point $p(x, y, z, \theta)$, instantaneous desirability is calculated by:

$$F_{des} = \frac{\sum_{i=1}^{n} N_i (\mu_i, \Sigma_i) \cdot pdf(p) \cdot W_i}{N}$$

(3.1)

where $n$ is the total number of parking locations found and $\mu_i, \Sigma_i$ and $W_i$ are the center, co-variance matrix and weight corresponding to each docking location, respectively. $N$ is a normalisation term which limits the desirability between $[0,1]$ and $pdf$ is the probability density function for the Gaussian distribution centered at each docking location.

To find the temporal desirability of a point, we combine $F_{des}$ over time. To achieve this, we draw inspiration from occupancy grid maps and replace occupancy with desirability value. Problems of this class estimate a fixed quantity in the environment from a sequence of sensor measurements. The expression for temporal desirability of a point $p$ at time $t$, $l_t(p)$, with instantaneous desirability $F_{des}$ is given by:

$$l_t(p) = \log \frac{F_{des}}{1 - F_{des}} - l_0(F_{des}) + l_{t-1}(F_{des})$$

(3.2)

where $l_0$ is prior in log-odds form. For our case, $l_0 = 0$ which means initially each grid cell is equally likely to be desirable or undesirable. Figure 3.1f shows the temporal desirability grid map for a table using Eq. 3.2.

### 3.3 Implementation Details

**Simulation Environment**: Gazebo [29] (version 9) offers the ability to accurately and efficiently model robots in complex indoor environments. The pipeline is tested in multiple simulation environments with furniture models taken from the 3DGEMS [47] library. Figure 3.5 shows two such environments.

**Robot**: A custom designed robot model was used as a stand-in for the wheelchair. This robot was equipped with a KinectV2 camera to capture color images and point cloud data. It was controlled through user commands given by a joystick. The perception and control modules communicate through the Robot Operating System (ROS) [46] platform (Melodic distribution). Most wheelchairs use two wheel differential drive, with two or four casters for balance (in front, rear,
Figure 3.5: (a) Large and (b) small office environment in Simulation
and mid-drive configurations). Our model uses Gazebo’s differential drive plugin to move the back wheels of the robot, which is roughly equivalent to a rear-drive powered wheelchair configuration. The plugin accepts two velocity commands (linear and angular) and publishes odometry information. Figure 3.6 shows the robot and the data captured by the simulated camera.

![Figure 3.6: Robot in the Gazebo simulation with image and point cloud captured by the camera.](image)

**Pre-processing**: We supply time-synchronised depth image and point cloud data to the pipeline for pre-processing (any RGB data from the sensor is ignored). First, we transform the point cloud from the camera depth optical frame to the world frame using ROS tf2 transformations. We filter out points whose depth value is NaN and randomly downsample to 40K points that are passed to Votenet.

**Votenet**: The object detection network, Votenet, is trained on the SUNRGB-D V2 dataset with 40K input points. We employ the same network architecture as used in the original paper and use the same inference strategy to filter out confident proposals [44].

**Docking Locations**: Algorithms 1–3 are written in C++ and make extensive use of the Point Cloud Library [50]. Custom ROS messages are used to transfer information between nodes in the pipeline. Figure 3.7 provides these details in depth. Finally, visualisations are done with ROS-Rviz and the Open3D [72] library.
Figure 3.7: Summary for ApproachFinder-CV algorithm. Each module (in blue) is further elaborated with the associated steps. On top, we provide the message information that is passed through ROS.

3.4 Experimental Results

In this section, we provide qualitative results for our vision pipeline. The algorithm is validated on multiple scenes taken from the SUN RGB-D dataset that contained candidate goal objects (tables and toilets). This dataset has 10355 RGB-D sample images of indoor scenes.

Figure 3.8 provides some example results from this dataset. In each of the sampled cases, the pipeline was able to detect numerous safe docking locations with correct orientation across different viewing angles. It rarely missed potential docking locations (see Fig3.8f). Due to occlusion, points that are farther away or underneath the table are not captured completely and the algorithm considers such empty spaces in the field of view as potential docking locations until it perceives data that indicates otherwise (see Fig. 3.8b). We consider this optimistic treatment of unobserved space as low-risk because when the wheelchair drives towards such docking locations more relevant data will become available and, if that space is obstructed then the algorithm will mark the corresponding docking locations unsafe. In contrast, locations outside the camera’s field of view are not updated; for
example, in Fig. 3.8c, we see how the Field of View (FOV) filter prunes the docking locations in front of the toilet because they are outside the perceivable sensor area.

We are not able to provide any qualitative comparison with other approaches because other approaches tend to simplify the problem by using fiducial markers [35, 62] or customised docking stations [49]. The only comparable approach that we were able to locate was given by Jain et al. [25]. This approach finds docking locations around tables with proper pose-alignment but is limited to rectangular or circular tables and is evaluated on a series of self collected data. In comparison, our approach can be applied to any table configuration and is qualitatively evaluated on a large scale indoor dataset.
Figure 3.8: ApproachFinder-CV results on SUN RGB-D dataset. Cyan spheres represent the docking locations and grey arrows represent the ideal orientation.
Our computer vision pipeline provides robust results for various input scenes but suffers from slow performance. The execution time depends on the number of candidate objects found in the scene. On average, it takes roughly 1.25 seconds to find docking locations for a candidate object. The bottleneck lies in iterating over the scene point cloud (≥ 40K points) multiple times at various steps in the pipeline. For example, in Algorithm 3, each sampled location is checked for collision against the entire scene point cloud (in the WheelchairFit function). We could parallelize such checks, but we believe a better solution to this problem is through a vision-based computationally-efficient end-to-end neural network. Therefore, we propose to use the data from our computer vision pipeline (generated offline where speed is less of an issue) to train such a network.

In this chapter, we first provide motivation for using a voting based approach in our neural network and then we talk about different modules that make our neural network (Sec. 4.1). Next, we talk about network related implementation details (Sec. 4.2). We then provide results on the SUN RGB-D dataset (Sec. 4.3). This is followed by an in-depth analysis (Sec. 4.4) where we conducted multiple experiments to better choose our pipeline architecture, such as layers in our backbone network, importance of voting, different sampling techniques and other aspects of the network that can help us improve performance.
4.1 Approach

Point clouds are challenging data structures for neural net based approaches due to the following properties:

- **Varying density**: The density and other attributes are non-uniform across point clouds which makes them unsuitable for traditional CNN based approaches. A standard CNN requires regular data to perform weight sharing and kernel optimisations.

- **Permutation Invariance**: Images have a well defined array-like structure while point clouds are unordered. In other words, a network that consumes a point cloud of size \( N \) should be invariant to its \( N! \) permutations.

- **Nearby points**: A point and its neighbours provide a meaningful subset. The model should not only capture local features but should also encode relationships amongst nearby points.

Pioneer work in this direction is PointNet [41]. The basic architecture involves spatial feature learning through a shared Multi-Layer Perceptron (MLP) and then applies a single symmetric aggregation function, max-pooling, to capture the global feature. However, this network cannot to capture local context that can provide valuable cues for the overall generalisability of the networks in unseen environments. In contrast, CNN based architectures accumulate multi-level features to produce confident predictions.

To solve this, PointNet++ [42] learns features in a hierarchical fashion. Similar to a CNN, it partitions the space into smaller overlapping regions and extracts local features from a small neighbourhood. Such local features are aggregated into larger features to build a hierarchy. This multi-level design enables the architecture to exploit both the local and global structures of the given sample. It uses PointNet to abstract local level features that are passed to subsequent higher levels. To overcome varying density, this work proposes two grouping strategies: Multi-Scale Grouping (MSG) and Multi-Resolution Grouping (MRG). MSG captures local contexts from the same hierarchical layer while MRG enables feature learning from previous multi-layers as well. Each level in this hierarchy is called
a set abstraction layer and combines sampling, grouping, and PointNet network layers.

ApproachFinder-NN leverages the PointNet++ design for hierarchical feature learning. This is an end-to-end trainable network that consumes a point cloud and produces multiple docking locations with a desirability weight corresponding to each. Inspired by traditional Hough voting this network uses deep features to vote for such docking locations.

The rest of the section is organised as follows. First, we draw similarities between our docking problem and general problems that can be solved using voting techniques. Next, we describe ApproachFinder-NN which is divided into three modules: a backbone network used to extract features; a voting module for Hough voting; and a proposal network to generate docking proposals. Figure 4.1 illustrates the output from these network modules.

**Figure 4.1**: ApproachFinder-NN: The input object point cloud is sub-sampled using PointNet++. These sub-sampled points are called seed points. Each seed point votes for its nearest docking location. These votes are aggregated into clusters and are further processed by the proposal module to generate docking locations.

### 4.1.1 Hough Voting

The Hough Transform is a well-known technique in computer vision and was initially used to detect object instances of certain shapes, including lines, circles and ellipses in images. The basic idea behind the Hough technique is very simple: each input measurement provides a contribution towards a globally consistent solution. Extracted image gradients that convey information about edges and object boundaries are combined as votes in the parameter space and used to discover shape instances in the image. The advantages of this technique are robustness to image
noise and tolerance to partially occluded or deformed shapes. This technique, due to its simple structure, is used across multiple domains in computer vision. Examples include plane extraction from 3D point clouds [7], 6D pose estimation [58] and implicit shape modeling [31].

Traditional Hough voting detectors can be divided into two steps: online and offline. In the offline step, a codebook is created to store the mapping between encoded features and their corresponding proposals using offsets values. During online inference, the extracted features from an image are compared against the codebook features and votes are generated using retrieved offsets. Under the assumption that features tend to vote in agreement, vote clusters form near objects of interest and these clusters are aggregated to generate proposals.

This approach is well suited for our problem for two reasons. First, point clouds are sparse sets where traditional region-based approaches fail [48]. Such voting based approaches are more suitable for such sparse sets as they can generate proposals from varying point densities. Second, the Hough voting technique is a bottom-up approach that builds from partial information at multiple levels to generate confident proposals. This is particularly beneficial for point-based methods, where local features can play an important role in generating accurate proposals. Although similar results can be obtained by varying the receptive field of a neural network, in the results sections we show experimentally why this voting is advantageous for such sparse point sets.

4.1.2 Backbone Network: PointNet++

The backbone network consumes a point cloud of $N$ points and generates a set of $K$ 3D coordinates augmented with a feature vector. Rather than relying on a set of handcrafted features that are not generalisable and are difficult to formulate, we leverage PointNet+ [42] for the task of feature learning. This feature learning is not restricted to a PointNet++ based backbone, but we adopt this network for its simplicity and good performance on related point-based tasks, like object localisation [43, 44] and semantic segmentation [30].

This network contains multiple set abstraction layers, each with three steps: sampling, grouping and feature learning. At each level, this abstraction layer di-
vides the point set into local regions and learns deep features on these groupings to produce higher level features.

**Sampling Layer**: Farthest Point Sampling (FPS) is used to select a subset of \(N'\) points from the \(N\) points provided. FPS provides better coverage of a given point set as compared to other sampling techniques (such as random sampling) because it iteratively selects the next candidate that is farthest away from the rest of the points already in the group.

**Grouping Layer**: Each of the \(N\) points in the point set is associated to one of the sub-sampled \(N'\) points found in the previous layer. These groupings form smaller local regions that are used to learn features in the next layer. For grouping, a ball query is used that finds all the points within a fixed radius.

**Feature Learning Layer**: On each of these \(N'\) local regions, we apply PointNet to extract features. The output of this layer is a set of \(N'\) points that are augmented with features that capture local region properties.

PointNet++ also has a set of feature propagation layers that upsample the point features (interpolation by taking the Euclidean distance weighted average of its nearest three neighbours). It also combines skip-linked features through an MLP (interpolated features and skip-linked features are concatenated before they are fed into the MLP). In summary, PointNet++ has several such set-abstraction layers and upsampling (feature propagation) layers that generate \(K\) seed points each of dimension \((3 + C)\).

### 4.1.3 Voting Module

Unlike traditional Hough voting methods that use a codebook [31], we use a shared MLP with fully connected layers, ReLU activation, and batch normalisation for the task of vote generation.

Specifically, the voting layer takes seed points \(\{s_i\}_{i=1}^{K}\), where each seed \(s_i = [x_i; f_i]\) is a vector \(x_i \in \mathbb{R}^3\) concatenated with feature \(f_i \in \mathbb{R}^C\), and maps its features to a vote using this shared MLP. The vote offsets are composed of XYZ offsets \(\Delta x_i \in \mathbb{R}^3\) and feature residuals \(\Delta f_i \in \mathbb{R}^C\) such that the vote \(v_i = [x'_i; f'_i]\) for a seed \(s_i\) is given by \(x'_i = x_i + \Delta x_i\) and \(f'_i = f_i + \Delta f_i\).

This MLP (shared by all seed points) saves us from an expensive codebook
lookup and can be easily trained with the rest of the pipeline. The vote loss used for training is given by:

$$L_{\text{vote}} = \frac{1}{N_{\text{pos}}} \sum \| \Delta x_i - \Delta x_i^* \| 1 \text{[s\_i is near a docking spot]}$$  (4.1)

where $\Delta x_i^*$ is the ground truth residual of the seed point $s_i$ from its closest docking location (from the training data provided by ApproachFinder-CV), the indicator function only includes a seed point $s_i$ if it is close to a docking location ($\leq 1m$), and $N_{\text{pos}}$ is the total number of such nearby seed points. Seed points far from any docking location are entirely ignored.

The generated votes are virtual points that have the same dimension as their corresponding seed points. Votes differ from their seed points in the Euclidean space as they are targeted to reach the docking locations around candidate objects. As a result, vote clusters emerge near these docking locations. These clusters are aggregated using another shared MLP to generate confident docking pose proposals and weights.

### 4.1.4 Vote Aggregation

In vote aggregation, we use clustering to filter out low quality votes and to generate the proposals from aggregated clusters. We group clusters based on their 3D geometric distance. From the set of votes $\{v_i = [x'_i; f'_i] \in \mathbb{R}^{3+C}\}_{i=1}^K$, we sub-sample a set of $M$ votes using FPS in the Euclidean space. Next, we find the association of the remaining votes to these $M$ groups by using a simple ball query: $C_k = \{v_i \in C_k \mid \| x'_i - x_{C_k} \| \leq r \}$, where $r$ is the ball radius, $x_{C_k}$ is the centroid of the $k^{th}$ group, and $k = 1 \ldots M$. Compared to kNN, a ball query avoids high search costs for finding the neighbours and ensures a fixed scale for all regions, thus making the method more generalised across space.

We realise vote aggregation through a set abstraction. Specifically, each vote location is normalised into its local cluster coordinate in euclidean space and is passed through an MLP before being max-pooled channel-wise to generate one global feature vector for the whole cluster. These aggregated $M$ vote clusters are passed through a proposal module to generate proposals.
4.1.5 Proposal Module

The cluster features from the previous step are passed through another shared MLP to generate one proposal per aggregated cluster. Each proposal is a multi-dimensional vector comprised of a docking location center, dockness score, and weights corresponding to each discretized heading direction.

These three components of the a proposal provide the remaining three components of the training loss function:

- **Dockness Loss** is inspired by the objectness score generated by Votenet. At inference, this is used to remove less confident docking locations. We supervise the dockness scores, via cross entropy loss, for two vote categories that are either located close ($\leq 0.2m$) or far ($\geq 0.5m$) from any ground truth docking center. We consider these as positive and negative proposals, respectively.

- **Center Loss** is supervised for the positive proposals for center prediction using Chamfer loss[16]. Chamfer loss computes the shortest distance of one point set to another point set in both directions. This is a two-way loss which encourages each seed point to generate a proposal near a ground truth docking location and each ground truth docking location to have a nearby proposal.

- **Heading Weight Loss** is supervised as a regression loss for each heading bin. We divide the heading into 12 equal-sized bins (each covering $30^\circ$) and calculate the desirability corresponding to each heading bin (refer to Eq. 3.1 in Ch. 3). In this regression, we use smooth-$L_1$ loss.

Thus, the loss function for training ApproachFinder-NN is a multi-task loss function given by:

$$Loss = c_1 L_{vote} + c_2 L_{dockness} + c_3 L_{center} + c_4 L_{heading-weight},$$

(4.2)

where the $c_i$ scalars are chosen to keep the magnitude of the individual loss functions balanced.
4.2 Implementation Details

The following subsections describe network specific implementation details.

**Input and data augmentation:** The backbone network takes a randomly sampled subset of 2K object points as input from the SUNRGB-D [56] dataset. We use the ground truth 3D oriented bounding box to extract the object points. Each geometric point (XYZ) is appended with a height feature that is the distance of the point from the floor, where the height of the floor is approximated as the 0.99 percentile of the point cloud height.

Data augmentation plays an important role in preventing model over-fitting. We follow three strategies to augment the training data: (i) randomly flip the point cloud along the YZ-plane (horizontally); (ii) randomly rotate the point cloud along the upright Z-axis in a uniform range [-30, +30]; and (iii) randomly scale the points between [0.85, 1.15]. For each of these augmentations, we modify the ground-truth data generated by ApproachFinder-CV as well.

**Architecture Details:** With an input of \( N \times 4 \) where \( N = 2K \) points, the output of the backbone network is a set of \( K \) seed points each of dimension \((3 + C)\) where \( K = 512 \) and \( C = 256 \). We use PointNet++ [42] with three set abstraction layers and one feature propagation layer. We sub-sample N points through three abstraction layers to 1024, 512 and 256 points respectively. The feature propagation layer up-samples from 256 points to 512 points. Table 4.1 provides a summary of these backbone network layers. Each set abstraction is specified by \((n, r, [f_1, f_2, f_3])\), where \( n \) is the number of sub-sampled points using FPS, \( r \) is the ball-query radius, and each \( f_i \) is the output of the \( i^{th} \) layer of an MLP for point feature transformation. The feature propagation layer up-samples by interpolating with nearby points (three nearest points), and it concatenates skip-linked features before passing them through an MLP. Each feature propagation is specified by \([f_1, f_2]\), where each \( f_i \) is the output of the \( i^{th} \) layer.

The voting module is an MLP that maps seed points to votes with fully-connected output sizes of \((256, 256, 3+256)\). Here, the last fully connected layer gives both the XYZ offsets and feature residuals. We use ReLU activation with batch normalisation in every layer excluding the last.

Vote aggregation is realised through a set abstraction layer that consumes votes
to give clusters. The number of points in FPS is set equal to the number of proposals. In the implementation, the number of proposals is set to 64, a number which was chosen based on the 0.90 percentile of the number of docking locations found per object in the SUN RGB-D dataset by ApproachFinder-CV.

Finally, another MLP is used to generate docking proposals of shape \((3 + 2 + NHW)\) from the votes. The output dimension consists of 3 center regression values, 2 dockness scores (positive and negative class) and \(NHW = 12\) regression weights (one for each heading bin).

**Training:** We train the deep network using an Adam optimiser with a starting learning rate of 0.001 and a step-wise decay (10x) after 80, 120, and 160 epochs. After 180 epochs we do not see much performance improvement. For all of the trainable layers except the regressions, we use batch normalisation with a decay rate of 0.5 initially, shifting gradually to 0.99. The batch size for training was set to 16.

**Inference and post-processing:** Proposals generated by the network are post-processed by weight and confidence filters that remove docking locations with low dockness confidence or low overall weight.

### 4.3 Results

Experiments were carried out on a PC with an Intel Core i9-9700K processor with 64GB RAM, running Ubuntu 18.04 LTS. The model was trained on an NVIDIA RTX 2070S GPU with 8GB VRAM.

**Benchmark Dataset.** A large number of datasets have been collected to evaluate network performance. For the purpose of our work, we use SUN RGB-D[56] to evaluate ApproachFinder-NN. This is a single-view RGB-D dataset for 3D scene
understanding. It has 10335 RGB-D images (divided roughly equally between training and test), with 64,595 3D bounding box annotations for 37 object categories. Out of these 10K images, 3784 and 1159 are novel and captured by Kinect V2 and Intel Real Sense cameras, respectively, while 1449 images are from the NYU-Depth V2 [52], 3389 images from the SUN3D [65] and 554 images from the B3DO [26] datasets. We selected SUN RGB-D because the images from this dataset closely resemble the partial scans which a camera mounted on a wheelchair would capture.

**Groundtruth Generation.** We select point clouds for tables and toilets with more than 2K points from this dataset and generate training labels by running the ApproachFinder-CV pipeline on such candidate objects. For each training sample, we store the object point cloud, docking locations, orientation, and the Euclidean vector offset from each object point to its nearest docking location.

**Evaluation Metrics.** We evaluate our trained network using two metrics: 3D bounding box tightness and instantaneous desirability.

The 3D bounding boxes are parameterised by size \((h, w, l)\), center \((cx, cy, cz)\), and orientation \(\theta\) around the z-axis. The size of the oriented bounding box corresponds to the dimensions of our nominal powered wheelchair (Permobil-M300) that is translated and rotated to the predicted docking location. We compare model predictions at two different 3D IoU values \((0.25, 0.5)\) and report the average precision and recall values for these thresholds.

From these docking locations, we generate instantaneous desirability cost maps according to Eq. 3.1. We use mean-absolute error between ground truth and predicted desirability cost map as one of our evaluation metrics (approximated by the summing samples over a 3D grid).

**Quantitative results.** The trained model has a recall value of 75.66 and 72.94 at 0.25 and 0.5 IoU respectively. The average precision value was 36.22 and 32.46 at 0.25 and 0.5 IoU respectively. The per-grid cell absolute instantaneous desirability error was 0.018326. ApproachFinder-NN has a high true positive rate which is beneficial for detecting multiple docking spots. Since the neural net produces multiple proposals it is possible to have misidentified parking spots. For this, we apply post-processing steps to filter out less confident proposals.

**Qualitative results.** In Figures 4.2 and 4.3, we compare detection results from
the two pipelines on the SUNRGB-D dataset. The NN pipeline shows promising detection results, despite the omission of portions of the tables from the point cloud in some cases. There are some limitations to our method. Our network produces approximate headings for parking spots due to the discretization of heading. In some cases, the predicted heading is off by ±1 bin (see Fig. 4.2d), which for 30° bins can be significant. Also, the density of sampled parking spots for the neural net is not uniform (see Fig. 4.2h). This uneven behaviour usually happens when there are insufficient seed points and hence votes. However, the strength of our approach is that it can still give detection results from partial scans, and mobile depth cameras in natural environments often produce such partial scans.

**Model Details.** The ApproachFinder-NN model has an inference time of 0.08s seconds on average, with a model size of 11MB. This is 15x faster than the ApproachFinder-CV pipeline (inference time of 1.25s). With fast inference, low resource requirements, and a small model size, ApproachFinder-NN appears suitable to implement on a smart wheelchair platform.

### 4.4 Analysis Experiments

In this section we present results from some exploratory experiments that were performed to choose features of the final ApproachFinder-NN architecture described above.

**Backbone Network:** ApproachFinder-NN is designed to take advantage of deep features to learn local context; consequently, features from different layers in PointNet++ should affect the network performance. In order to explore this effect, we vary the number of set-abstraction (SA) and feature propagation (FP) layers. Figure 4.4 presents these results. We start from only one set abstraction layer (SA1) and incrementally increase contextual learning till five set abstraction layers and three feature propagation layers. Voting using three set abstraction and one feature propagation layer gives the best network performance.

**Voting Importance:** ApproachFinder-NN is compared against a baseline network, BApproachFinderNN, which directly proposes docking locations, orientations and weights from sub-sampled seed points. In other words, BApproachFinderNN has the same backbone and proposal network architecture, but skips the vot-
Figure 4.2: Comparison between ApproachFinder-CV (left column) with ApproachFinder-NN (right column)
Figure 4.3: Additional qualitative results for the two pipelines.
Figure 4.4: Effect of increasing seed layers in PointNet++

ing step and directly generates proposals. To do so, it removes the voting module and uses seed points for clustering. The loss function is similar to the original implementation, but lacks the vote-regression loss:

\[
\text{Loss} = c_2 L_{\text{dockness}} + c_3 L_{\text{center}} + c_4 L_{\text{heading–weight}},
\]

(4.3)

where the constants \(c_i\)'s keep the losses in same magnitude.

Table 4.2 provides a comparison between the design with and without votes. Clearly, we gain significantly (∼18AP) by adding the voting module. Since docking locations are far away from actual seed points, the proposals generated by them have low confidence. Voting brings these far-away low confidence points closer, thereby forming confident clusters that lead to confident proposals. Voting also provides better coverage to predicted docking locations, which are predicted from aggregated votes.

<table>
<thead>
<tr>
<th></th>
<th>AP</th>
<th>Recall</th>
</tr>
</thead>
<tbody>
<tr>
<td>BApproachFinderNN</td>
<td>18.98</td>
<td>58.05</td>
</tr>
<tr>
<td>ApproachFinder-NN</td>
<td>36.22</td>
<td>75.66</td>
</tr>
</tbody>
</table>

Table 4.2: ApproachFinder-NN vs BApproachFinderNN: Baseline mode proposes directly form seed points. Average precision (AP) and Recall are at 0.25 IoU.
Vote aggregation: We analyse how changing the aggregation function or radius affects network performance. First, we try different aggregation techniques to combine the votes. Specifically, we test three types of aggregations schemes: max, average and RBF weighting (distance-based). All these functions are symmetric and invariant to point permutations. Table 4.3 shows max-pooling performed the best. This result is not unexpected, as PointNet uses the same aggregation function.

Second, we vary the clustering radius and find the best performance at 0.3m (see Fig. 4.5). Increasing the aggregation radius beyond that appears to add clutter votes (noise) which decreases network performance.

Proposal Sampling: In the vote aggregation step, we select $M$ proposals by sampling over votes. Table 5 summarises the effect of using different sampling techniques. The algorithm described above uses FPS on votes, but here we compare against two additional sampling strategies: random sampling and seed-FPS. In random sampling, we randomly select 64 votes while in seed-FPS we sample from

<table>
<thead>
<tr>
<th>Aggregation Function</th>
<th>AP@0.25</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average Pooling</td>
<td>34.98</td>
</tr>
<tr>
<td>RBF Pooling</td>
<td>34.32</td>
</tr>
<tr>
<td>Max Pooling</td>
<td>36.22</td>
</tr>
</tbody>
</table>

Table 4.3: Results of different symmetric pooling functions

Figure 4.5: Effect of increasing vote aggregation radius
seed points but still use the corresponding votes for further processing.

<table>
<thead>
<tr>
<th>Sampling Strategy</th>
<th>AP@0.25</th>
</tr>
</thead>
<tbody>
<tr>
<td>FPS on seeds</td>
<td>31.19</td>
</tr>
<tr>
<td>FPS on votes</td>
<td>36.22</td>
</tr>
<tr>
<td>Random sampling</td>
<td>24.68</td>
</tr>
</tbody>
</table>

**Table 4.4: Effect of different sampling strategy**

**Input Features:** We test three options for the input data provided to the backbone network: XYZ, XYZ with height, and XYZ-RGB with height. We wanted to explore if the backbone network can learn useful local context from color or height values. Table 4.5 summaries these results. Adding just the height feature gave the best model performance.

<table>
<thead>
<tr>
<th>Input Features</th>
<th>AP@0.25</th>
</tr>
</thead>
<tbody>
<tr>
<td>XYZ</td>
<td>34</td>
</tr>
<tr>
<td>XYZ+height</td>
<td>36.22</td>
</tr>
<tr>
<td>XYZ+RGB+height</td>
<td>31.1</td>
</tr>
</tbody>
</table>

**Table 4.5: Effect of varying input features**
Chapter 5

Shared Control for Wheelchairs

A major challenge in designing a user-friendly shared controller for wheelchairs is to find the right balance between operator and robot control. Such systems should support goal achieving behaviour while avoiding over-dependence. Viswanathan et al. [61] conducted a Wizard of Oz study in which they found users preferred shared-control over full autonomy. Wheelchair operators suggested the use of joystick or voice commands to convey their intent to the shared control system. Based on these observations, we have created a shared controller that provides safe maneuvers while considering user intention; however, the system described in this chapter should be considered a proof-of-concept demonstration of how the docking locations identified by the pipelines described in the previous chapters could be incorporated into a shared control system. As a proof-of-concept, we do not claim it to be the only way of doing so, or that it represents a rigorously designed and analyzed contribution.

This chapter is divided into the following sections: First, we briefly discuss the essential properties for a shared-control wheelchair system (Sec. 5.1). Next, we describe the system we built using an MPC based framework (Sec. 5.2). This followed by implementation details in Sec. 5.3. Finally, we conclude by providing navigation results in simulation (Sec. 5.4).
5.1 Properties for a shared-control wheelchair system

We wanted to develop a shared control system for two main reasons: (i) this would help older adults and individuals with cognitive impairment; (ii) shared control includes promotes user engagement which is an important in the case of system malfunction and also maintain user’s interest.

Next, we describe the properties that our shared controller should follow.

- **Dynamic obstacle avoidance:** The controller should sense dynamic obstacles and update its belief when the environment changes. The generated trajectories should be collision free.

- **User intent:** The system should allow the user to assert control over the wheelchair.

- **Smooth paths:** Generated paths should avoid jerks and large slip angles for safe motion.

- **Goal oriented:** The shared control should promote goal achieving behaviour and must provide assistance in docking the wheelchair near candidate objects.

5.2 Model Predictive Controller Design

The shared control signal is synthesized by a closed-loop, receding horizon MPC whose architecture is summarized in Figure 5.1. First, an optimal control sequence is computed and the first step from this sequence is executed. Then the system receives state feedback from the sensors, and re-optimises the control sequence using the unexecuted portion of the previous plan as a warm start. For the MPC algorithm we use a modification of the AutoRally codebase [20], a derivative-free, sample-based approach that can achieve high frequency updates through parallelism on a GPU. The behaviour of the MPC is determined by the dynamics model and cost functions we have designed.
Figure 5.1: An MPC based method that uses a system dynamics model and task oriented cost functions to generate control plans. These control plans are optimised in real-time on a Graphical Processing Unit (GPU).

5.2.1 Dynamics Model

Although it was designed for aggressive, high-speed, outdoor, dirt-track driving maneuvers by a 1/5 scale RC truck chassis and hence is overkill for low-speed indoor wheelchair navigation, for implementation convenience we use the same dynamics model as implemented in the original work [63]. The state space of the robot is modelled by seven state variables: \((p_x, p_y, \theta, r, v_x, v_y, \dot{\theta})\). Here \(p_x\) and \(p_y\) are XY positions, \(\theta\) is heading, \(r\) is roll, \((v_x, v_y)\) are the body frame forward and sideways velocities respectively and \(\dot{\theta}\) is the heading rate. This system has two control inputs: steering \((u_1)\) and throttle \((u_2)\).

The model partitions the state vector into two groups: kinematic state variables \(x_k\) and dynamic state variables \(x_d\). Then, the complete state is given by:
\[ x = \begin{pmatrix} x_k \\ x_d \end{pmatrix} \]  
(5.1)

where \( x_k = (p_x, p_y, \theta)^T \) and \( x_d = (r, v_x, v_y, \dot{\theta})^T \).

The kinematics update equation is given by:

\[ x_k(t+1) = x_k(t) + k(x(t))\Delta t \]  
(5.2)

where \( k(x) \) is defined as:

\[ k(x) = \begin{pmatrix} \cos(\theta)v_x - \sin(\theta)v_y \\ \sin(\theta)v_x + \cos(\theta)v_y \\ \dot{\theta} \end{pmatrix} \]  
(5.3)

The dynamic equation of motion is given by:

\[ x_d(t+1) = x_d(t) + f(x_d, v(t))\Delta t \]  
(5.4)

where \( v(t) \) the perturbed control input given by:

\[ v(t) = \begin{pmatrix} u_1(t) + \varepsilon_1(t) \\ u_2(t) + \varepsilon(t) \end{pmatrix} \]  
(5.5)

From the above set of equations, the system dynamics can be expressed as:

\[ x(t+1) = \begin{pmatrix} x_k(t) \\ x_d(t) \end{pmatrix} + \begin{pmatrix} k(x(t)) \\ f(x_d(t), v(t)) \end{pmatrix} \Delta t. \]  
(5.6)

Here, \( f \) is a basis function model that represents the vehicle dynamics. In this case we simply reused the set of 25 empirically derived basis functions included in [20]. A more rigorous implementation would want to substitute a physically derived or learned model of a wheelchair (for example, see [38]); however, we found that the existing model produced qualitatively reasonable trajectories for a ground robot and so focused our efforts on designing cost functions suitable for shared control goal pursuit (and that could be integrated easily into the codebase).
5.2.2 Cost Functions

We designed the cost functions to model the properties highlighted in Section 5.1. At time \( t \) and given current state \( x_t \), the MPC controller constructs predicted future state trajectories \( x_{[t..t+T]} \) by choosing future inputs \( u_{[t..t+T]} \) over horizon length \( T \) to optimize the trajectory cost function objective:

\[
C_{\text{tra}j}(x_t, u_{[t..t+T]}) = \sum_{s=0}^{T} C_{\text{step}}(x_{t+s}, u_{t+s-1}, u_{t+s}, s)
\]  

(5.7)

where the trajectory evolution is given by (5.6) and the step cost is

\[
C_{\text{step}}(x, u', u, s) = w_1 \text{ObsMap}(p_x, p_y) + w_2 0.9^s I(\text{crash}(x)) + w_3 \text{CostMap}(p_x, p_y, \theta) + w_4 \text{joy}(u, j, s) + w_5 C(u) + w_6 S(u, u')
\]

and

- \( \text{ObsMap}(p_x, p_y) \) is the cost of robot being at position \((p_x, p_y)\). The obstacle cost map is an occupancy grid map with inflation layer.
- \( 0.9^s I(\text{crash}) \) is a time-decaying indicator function that is enabled in crash scenarios. A sampled trajectory that leads to a crash in near future time-steps is given high cost value using this function.
- \( \text{CostMap}(p_x, p_y, \theta) \) is the cost given by the 3D desirability cost map discussed in Section 3.2. We round the heading to the nearest heading bin and extract the XY temporal desirability value.
- \( \text{joy}(u, j, s) \), seeks to keep the planned input close to the user’s joystick signal \( j \) (at time \( t \)). The penalty for deviation decays as \( s \) increases.
- The final two terms \( C(u) \) and \( S(u, u') \) encourage stable, smooth and jerk free trajectories from the given control inputs.

In our implementation, we set weight vector \( w = [300, 1000, 200, 450, 20, 45] \).
5.3 Implementation Details

5.3.1 Mapping
To map the environment, we use a ROS wrapper for OpenSLAM’s Gmapping [21]. Specifically, we use the `slam_gmapping` node that uses laser scan data and odometry information to build a 2D occupancy grid map. For this work, we transform the 3D point cloud data from the simulated Kinect V2 camera to the equivalent of a 2D laser scan using the `pointcloud_to_laserscan` package.

The Gmapping package uses Rao-Blackwellized particle filters to solve the simultaneous localization and mapping (SLAM) problem. It improves on previous work [22] by reducing the number of particles for learning grid maps. It uses both robot movement and sensor observations to calculate proposal distributions along with selective re-sampling to make the algorithm highly efficient.

5.3.2 Dynamic Obstacle Avoidance
We use the ROS `costmap_2d` package to generate the dynamic obstacle costmap. This node uses sensor data and information from a static map to mark and clear the obstacles in a dynamic environment. We use one global costmap for wheelchair navigation. This costmap starts from a static floor plan to mark obstacles initially and consumes sensor data to update its occupancy belief. The package inflates the obstacles by propagating the cost values out from the occupied grid cells to the empty cells while gradually decreasing the cost as the distance to the obstacles increases.

The MPC controller is implemented in the CUDA parallel programming environment and runs on the GPU. For efficient access the occupancy grid is loaded into GPU texture memory. This texture memory is optimised for read-only lookup, which is beneficial for cost functions as they are accessed over a million times per second.

5.3.3 3D Desirability Costmap
The temporal 3D desirability costmap is essentially an occupancy grid, except that larger values are better. In the implementation, we discretize the heading into 12
bins and generate one 2D desirability costmap corresponding to each heading bin. This costmap is also loaded into texture memory for efficient lookup.

5.3.4 Navigation

![Diagram of wheelchair navigation stack]

**Figure 5.2:** Navigation stack for the proposed approach using an MPC controller. Each node in the system is accompanied by a small description that highlights its importance.

Figure 5.2 explains the wheelchair navigation stack. The "Robot Current State" node listens to the Gazebo link states topic and extracts the pose of the robot’s base footprint. This pose is passed as an odometry message to the MPC controller. The "Obstacle Costmap" node publishes a 2D occupancy grid that has inflated obstacles for the controller. The point cloud data from the camera is used to update dynamic obstacle information. The 3D desirability costmap is constantly published to the controller as well as any user joystick commands. The controller outputs an Ackermann message that include steering and throttle commands. This message is converted into linear and angular velocity for our simulated differential drive wheelchair. Finally, the joystick filter allows these velocity commands through to the robot only when the user’s joystick has a nonzero value; in other words, releasing the joystick results in the wheelchair halting no matter what the output of the MPC.
5.4 Experimental Results

This shared control system is tested in simulation using three different Gazebo environments. Figure 5.3 shows results from one such environment: A large office room with a number of tables of different shapes with different potential docking locations, some of which are obstructed by a variety of other objects (such as chairs, sofas, walls, filing cabinets).

![Figure 5.3: An example Gazebo simulation environment.](image)

The simulated robot is mounted with a Kinect V2 camera that is used to capture point cloud and depth images from the scene. Figure 5.4a shows the robot spawned in the Gazebo environment while fig 5.4b shows the robot model, point cloud, and depth image visualised in RVIZ. This figure also highlights important links used in the simulated robot.

Figure 5.5 presents a static obstacle map built using the *slam_gmapping* node while manually navigating the robot around the room prior to the experiments. This map is initially loaded into the *costmap_2d* package and then simulated laser
scan data is used for online updates. The costmap_2d node also adds an inflation layer to this map.

![Robot spawned in Gazebo.](image1)
![Sensor data and robot model in RVIZ](image2)

**Figure 5.4:** Robot model used as a rough substitute for a wheelchair.

(a) Map built by OpenSLAM in advance.  
(b) Corresponding obstacle costmap with inflation layer.

![Map built by OpenSLAM in advance.](image3)
![Corresponding obstacle costmap with inflation layer.](image4)

**Figure 5.5:** Results from gmapping and costmap_2d ROS packages.

We leverage Votenet to detect tables in this large office environment and applied our docking location finder algorithm to detect potential docking locations around these tables. Figure 5.6 presents an example of these detection results.

Finally, a 3D desirability costmap is generated from detected docking loca-
(a) Candidate object detection.  

(b) Docking locations with heading.

**Figure 5.6:** Finding docking locations around a table using ApproachFinder-CV. A similar set of results were observed with ApproachFinder-NN.

tions, headings and weights. Figure 5.7a shows the desirability costmap corresponding to the current robot heading. As the robot changes its orientation, the visualized 2D slice of the desirability costmap is updated to the nearest heading bin, but the MPC controller always has access to the full 3D costmap. Figure 5.7b presents the nominal path generated by the shared controller using the different task related cost functions and robot dynamics. In this case, the user joystick command was forward and slight right. By following this nominal path, the robot is able to move to the goal in a jerk-free and smooth manner.

(a) 2D slice of the 3D desirability costmap  
(b) Nominal path generated by MPC controller.

**Figure 5.7:** Navigation using the shared controller.
Chapter 6

Conclusions

In this work, we presented two methods to find potential docking locations around tables and toilets. We also presented a method to evaluate these locations based on their relative position, visibility and heading.

ApproachFinder-CV uses a set of complex vision-based techniques to find docking locations in indoor scenes. Although robust, this vision pipeline is computationally slow.

To solve this, we presented a novel deep network that uses data generated by the vision pipeline for training. ApproachFinder-NN is inspired by Hough voting, which votes and then aggregates the votes to generate docking proposals using only geometric information. This model has demonstrated promising performance on a large indoor dataset with partial scans. Lastly, we encoded the docking locations (from either detection pipeline) as a 3D desirability costmap that can be integrated into any real-time shared control system. As a proof-of-concept, we integrated the costmap with an MPC based shared controller in which we used hand-crafted cost functions to find safe navigation paths while considering user inputs. The larger goal of this work is to solve the problem of assistive docking for smart wheelchairs.

Readers are encouraged to test out the code for ApproachFinder-CV\(^1\), ApproachFinder-NN\(^2\) and Wheelchair-Navigation\(^3\) using their own test cases.

\(^1\)ApproachFinder-CV code at: https://github.com/ShivamThukral/ApproachFinder-CV.
\(^2\)ApproachFinder-NN code at: https://github.com/ShivamThukral/ApproachFinder-NN.
6.1 Limitations

Both pipelines involve a stage that narrows down the search space using a fast 3D object detection network to find candidate goal objects (tables or toilets). A pipeline that generates docking locations in a single step, without relying on a preliminary object detection network, might improve inference time further.

In some cases, ApproachFinder-NN provides erratically spaced docking proposal distributions, with patches of sparse and dense coverage (for example, see fig. 4.2h). Also, the quality of docking orientation is limited by discretization in the heading. A large number of bins leads to increased heading accuracy but requires additional training and inference effort.

6.2 Future Work

Future work includes exploring strategies to extend goal detection to indoor objects that are non-planar (such as bathtubs). The pipeline can be extended to handle objects of different geometries using the semantic class detected by Votenet. Object semantic class reveals critical information about geometry, like average dimensions, that can help in docking location detection. We would also like to explore how ApproachFinder can be utilised to find safe passage through doorways. Doorways are categorically different from other objects, as the potential goal locations lie inside the doorway rather than around it.

We would also like to explore how images can provide visual cues to assist docking spot detection. The images can perhaps capture regions that are missed by depth sensors. With advancements in 2D image processing, we believe semantic cues from images can help improve our detection results.

Finally, due to COVID-19 restrictions we were not to deploy our pipeline on a real wheelchair. One of our future goals is to analyse the functionality, safety and usability of our approach with older adults through a user study.
Bibliography


