Implementation of Vision-Based Navigation for Pedestrian Environments

by

Connor William Anderson

B.S. Electrical Engineering and Computer Science
Massachusetts Institute of Technology, 2021

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2022

Author .....
Department of Electrical Engineering and Computer Science
August 19, 2022

Certified by .....
Jonathan P. How
R. C. Maclaurin Professor of Aeronautics and Astronautics
Thesis Supervisor

Accepted by .....
Katrina LaCurts
Chair, Master of Engineering Thesis Committee

Implementation of Vision-Based Navigation for Pedestrian Environments

by

Connor William Anderson

Submitted to the Department of Electrical Engineering and Computer Science
on August 19, 2022, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

Autonomous navigation has rapidly grown to become a predominant field of study utilizing recent advances in robotics and artificial intelligence. Most autonomous navigation methods rely on expensive and complex sensor arrays such as Lidar, which pose practical limitations on the widespread deployment of these devices. This thesis presents an end-to-end implementation of a vision-based navigation pipeline for autonomous navigation in pedestrian environments, utilizing only a single front-facing RGB-D camera and tracking camera as perception devices. This pipeline utilizes 3D monocular object tracking in combination with an advanced Kalman-filter based geometric tracking scheme to track nearby pedestrians, in combination with full SLAM for localization and a reinforcement-learning based navigation stack to navigate through challenging dynamic multi-agent environments. The functionality of this pipeline is demonstrated through a series of pedestrian tracking and navigation experiments with many pedestrians. The tracking module of this pipeline is able to correctly localize pedestrians within 0.4 meters in simple scenarios and 0.6 meters in challenging multi-pedestrian stress testing cases inside of a 12 meter space in spite of limited field of view and relying on only inexpensive RGB camera images. Full end-to-end navigation was demonstrated in a crowded environment with 5 pedestrians, with only one collision out of 13 trials.

Thesis Supervisor: Jonathan P. How
Title: R. C. Maclaurin Professor of Aeronautics and Astronautics

Acknowledgments

I would like to thank Professor How for his guidance, advice, and support throughout my time on this project. I would also like to thank the rest of the Aerospace Controls Lab for sharing their knowledge and experience with me throughout my time there, as well as for creating a great environment for pursuing knowledge together and supporting each other. Special thanks as well to Dr. Golnaz Habibi for her time and expertise in advising and guiding the direction of this project.

This work was supported by the Ford Motor Company.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Autonomous vehicles and autonomous navigation has emerged as a prominent field of study following the advancement and widespread availability of sensor technology, robotics platforms, and the increased scale and power of artificial intelligence models. Recent advances in the capabilities of these systems have enabled deployment of autonomous vehicles in the self driving settings, factory automation, autonomous subterranean exploration, and more. Over the years, many have speculated that autonomous navigation will enable robots to maneuver around pedestrian environments for delivery, exploration, and other applications. However, these predictions have not yet come to pass. Large scale deployment of autonomous vehicles in pedestrian environments has not been deployed in part due to the challenges associated with maneuvering around these pedestrian environments as well as cost barriers to large scale deployment due to reliance on extensive sensor arrays often including expensive LiDaR devices and multi-camera arrays.

Pedestrian environments are particularly challenging to autonomous vehicles for a number of reasons. First, pedestrians environments are inherently unstructured, with Pedestrians moving in any number of possible paths to a large number of potential goals. This stands in contrast to self driving, where cars must follow a strict set of rules in how they move, and for the most part will stay in dedicated lanes outside of intersections. Second, pedestrian environments are varied and can include both indoor and outdoor environments. This poses additional challenges due to the lack of widely

available maps, GPS localization, as well as a variety of environmental conditions that could impair sensors ability to collect data. Third, pedestrian environments often suffer from high levels of occlusion, with static obstacles or other pedestrians often occluding pedestrians from available sensor arrays.

In addition to these inherent challenges, utilizing expensive sensor arrays pose another limitation to the large-scale development of autonomous vehicles in pedestrian environments. For instance, previous attempts at navigation in such environments utilize LiDaR, which can cost tens of thousands of dollars for a single device. Systems that do not utilize LiDaR will often utilize complicated multi-camera sensor arrays that allow the vehicle to see in all directions. In addition to adding cost, these many-camera arrays increase computational complexity that require more expensive and power-hungry compute. In contrast, humans are able to navigate using vision alone without the need for 360 degree field of view(FOV) in part by being able to infer what movements are safe based on available information that we can see. For these reasons, there exists significant demand for autonomous navigation systems that use a single camera device and do not require extensive many-camera sensor arrays (operate under a limited FOV).

This demand for vision-based navigation systems has given rise to increased research in vision-based methods for object detection, localization, mapping, and planning algorithms. Many progress has been made in each of these individual topics, but it is uncommon for research to be done in developing end-to-end pipelines for autonomous navigation using these vision-based systems. Developing these full pipelines introduces new challenges, as individual modules are often trained and tested under a specific set of conditions or using specific datasets/simulations and need to be extended to real-world environments while still providing accurate and robust trajectories to avoid collisions.

This thesis will address all of these considerations by presenting an implementation of a vision-based autonomous navigation stack for use in pedestrian environments. This pipeline is implemented using state of the art algorithms for object detection and tracking, simultaneous localization and mapping(SLAM), and navigation. The

main contributions of this work are a functional ROS implementation of the vision-based navigation pipeline, and an evaluation of these modules in isolation and in end-to-end navigation with real pedestrians. Specifically, this ROS implementation utilizes 3D monocular tracking for robust, long range, accurate pedestrian detection and tracking. In addition, a field of view(FOV) aware kalman filter-based tracking scheme is used to predict the future location of pedestrians in the face of occlusions and when outside the FOV of the camera.

Chapter 2

Related Work

Full end-to-end navigation requires many different modules in order to function. This includes object detection/tracking, localization/SLAM, and planning.

2.1 Previous Visual Navigation Pipeline

The work of this thesis is inspired by previous work on a visual navigation pipeline designed and implemented by Susan Ni from the Aerospace Controls Laboratory(ACL)[11]. While this previous visual navigation pipeline was successful in achieving autonomous navigation and pedestrian avoidance in crowded environments, it was limited by its reliance on outdated 2D object detection methods for object tracking, simple visual odometry methods for localization which do not provide information of the surrounding environment, and simple object tracking scheme with very short-term memory.

The hardware platform for this previous visual navigation pipeline was a Clearpath Jackal outfitted with a pair of front-facing RealSense depth cameras and an on-board IMU. These sensors were used in combination with OpenVINS to perform visual-inertial odometry pose estimation to track the Jackal's position over time. While this localization approach was shown to perform well over short distances, it did not enable the system to detect nearby static obstacles, generate maps, or perform loop closures. In addition, the fact that OpenVINs does not generate maps prevents the navigation module from navigating around static obstacles. While this previous iteration of the

visual navigation pipeline utilized CADRL, which is not able to process static obstacle information, the fact that it does not provide such information restricts its ability to be extended to real-world environments in the future.

In addition, this previous pipeline utilized a 2D object detection module to detect and track pedestrian locations in the 2D camera frame. In order to extract the depth and determine the 3D position of the pedestrian, the pipeline utilized the stereo depth image of the Realsense cameras on the jackal by sampling depth values from the center region of the 2D bounding box. An alpha-beta filter was used to smooth the values for consistency. While this method was able to achieve accurate results, it is limited by the range of the stereo cameras used, which usually limits depth estimation to 10 meters. Additionally, this method may not be robust to more complex crowded environments, where pedestrian bounding boxes may overlap with each other and can suffer from significant occlusion. Further, by utilizing 2D object tracking instead of 3D object tracking, we lose out on additional information that other models would be able to provide, such as orientation and object size/dimensions.

The previous pipeline also suffered from significant delay latency in responding to pedestrian movement and generating new routes. It is unknown if this latency was caused by delay in the collision avoidance model or some other module in the pipeline, but the new implementation eliminates such delay.

The new iteration of this visual navigation pipeline presented in this thesis addresses these limitations by implementing 3D monocular object tracking, allowing for greater robustness and consistency over the previous 2D detection method. In addition, OpenVINS is replaced with full SLAM implemented using the RealSense T265 tracking camera combined with RTABMap, which can generate dense pointmaps and occupancy grids which can be used by the navigation stack for navigation around static obstacles. Finally, by utilizing FOV-aware Kalman filter-based tracking, the perception pipeline is able to more effectively guide navigation in the face of pedestrians frequently exiting and entering the field of view of the camera.

2.2 Object Tracking

Object Tracking is the study of how to detect objects of interest(typically cars, pedestrians, cyclists, cones, ect.) and track their position over time. In autonomous navigation, these object positions and trajectories are used by the navigation systems to generate planned trajectories and avoid collisions with these objects. Object tracking modules can use LiDAR, vision, or a combination of LiDAR and vision. In our case, we limit ourselves to vision as it is more accessible and affordable.

3D object tracking utilizing vision alone poses a considerable challenge to autonomous navigation because autonomous navigation benefits from being able to precisely locate the 3D position of all obstacles. In contrast to LiDAR, which is able to directly measure the 3D position of nearby objects by providing multi-scan pointclouds at high frequency, measuring object depth from 2D RGB image data is very difficult due to the fact that we are trying to extract 3D information from 2D data. For this reason, measuring object depth using vision alone is an inherently ill-posed problem, since there are many possible 3D world which could project to the same 2D images, in the mathematical sense.

The difficulty of vision-based object detection can be shown by comparing the performance of state-of-the-art object tracking methods that utilize LiDAR and those that do not utilize LiDAR. Table 2.1 shows the leading object tracking methods on the nuScenes dataset leaderboard, a large public dataset for self driving [4], at the time of writing. Each of the models in Table 2.1 is compared in terms of Average Multi-Object Tracking Accuracy(AMOTA) and Average Multi-Object Tracking Precision(AMOTP). AMOTA measures the detection accuracy of the tracking model in terms of the number of missed detections, number of false positive detections, and number of incorrect ID switches relative to the total number of ground truth detections(higher is better). AMOTP measures the average distance(error) between the the model and ground truth detections, usually measured as an intersection over union(IOU) between the two bounding box regions(lower is better). Leading Lidar-based approaches outperform vision-based approaches by 0.753 AMOTA on all object

Table 2.1: Performance Comparison of State-of-the-Art Object Tracking methods on the nuScenes dataset

Name	Sensor Modalities	AMOTA	AMOTP
Camo-MOT	**LiDAR, Camera**	0.753	0.472
BEV-Fusion	LiDAR, Camera	0.741	0.403
GTFS	**LiDAR**	0.733	0.550
Trans-Fusion	LiDAR, Camera	0.718	0.551
SRCN3D	**Camera**	0.398	1.317
BEVTrack	Camera	0.341	1.107
QD-3DT	Camera	0.217	1.550
CenterTrack3D	Camera	0.046	1.543

categories(cars, pedestrians, cyclists, ect.) compared to 0.398, and by 0.472 AMOTP compared to 1.317 on all object categories.

Recent visual object detection algorithms utilize convolutional neural networks which take image data as input and provide a list of annotations as output, which describe all objects visible in the frame in terms of 2D or 3D bounding boxes. 2D object detection provides 2D bounding boxes which describe the 2D position and size of detected objects in the image plane. 3D object detection, on the other hand, provides estimated 2D position as well as 3D bounding box parameters including relative 3D position, dimensions (length, width, height), depth, and orientation. Vision-based 3D object detection can be further broken down into monocular object detection and stereo object detection. Monocular object detection attempts to infer 3D bounding box parameters from a single RGB image alone, whereas stereo object detection utilizes either a pair of RGB images from a two-camera stereo array or an RGB-D image, which includes 3 color channels in addition to a pre-calculated depth map.

Visual object tracking is the process of detecting object position over time and maintaining unique object IDs over time for each visible object. There are two main methods of visual object tracking: tracking by detection and simultaneous detection and tracking. Tracking by detection runs detection independently on every image frame, and includes a separate tracking module on top that associates annotations of sequential frames together using their location or extracted features. Simultaneous detection and tracking modules, on the other hand, link the detection and track-

ing together, typically as output of a single neural network that takes as input the annotated output of the detection module.

2.2.1 2D Object Detection

2D object tracking modules do not provide any depth or 3D location information about detected pedestrians, which is necessary for planning and collision avoidance in 3D environments. Depth information can be acquired using depth cameras, but these depth cameras usually have an effective range of <10 meters, thus significantly limiting the system beyond 10 meters. Additionally, extracting depth values from these depth maps can be problematic in complex high-occlusion environments since it is difficult to determine exactly which pixels correlate to the detected pedestrian. The previous visual navigation pipeline implemented by ACL utilized a 2D object tracking module called CenterTrack-2D [19]. We hope to improve upon this module by using a 3D-based object detection module that provides richer information about the location of pedestrians in the surrounding environment.

2.2.2 3D Object Detection and Tracking

This section outlines current state of the art visual 3D object detection and tracking algorithms. As mentioned previously in this section, object tracking can be performed in one of two ways: tracking by detection or simultaneous tracking and detection. Some of the algorithms explored in this section perform 3D object detection only, meaning that an additional object tracking algorithm would need to be implemented in addition in order to perform object tracking and to estimate velocities.

3D visual object detection can be further classified into two categories: monocular object detection and stero detection. 3D monocular detection uses the image of a single RGB camera image as input into a neural network to generate 3D location, dimensions, and orientation estimates. 3D monocular detection provides much richer information about the detected agent that is useful in navigation, at the cost of increased training times and higher computational intensity. 3D stereo detection, on

the other hand, uses a pair of RGB cameras or a single RGB-D depth camera to perform the same task as 3D monocular tracking. Some research has been made into 3D stereo object detection, but these modules are generally more computationally expensive and often do not actually outperform 3D monocular object detection.

Object Detection Only Algorithms

SMOKE is a 3D monocular detection model that predicts 3D bounding boxes for each detected object by combining a single keypoint estimate with regressed 3D variables, in contrast to other 3D monocular trackers that rely on 2D labels in the image plane to estimate 3D information[9]. In contrast to models such as CenterTrack3D and Quasi-Dense, SMOKE is a detection only algorithm. Thus, in order to implement object tracking with SMOKE, a tracking by detection scheme would have to be implemented on top.

MonoLoco is a 3D vision library that predicts 3D pedestrian depth and orientation using 2D keypoints generated by OpenPifPaf[1]. These keypoints identify the 2D location of specific regions of the human body, such as the head, torso, elbows, and knees. This works quite well for detection and orientation estimation, but is not very accurate in terms of depth estimation. In particular, MonoLoco struggles to accurately resolve height differences between pedestrians, often measuring the depth of shorter pedestrians to be farther away due to the fact that it relies on the distance between 2D keypoints to generate depth estimated. Additionally, MonoLoco is an pedestrian detection model only, meaning a separate tracking module would be required to implement tracking by detection.

MonStereo, an extension of MonoLoco, attempts to target the poor depth estimates of MonoLoco by using two cameras set up in a stereo fashion, identifying keypoints in both images independently, and then using both sets of keypoints as input into the detection module used to output orientation and distance[2]. Using stereo vision in this way, as opposed to depth cameras, mimics the stereo vision that humans have using our two eyes. As with MonoLoco, MonStereo is for detection only would require an additional object tracking module to implement simultaneous

detection and tracking.

Detection and Tracking Algorithms

CenterTrack3D is the 3D version of the CenterTrack2D module[19]. CenterTrack3D is a simultaneous detection and tracking model that performs tracking between consecutive frames by passing two consecutive image frames into a single neural network along with a heat map generated around the center points of all detections in the previous frame. The neural network generates 3D annotated bounding boxes labeled with IDs to maintain continuity between consecutive frames. CenterTrack3D was not used in the previous visual navigation pipeline [11] because the only model trained for 3D object detection and tracking utilized the nuScenes dataset. Because nuScenes is a self-driving dataset, this model tends to underperform in several key areas of interest for pedestrian tracking, such as when pedestrians are close to the camera, when pedestrians are crowded very close together, and in indoor environments. To work around this limitation, we plan to utilize additional pedestrian focused datasets to retrain Centertrack-3D or fine tune the nuScenes model to achieve better performance in these key areas.

Quasi-Dense(or QD-3DT as shown in 2.1) is another more recent 3D monocular tracking module that performs simultaneous detection and tracking using camera input[12]. Quasi-Dense uses a similarity learning approach that samples hundreds of proposed regions on a consecutive pair of images to more effectively detect and track agents in some cases. Quasi-Dense is able to achieve effective object tracking by extracting visual feature vectors in addition to bounding box information in its region proposal network, which is then used in its quasi-dense matching approach to match bounding boxes between frames using visual features in addition to bounding box location. However, because of the additional overhead of this large neural network with visual feature vector extraction, Quasi-Dense is significantly more computationally intensive then alternative methods.

2.2.3 Object Detection Datasets

All Object Detection methods described in this section rely on CNNs to generate object annotations. CNNs require extensive data and training time in order to be effective, and thus rely on large datasets to perform training. Many public datasets are now available to researchers to perform such training for the development and testing of object detection methods. Most of these datasets, such as Waymo, KITTI, and nuScenes are self-driving datasets that provide Lidar scans and images paired with a set of 3D annotations for objects including cars, pedestrians, and cyclists [4]. These datasets work well for training self driving object detection models, but do not work well for pedestrian environments. This is because self-driving datasets provide camera images from a high point of view and usually annotate objects from a large distance away. These patterns do not extend well to indoor pedestrian environments where pedestrians are seen from below and often within 3 meters away.

Thus, effective object detection in pedestrian environments requires specific datasets targeted for that specific task. JRDB is one such pedestrian dataset that collects 360 degree image data from their JackRabbot robot [10]. For this reason, the detection model in this thesis is re-trained using this JRDB dataset rather than relying on models trained on the more common self driving datasets.

2.3 Localization and Mapping

In order to navigate through space, a navigation module must be able to track it's own position in 3D space over time. This problem is known as state estimation or localization. The goal of the mapping task is to generate feature maps of the surrounding environment using camera or LiDaR data. These two tasks are often paired together to implement Simultaneous Localization and Mapping(SLAM). SLAM methods are able to produce maps of the environment that can be used to navigate around static obstacles as well as perform loop closures.

The simplest way to implement localization is to simply integrate sensor data such as wheel odometry or IMU measurements over time. However, relying on a single

sensor can result in drift, and wheel odometry can suffer from wheel slippage which results in odometry measurements that differ significantly from actual movement.

In order to further refine localization, multiple sensors can be combined together. One way to accomplish this is the use of an Extended Kalman Filter(EKF), which can incorporate odometry measurements from multiple sources to generate actual displacement measurements closer to reality. Another approach is to use Visual Inertial Odometery (VIO). These visual localization methods track basic visual features across frames within a temporal window in combination with inertial measurements to track visual features over time, estimate their 3D location, and use the movement of these 3D points over time to refine the camera position estimate. Examples of this include OpenVINS [8] and VINSFusion [15] [14].

Another common and powerful approach is Simultaneous Localization and Mapping(SLAM). The goal of SLAM is not only to localize to position of the agent in space, but also to generate a map of the environment. This map of the environment can then be used to navigate around static obstacles, as well as to refine localization predictions utilizing that static information and by performing loop closures. Loop closures are when a SLAM system detects the presence of a trajectory loop using the current position and local data in the map and comparing against other nearby possible locations in the global map.

SLAM algorithms generally perform localization and mapping by tracking visual features as point in the 2D camera frame over time. The movement of these 2D points over time can then be used to estimate the 3D position of each of those points in addition to the relative movement of the camera over time. The relative movement of the camera is used to perform localization and generate camera pose estimated, while the 3D points tracked via their corresponding 2D visual features can be used to create a map. Then, SLAM systems can perform loop closures by comparing any currently tracked 3D points to the 3D points on the map in order to recognize when it has arrived in a location it has already been to.

One such example of these SLAM algorithms is ORB-SLAM3. ORB-SLAM3 performs SLAM by tracking efficient and orientation-invariant "ORB" visual features

in the 2D camera frame over time [5]. ORB-SLAM3 uses these visual features to generate a sparse map of 3D map points(which each correlate to a specific ORB feature) and estimate camera velocity. ORB-SLAM3 is able to further utilize a variety of different visual data types including standard monocular(singe-camera) data, as well as stereo(two-camera), RGB-D (depth map generation), and fisheye cameras. Although ORB-SLAM3 does provide the option of utilizing IMU data, this IMU data is only used during the initialization phase when the algorithm is attempting to create enough 3D map points required to perform localization. As a result, ORB-SLAM3 will often experience significant drift in areas which lack enough visual features such as hallways, open outdoor areas, or in darkness. ORB-SLAM3 attempts to resolve these errors by performing re-initialization when tracking fails, but it often does not detect the failure fast enough, resulting in significant pose deviations from reality. To make matters worse, these pose deviations result in the incorrect placement of 3D map points that are committed to the map, resulting in a map that is tarnished with incorrectly placed points that impair the ability to perform loop closures. In addition, ORB-SLAM3 only produces a sparse 3D point map consisting of map points which can be easily detected from 2D orb point features. This can be effective in performing localization and loop closures, but is not very useful in generating a meaningful map of the environment that can be used to navigate around static obstacles.

GR-SLAM is another SLAM approach that tightly integrates multiple sensors including camera image data, IMU data, and odometry information to perform SLAM [16]. GR-SLAM has promising results, in comparison to VINS-Fusion. However, GR-SLAM does not have open source code, posing a significant barrier to utilizing this approach in a short time frame for this thesis.

An alternative localization approach offloads all or part of the localization task to specific hardware in the form of tracking cameras. These tracking cameras perform visual feature tracking similar to the SLAM method outlined in this section, but is able to run at a much higher frequency due to dedicated hardware. Additionally, tracking cameras are able to tightly integrate visual odometry with IMU measurements in hardware, resulting in pose estimates that are very reliable. These tracking

cameras perform visual feature extraction and tracking with dedicated hardware and output high frequency pose estimated to the main computer. One such device, the RealSense T265 tracking camera, has been demonstrated to effectively perform such localization [17]. Despite performing visual feature tracking as utilized in SLAM, these tracking cameras do not utilize this information to generate maps or perform loop closures. Thus, any scheme that utilizes on of these cameras could benefit from being combined with an additional mapping algorithm such as RTABMap to generate maps, occupancy grids, and perform loop closures to further refine pose estimates.

2.4 Navigation Methods

Navigation modules are responsible for taking robot location (ego-pose), tracked object position, and static obstacle/map information as input and calculating optimal robot trajectories to reach local goals in as little time/distance traveled as possible while avoiding collisions with other agents or static obstacles.

CADRL is a motion planner that utilizes deep-reinforcement learning to perform navigation and collision-avoidance among a series of dynamic agents [6]. Reinforcement learning is well suited to the task of pedestrian navigation because it is able to model interactions between many moving agents and complex interactions that will occur between decision-making agents. CADRL encodes world state (position and velocity of other agents) as input vectors to a cell-based long short-term memory (LSTM) network. This LSTM is used to predict a robot trajectory to avoid collisions while making progress to a local goal. While CADRL is able to perform effective navigation in pedestrian environment with many agents, it does not incorporate static obstacle information in its trajectory planning and does not provide any kind of global goal routing.

Go-MPC is another promising navigation alternative to CADRL that utilizes local trajectory optimization methods in the form of model predictive control (MPC) in addition to a reinforcement learning approach for selecting local subgoals to achieve autonomous navigation around both static and dynamic obstacles [3]. However, Go-

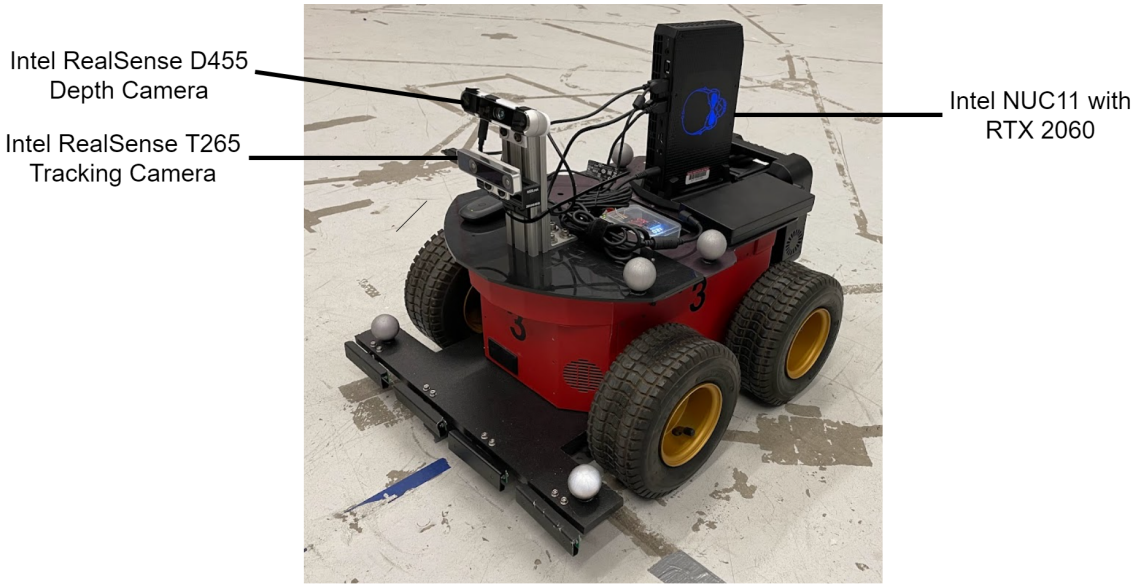


Figure 2-1: The Red Rover used as a testing platform for the vision navigation pipeline.

MPC requires a map of the environment to perform navigation to a global goal. While providing a map ahead of time is feasible in many pedestrian environments, it poses limitations in many environments that are large, wide open and difficult to map with vision-based methods, or environments which are prone to changing frequently.

2.5 Hardware Platform

The hardware platform used to implement and test the vision navigation pipeline presented in this thesis is shown in figure 2-1. The rover itself is a Pioneer 3-AT rover, and comes equipped with two front-facing cameras: an Intel RealSense D455 depth camera and an Intel RealSense T265 tracking camera. In addition to providing RGB images, the D455 depth cameras produces a depth map using its' dual infrared cameras, and automatically performs alignment between the depth and color images to generate ROS pointcloud2 images. The T265 camera utilizes its two fisheye cameras to track visual features in combination with its internal IMU to perform accurate 3D pose estimation over time. In terms of computation, the rover is equipped with an Intel NUC11 with an Intel i7 desktop processor and RTX 2060 graphics card.

The D455 depth camera is oriented at a 20 degree angle so that it can capture the floor in front of the rover in addition to capturing as much of the pedestrian as possible when pedestrians are up close to the robot. While D455 depth information is not utilized in the object detection/tracking modules of this project, it is utilized in localization/mapping techniques using RTABMap.

Chapter 3

Methods Summary

This thesis presents a new implementation of a vision-based navigation system with the goal of improving on the previous work of [11]. The approach outlined in this section first provides a full system overview, then details each of the key modules necessary for the implementation of end-to-end navigation.

3.1 Full Pipeline Overview

Full end-to-end vision-based navigation is implemented as shown in Figure 3-1. Each of the boxes in the diagram correspond to ROS nodes implemented in ROS Noetic, with the dotted box corresponding to the perception devices. Arrows correspond to ROS messages sent over the local ROS network. As input, the pipeline receives images and pose data from the Intel Realsense D455 and T265 devices in addition to a Local Goal to navigate to. The primary output is a velocity command that is received by a Teensy onboard the Red Rover platform, where it is converted into motor controls to produce the desired velocity.

The RTABMap node receives 3D Pose information from the T265 in combination with the RGBD depth image/pointcloud from the D455. RTABMap uses this information to generate a dense 3D pointmap of the environment, a 2D top-down occupancy map, and the refine the 3D pose passed to it by the T265. The RGB images from the D455 are passed to the CenterTrack3D module, where they are used to gen-

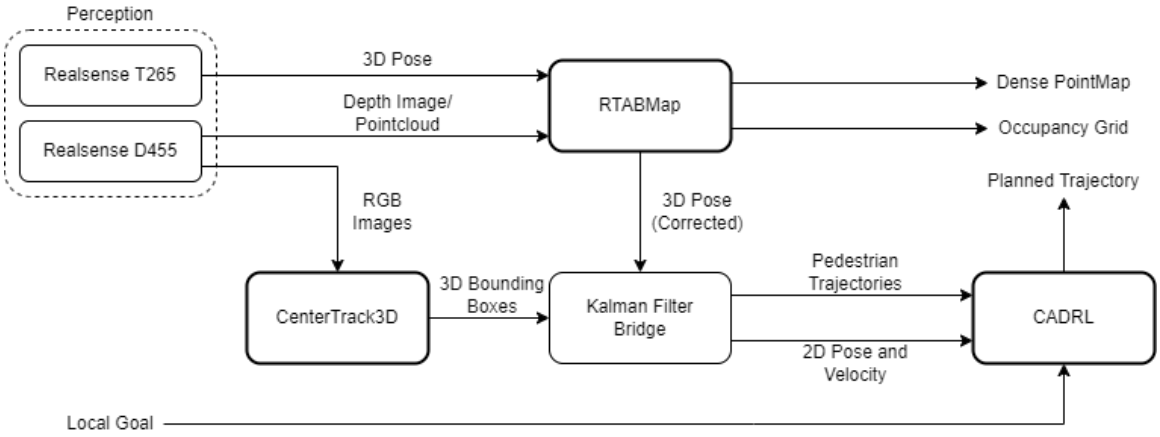


Figure 3-1: Vision Navigation Pipeline including inputs, hardware, ROS modules, and outputs

erate the 3D bbox annotations. These 3D annotations are passed to the Kalman filter bridge along with the rover pose estimate from RTABMap. The Kalman filter bridge maintains its list of tracklets corresponding to tracked pedestrians, and on each frame published the latest tracklet information to CADRL along with the pose and velocity of the rover. The Kalman bridge node converts the rover pose and tracklet position to a 2D top-down perspective when passing to CADRL since CADRL only utilizes 2D information. CADRL utilizes this information along with a local goal specified by the user and outputs a planned trajectory for the rover to take to reach the specified local goal.

3.2 Key Modules

3.2.1 Full Visual SLAM using RTABMap

The localization method of this visual navigation pipeline integrates the accurate and high-frequency pose estimations provided by the T265 with the pointcloud generated by the D455, and passes the result into RTABMap. RTABMap utilizes this information to create a dense 3D pointcloud map of the environment. This pointcloud map is used to refine the pose estimates generated by the T265, and can perform loop closures when the rover navigates somewhere it has already been. In addition, the

3D pointcloud map generated by RTABMap can generate an occupancy grid which can be used to inform the navigation stack of static obstacles in the environment.

This approach was chosen because the T265 is able to generate high frequency and high accuracy pose estimates without placing significant load on the processor. Additionally, the T265 tightly integrates IMU measurements with visual odometry automatically, enabling accurate pose estimates in a variety of conditions. While the T265 does suffer from drift over time, such drift is common among visual odometry methods, and the T265 performs comparatively well as long as it has access to enough visual features. Combining the T265 with RTABMap allows for easy and efficient generation of maps which can be used to perform loop closures and refine localization results. The mapping functionality of RTABMap is called with a frequency of 1 Hz. While this is a very relatively low frequency, it is sufficient for generating accurate dense pointcloud maps, avoids unnecessary load on the processor, and keeps the size of the map low so it can fit comfortably in memory.

This approach was compared against ORB-SLAM3 and produced pose estimates that were more accurate and robust, in addition to utilizing considerably less compute. Trajectories estimated using ORB-SLAM3 would often lose tracking in areas with poor visual features, resulting in radically incorrect trajectories. ORB-SLAM3 attempts to resolve this by re-initializing, but usually was not able to detect these drifts before committing the results to its database, resulting in incorrect paths and incorrect maps. This problem was particularly common in tight indoor corridors and when performing turns.

An example of one of these ORB-SLAM3 tracking failures occurring is visualized in Figure 3-2. In this figure, the top window visualizes the camera trajectory and generated map while the bottom window shows the image with green dots to indicate tracked 2D visual features. In the map window, the blue line represents the camera trajectory, the black dots indicate committed 3D map points, and the red dots correspond to 3D map points that are currently being tracked. In the left frame, SLAM is being performed correctly with a properly generated trajectory estimation and map shown in the top window. In the right frame, the river makes a right turn

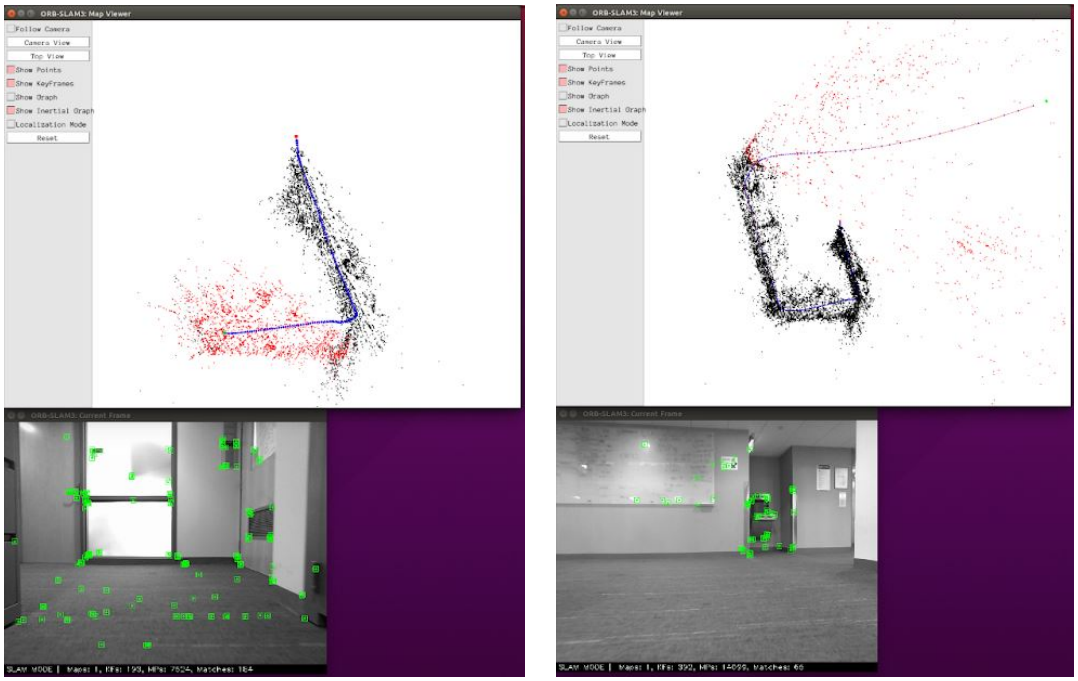


Figure 3-2: ORBSLAM3 Test Case Example

and ORB-SLAM3 loses enough visual features to correctly perform SLAM, resulting in an incorrect trajectory which quickly accelerates to the right.

In contract to ORB-SLAM3, the T265 performs more effective, accurate, and consistent trajectory estimations because it operates at a much higher frequency enabled by dedicated hardware. At this higher frequency, it is able to more effectively track visual features when the camera is moving rapidly such as the turning case shown in Figure 3-2. While we do find that the performance of the T265 does degrade in some environments where visual features are lacking, it does not suffer from the same large errors as ORB-SLAM3.

3.2.2 Pedestrian Detection using CenterTrack3D

Pedestrian detection in this pipeline is performed by a CenterTrack3D model that has been retrained on the JRDB pedestrian dataset. This model was chosen because it is able to produce accurate 3D annotations at a frequency of $> 10Hz$ on our hardware. 3D monocular tracking provides robust estimates of 3D bounding boxes that can be used to estimate pedestrian location and trajectory up to and beyond

25 meters and in the presence of significant pedestrian occlusion. This frequency is important because the rover must be able to react to pedestrians that walk in front of the rover from outside the field of view, potentially giving the rover very little time to react. While Quasi-Dense does outperform CenterTrack on AMOTA as shown in figure 2.1, CenterTrack achieves a nearly identical AMOTP while utilizing considerably less compute and thus operating at a higher frequency on our hardware. The lower AMOTA does mean that the ids provided by CenterTrack are less reliable than Quasi-Dense, this doesn't matter in our case as we opt for another tracking approach utilizing Kalman filtering.

In this work, CenterTrack3D was retrained on the JRDB dataset because it was observed that the provided model trained on the nuScenes self driving dataset did not perform well under our hardware setup. Specifically, the nuScenes dataset was designed for self driving cars as opposed to pedestrian environments, and benefits from the relatively high position of the camera in that environment and a relatively large distance between the camera and detected objects. When extended to pedestrian environments, the nuScenes trained model struggles to consistently detect pedestrians, pedestrian depth values, and is unable to detect pedestrians up close (< 3 meters) when the whole pedestrian is not visible. By retraining the model on the JRDB dataset, the CenterTrack model is able to more consistently detect pedestrians, has more accurate depth estimates, and can detect pedestrians up close without the need for retraining. An evaluation of the performance of this model is provided in section 4.1.1.

In addition to generating 3D bbox annotations, CenterTrack3D also labels each detection with an ID, and can associate object IDs between consecutive frames. In practice however, this ID is often switched or renewed when pedestrians cross in front of each other in the 2D camera frame. To address this issue, we rely on an alternative tracking scheme which only relies on the CenterTrack IDs when initializing tracklets for pedestrians, and relies on a Kalman filter after being initialized.

3.2.3 Pedestrian Tracking using Kalman Filtering

While CenterTrack3D does label each 3D annotation with a label, it can only associate IDs between consecutive frames. This means if a pedestrian is not detected for a single frame, because of a dropped detection or occlusion, then it is assigned a new ID the next time it is detected. This type of tracking is not sufficient for our purposes because it is unable to track pedestrians through occlusions. Additionally, if a pedestrian leaves the field of view of the camera, this tracking scheme immediately "forgets" that the pedestrian is there. These problems would interfere with a navigation system because tracked pedestrians could frequently jump in and out of the planner's input. In order to provide more continuous estimations of pedestrian location, a Kalman Filter Bridge node is utilized. This Kalman filter bridge node receives as input the 3D annotations produced by CenterTrack, utilizes a Kalman filter to predict the next location of each pedestrian on every time step, and is able to maintain continuous tracklets for pedestrians even in the event of short term occlusions. This is able to more effectively keep tracklets alive in the face of occlusions, leads to re-identification after occlusions longer than 1 frame, and enables the tracking scheme to remember the location of pedestrians even when outside the field of view of the camera.

The behavior of this Kalman Bridge module is further explained by figure 3-3. The Kalman Bridge module maintains a list of tracklets that each represent a pedestrian in the environment. On every frame, CenterTrack3D receives the incoming RGB images and produces a corresponding set of 3D bbox annotations. These annotations are used as input to the Kalman Bridge Module. On every new set of annotations, the bridge module attempt to find a matching annotation for each tracklet in its tracklet list. To solve the matching problem between incoming annotations and tracklets in the tracklet list, the problem is formulated as a stable marriage problem, and is solved using a modified version of the Gale-Shapley algorithm [7]. This algorithm matches each tracklet to the detection that is closest in 3D space. In addition, the bridge node has a threshold distance value such that every annotation must be below the threshold distance to be considered a match. Thus, if there is not matching detection

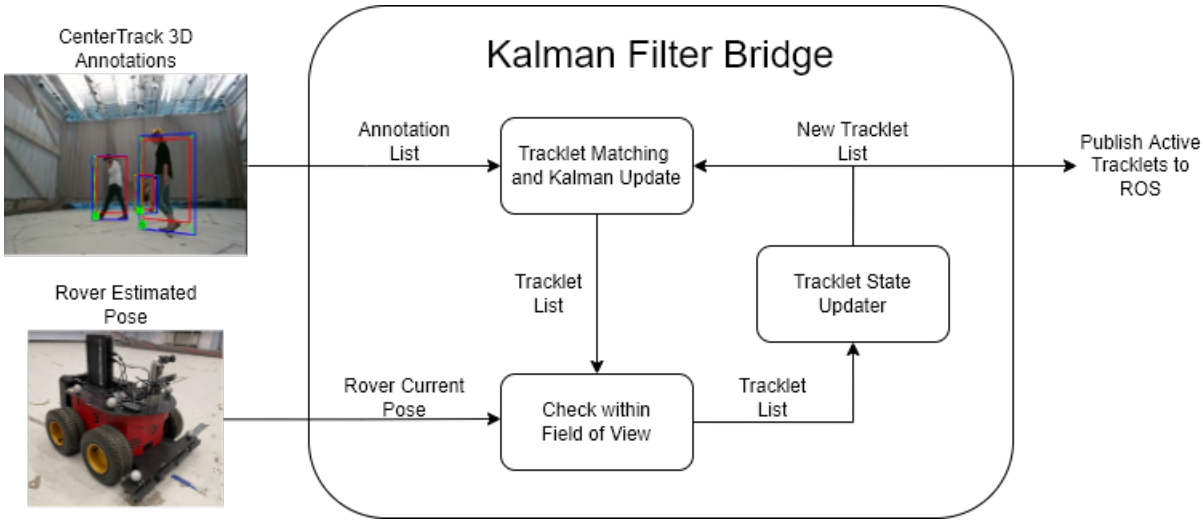


Figure 3-3: The basic flow of the Kalman Bridge Module. A: Incoming Annotations are used to populate an annotation list. B: Each Tracklet in the Tracklet list is compared against all remaining annotations in the annotation list. If a matching annotation is found, that annotation is removed from the annotation list. C: The robot pose input is used to estimate whether estimated tracklets are in the robot's field of view. D: Tracklet state is updated according to the procedure in figure 3-4. Tracked/Lost/Lingering tracklets are published to ROS and old tracklets are removed.

for a tracklet on a given frame, it does not receive a match. Similarly, if a detection is not within the threshold distance of any annotation, it is considered unmatched and is used to create a new tracklet (according to the discovery process outlined later in this section).

Each tracklet tracks the position, dimensions, orientation, and velocity of the estimated pedestrian using a Kalman Filter. Once a tracklet is matched with an annotation, the 3D location, dimensions, and orientation of the annotated bbox are used to update the Kalman filter parameters. If a tracklet does not have a matching annotation, the next position of the estimated pedestrian is predicted using the velocity estimate calculated by the Kalman filter.

Each tracklet maintains a state with one of 5 possible states: Discovered, Tracked, Lost, Lingering, and Removed. The transition between states is primarily dependent on whether or not the tracklet is matched against a CenterTrack3D annotation.

When a CenterTrack3D annotation is passed to the Kalman bridge module that does not match any tracklet in the tracklet list, a new tracklet is created from that

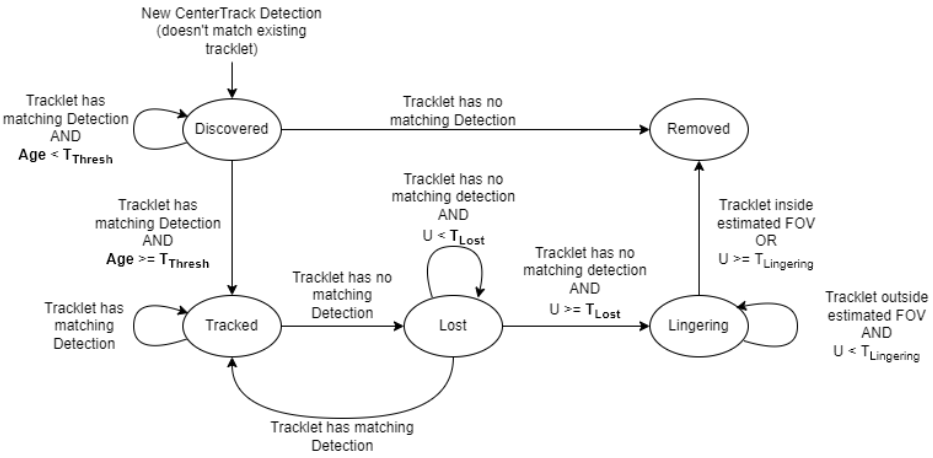


Figure 3-4: Kalman Bridge State Diagram. A transition occurs across each arrow when the Kalman Bridge Module receives a set of annotations from the CenterTrack module.

annotation and begins in the "Discovered" state. This discovered state is an intermediary state that tracklets remain in for a few frames when first detected. Tracklets in this state are not published to the planner, and move to the "Tracked" state after a few frames. The state transitions from the discovered state are described as follows:

- The tracklet remains in the discovered state as long as an annotation with the same CenterTrack ID is received on each frame.
- If no match is received, the tracklet is set to the removed state, signaling the Kalman bridge node to remove it from the list. Every time it is matched, the age of the tracklet increases by 1.
- Once the age of a tracklet in the discovered state exceeds some threshold value T_{thresh} (set as 5 in our tests), then it moves to the Tracked state.

Note that it is only in this discovered state that the CenterTrack ID is used. This is because tracking geometrically using the Kalman filter is more accurate in the long term than the CenterTrack ID in practice, so we only use the CenterTrack ID in initialization.

Once the tracklet is in the tracked state, the tracklet will persist even if there is no matching annotation on that frame. The tracked state indicates that the tracklet

is actively being matched to a CenterTrack detection for a pedestrian in the field of view of the camera. Instead of being removed, if a tracklet in the tracked state is not matched with an annotation, it is moved to the "Lost" state. In the lost state, the Kalman filter is used to predict the next position of the tracklet, and the incoming annotations are still checked for matching lost tracklets on every frame. If a new match is found while in the lost state, the tracklet is updates accordingly and its state is moved back to the discovered state. However, if no match is found for a tracklet in the lost state for T_{lost} frames, then the tracklet is moved to the lingering state.

Once in the lingering state, the Kalman bridge no longer attempts to match the tracklet against incoming annotations. Instead, the tracklet is maintained and its position is updated purely according to the Kalman filter. However, Lingering tracklets only persist so long as they are not estimated to lie in the field of view (FOV) of the camera. Whether or not a lingering tracklet is in the FOV of the camera is determined geometrically by considering the position of the camera, orientation of the camera, and position of the tracklet. The purpose of the lingering state is to provide a longer lifetime to tracklets that lie outside the field of view of the rover. This is useful because the rover has a very limited field of view, and explicitly informing the rover that it would not be able to see a tracklet even if it was still there can be used to extend the lifespan of lost tracklets and give the rover a better idea on what is happening outside its FOV. If the lingering tracklet ever enters the estimated FOV, or if it exceeds $T_{lingering}$ frames since it was last matched with a detection, then it is removed.

While keeping tracklets alive for additional time in the lingering state if they are outside the field of view does create the possibility for significant discrepancies between the tracklet position and reality, this trade off is acceptable in this case due to the limited field of view of our system. If this were not done, then the pipeline would quickly forget the position of a pedestrian if it turns away from that pedestrian. In practice prior to the implementation of this lingering state, the rover would frequently oscillate between trajectories as it turns to avoid a pedestrian, forgets that it is there,

and turns back to its original trajectory.

The Kalman bridge node then publishes each tracklet that is in the discovered, lost, or lingering states to ROS where it is received by the navigation module.

3.2.4 Navigation using CADRL

The navigation module of this pipeline utilizes CADRL. CADRL utilizes reinforcement learning to train policies to navigate around a specified list of nearby clusters that represent pedestrians in this case [6]. This method was chosen because it has been demonstrated to effectively navigate around pedestrians in dense environments. Additionally, CADRL already has a ROS implementation allowing it to be easily integrated with the visual perception stack.

While CADRL is able to perform navigation around dynamic (moving) obstacles, it is not able to utilize static obstacle information simultaneously. This means that the rover will navigate to a specified local goal without considering the 3D static map generated by RTABMap. Go-MPC was considered as an alternative module for the navigation stack of this pipeline, but was not implemented because it did not have a readily available ROS implementation and utilizes a MPC solver that requires a license for each device it is used on. Future iterations of this pipeline should consider writing a ROS implementation of Go-MPC or a similar pipeline to utilize the advantages of MPC in combination with reinforcement learning.

In addition, CADRL was trained in a simulation with a much larger field of view (180 degrees) than the limited field of view of our pipeline. This greater field of view was achieved using a Lidar device. This presents an additional challenge to navigation, as the trained model assumes knowledge of a full 180 degrees even though it only has a field of view of 80 degrees.

3.3 Summary

Chapter 4

Results

This section provides detailed performance analysis of each key modules of the presented vision-based navigation pipeline. It also includes end-to-end tests showcasing the pipeline's ability to safely and efficiently navigate autonomously through crowded environments.

4.1 Pedestrian Tracking Evaluation

This section presents an evaluation of the pedestrian detection and tracking performance of the approach detailed in the previous chapter, including CenterTrack3D object tracking trained on the JRDB dataset and the Kalman-filter based tracking implemented on top.

4.1.1 CenterTrack3D Evaluation

We compare pedestrian detection and tracking performance the CenterTrack3D model trained on the JRDB pedestrian dataset (JRDB model in short) with the original CenterTrack3D model which has been trained on nuScenes dataset (nuscenes model) [19]. For our experimental setup, the former outperforms the later in a number of ways that will be explained in this section. In the first experiment, a single pedestrian is walking around the ACL high bay space in the field of view of the stationary rover

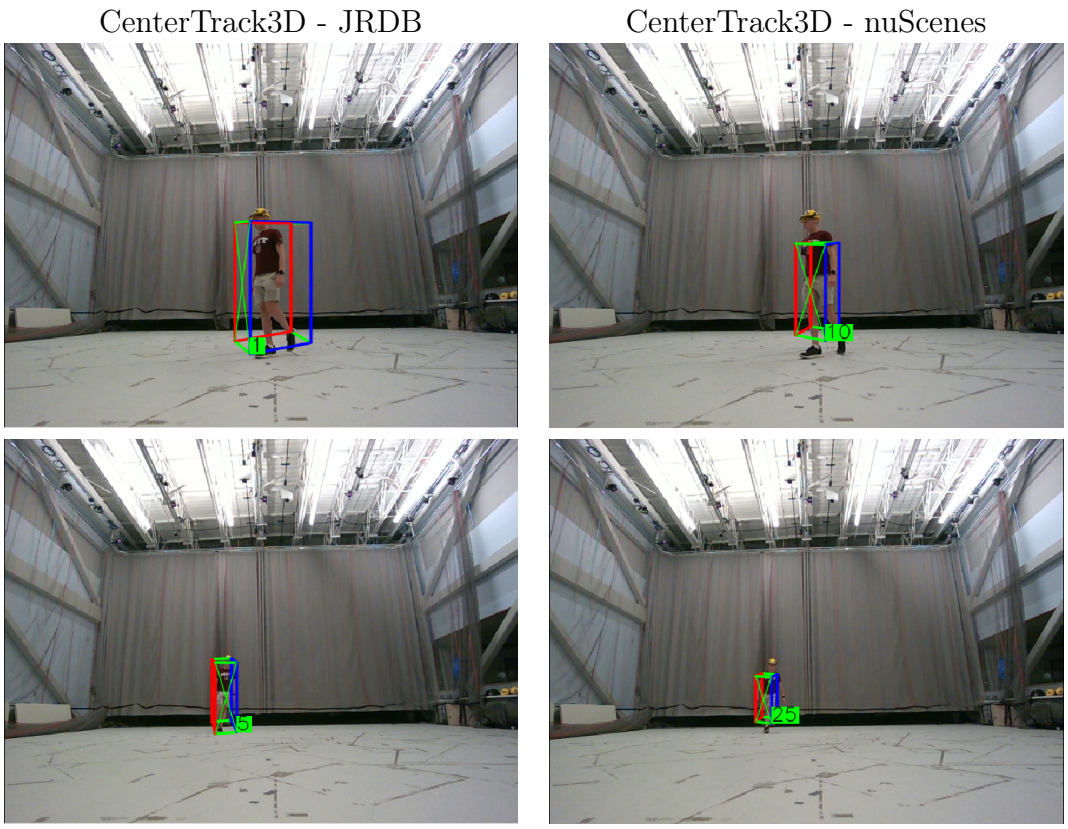


Figure 4-1: Comparison of CenterTrack3D performance when trained on JRDB dataset vs nuScenes dataset

for a period of 40 s.

The predicted position of the pedestrian by two models of JRDB and nuScenes of CenterTrack3D was compared to the "ground truth". Ground truth result is obtained using a Vicon tracking system, which tracks the helmet worn by the pedestrian. The resulting bounding boxes can be visualized over the original image in Figure 4-2. These images illustrate how the nuScenes model consistently overestimates the pedestrian depth in all cases(produces a depth estimate that is greater than reality). The overall quantitative result of the test are summarized in Table 4.2. The JRDB model is able to successfully detect the pedestrian 100% of the time in this case, while the nuScenes model only detects the pedestrian 76.3% of the time. In addition, the JRDB model was able to correctly predict the pedestrian's 3D position much more accurately. In terms of tracking, the ID switches in JRDB model are occurred less frequently than nuScenes model due to the consistent detection rate.

Table 4.1: CenterTrack3D Models Performance Comparison

CenterTrack3D Model	Detection Rate (percentage)	Average Displacement Error (meters)	ID Switches
JRDB	100.0	0.362	4
nuScenes	76.3	2.768	34

The new JRDB trained CenterTrack model still suffers from 4 incorrect ID switches in this experiment, at a rate of one id switch every 10 s in ideal conditions(high visibility, no occlusions). This rate of id switching is too high to be reliable for consistent pedestrian tracking, illustrating the need for a more consistent tracking scheme. This is the motivation behind the Kalman filtering approach. detailed in section 3.2.3

It is likely that the pedestrian detection results would be even more accurate if fine-tuned on data collected and annotated from our specific hardware setup. However, such fine tuning would require extensive data collection and time-consuming manual annotation which is beyond the time frame of this project.

4.1.2 Object Tracking Result using Kalman Filter Bridge

The performance of the Kalman filter bridge module was evaluated on a similar single pedestrian tracking test. In this test, the raw position estimated by the JRDB model was compared to the same annotations passed through the Kalman filter bridge module. The ground truth position was calculated using a Vicon tracking system that tracks the position of a helmet as a rigid body. The resulting path estimation and Vicon ground truth path are visualized in figure 4-2. Numerical analysis is provided in Table 4.2. The results are reported in average displacement error, which represents the average 3D euclidean distance(error) between predicted pedestrian center point location and the ground truth center point of the rigid body tracked by the Vicon tracking system.

The average displacement error for each of these estimated trajectories is compared in table 4.2. While the Kalman filtering produces a much more stable estimated trajectory, as shown in Figure 4-2, it does so at the cost of some accuracy. In particular, the Kalman filter output in this case has an additional 0.037 m of accuracy on average

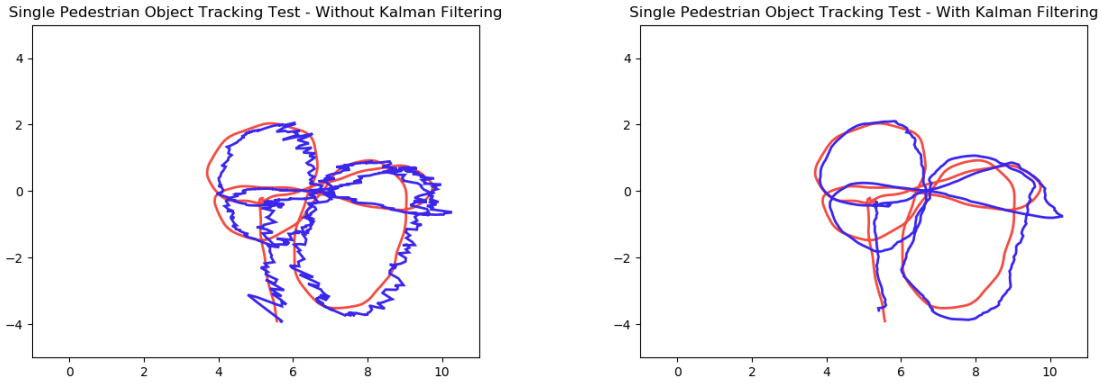


Figure 4-2: Single Pedestrian Tracking Test - Tracklet Paths with and without Kalman Filtering. Red path represents Vicon ground truth and blue path represents estimated tracklet trajectory

more than the raw CenterTrack3D output, as shown in Table 4.2. This additional error is present because the Kalman filter predicts the next position of the pedestrian using the mathematical model it has created based on previous observations in addition to the current observation, factoring in sensor noise. The result is a much smoother estimated trajectory and a much more accurate velocity estimation. However, this comes at the cost of lagging behind the actual pedestrian location more closely predicted by the raw CenterTrack output. Of course, the Kalman filter also provides the additional benefit of being able to predict position continuously even in the presence of occlusions, which is the primary benefit and reason it is used.

Table 4.2: Single Pedestrian Tracking Test

Test Trial	Average Displacement(meters)
Without Kalman Filter	0.362
With Kalman Filter	0.399

4.1.3 Multi-Pedestrian Tracking

The performance of the object tracking with Kalman filtering was evaluated using a more difficult test involving the simultaneous tracking of 5 pedestrians. The pipeline was tasked with tracking each of the 5 pedestrians simultaneously and predicting their

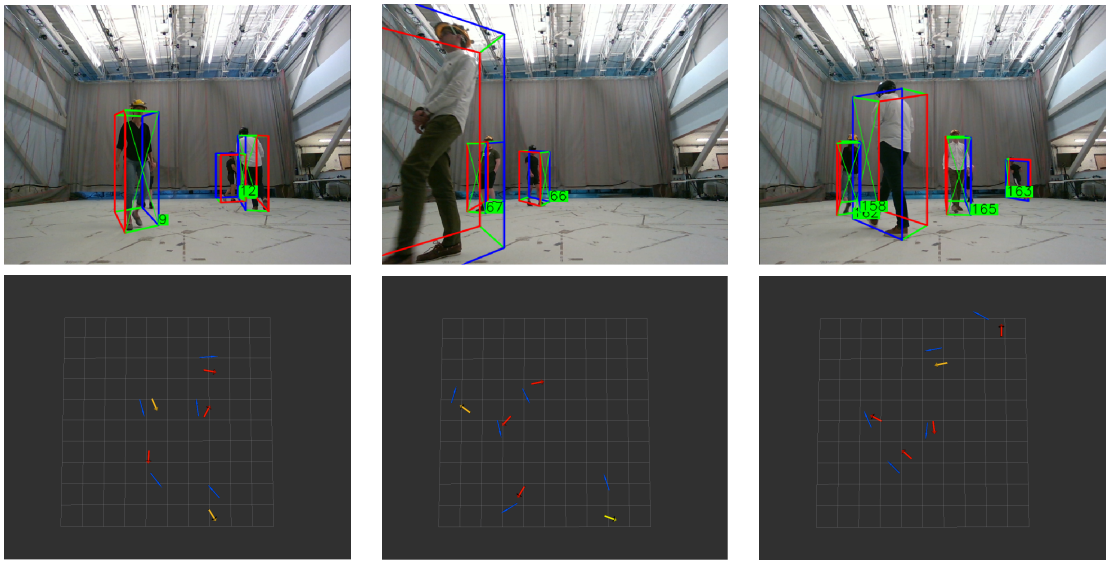


Figure 4-3: Selected samples from multiple pedestrian object tracking test. Top row: CenterTrack output. Bottom row: Birds eye view of Kalman tracklet estimated pedestrian positions

trajectories over time. Selected qualitative results of this test are shown in figure 4-3. In this figure, the upper images are the raw annotated output of the CenterTrack model(note: not the Kalman filter output), and the bottom images is the visualized estimated and ground truth positions of all pedestrians. The blue arrows represent the ground truth positions of each pedestrian, and the other arrows represent the estimated positions of the pedestrians. The scale of the grid is 1 m for every unit of the grid. The arrow color indicates the state of the tracklet in each case: red for tracked tracklets, orange for lost tracklets, and yellow for lingering tracklets. Tracked tracklets correspond to tracklets that can be matched to a CenterTrack 3D annotation on every frame(in view of the camera), lost tracklets are not matched to a CenterTrack annotation and their location is being predicted according to the Kalman filter(likely due to an occlusion), and lingering tracklets exist when a lost tracklet leaves the field of view in an attempt to predict where pedestrians that leave the field of view are generally. All 3 of these states are passed to the CADRL navigation module to provide greater levels of continuity compared to when only in the tracked state.

The results of the multi-pedestrian tracking test are summarized in Tables 4.3 and 4.4. Table 4.3 summarizes the average displacement for each tracklet type, and

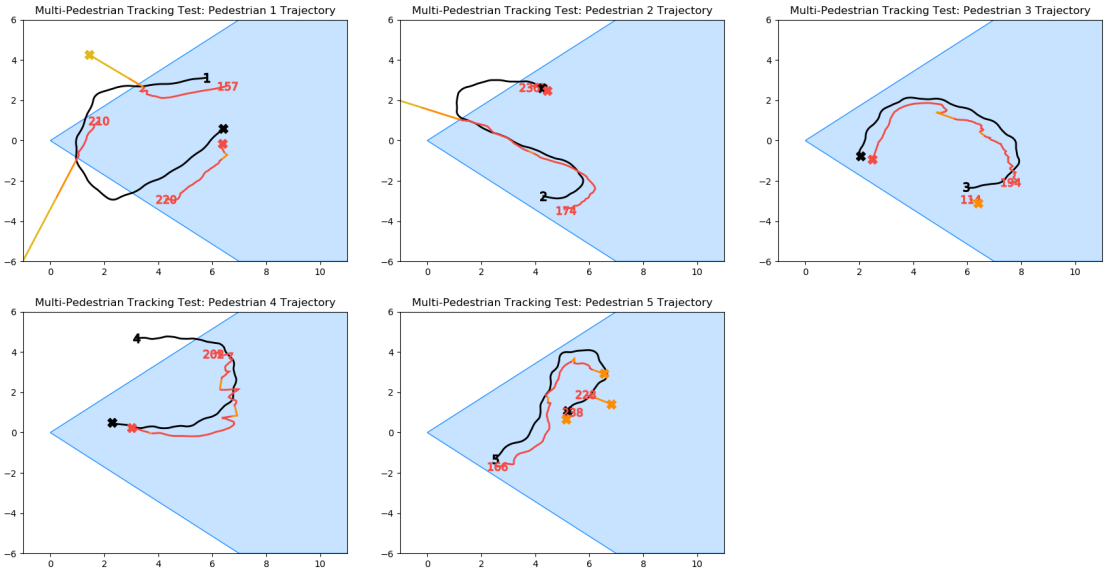


Figure 4-4: Multi-Pedestrian Tracking Test for 30-45 s: Ground Truth (black) vs Tracklet Trajectory

the number of occurrences for each tracklet type that can be matched against an actual pedestrian in the ground truth (not counting duplicate tracklets). The average displacement of tracklets in this multi pedestrian case was greater than the single pedestrian case due to the decreased depth accuracy when detections are close to each other in the image frame and the increased distance associated with relying on Kalman filter predictions in the presence of occlusions. In spite of this increased difficulty, the average displacement error is only 0.771 m when averaged among all cases and 0.648 m when just considering tracklets in the tracked state.

Selected pedestrian trajectories and their matching tracklet trajectories are visualized in figure 4-4 and 4-5. In these visualizations, each graph plots the ground truth trajectory of 1 pedestrian and all tracklets associated with that pedestrian. The light blue area represents the field of view of the camera, the black lines represent the ground truth trajectory of that pedestrian, and the other colored lines represent the tracklet(s) matched to that pedestrian. The tracklet path color corresponds to the state of that tracklet when it was estimated at that position: red for the tracked state, orange for the lost state, and yellow for the lingering state. For each tracklet path, a number is placed at the starting point of the tracklet. This number correlates

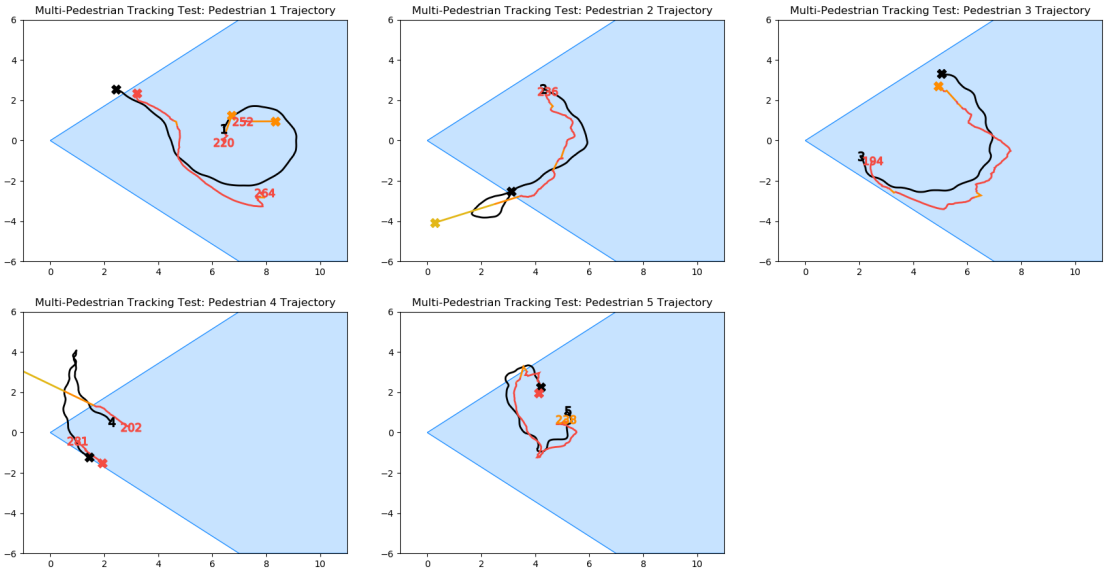


Figure 4-5: Multi-Pedestrian Tracking Ground Truth vs Tracklet Trajectory Test (Time 45-60 s)

to the ID assigned to that tracklet.

Figure 4-4 showcases the effectiveness of the tracking method visually. During this 15 second period of time, the position of each of the pedestrians is detected effectively while they are within the field of view of the camera, as shown by the strong correlation between the red tracklet paths and black ground truth paths. In addition, the Kalman filter is able to successfully perform Re-IDs in spite of minor occlusions. This is shown in the plots for Pedestrian 3 and Pedestrian 4. For both pedestrians, the tracklet path turns orange, indicating the tracklet has transitioned to the lost state, before it turns back to red, indicating a correct Re-ID has been performed. However, this Re-ID process is not perfect. For pedestrian 5, 3 different tracklet ids are assigned to pedestrian 5 during this period. However, this Re-ID failure is due to heavy occlusion by pedestrians 1, 2, and 4, all of which find themselves in between the camera and pedestrian at around the same time. While this ID switching is undesirable, the tracking performs well even in this challenging case by maintaining high level of continuity for the other pedestrians and by still estimating the position of Pedestrian 5 semi-accurately in spite of high occlusion.

Figure 4-4 also shows the use of the lingering state in the case of pedestrians 1

Table 4.3: Multiple Pedestrian Tracking Test - Average Displacement by Tracklet State

Tracklet State	Average Displacement(meters)	Number of Occurrences
Tracked	0.648	4847
Lost	0.798	1177
Lingering	1.724	589
All States	0.771	6613

and 2, visualized by the yellow portions of the tracklet paths that occur outside the field of view. These tracklet paths show how the lingering state is used to extend the lifespan of the tracklet beyond the field of view in an attempt to conservatively estimate that a pedestrian is likely still continuing in that direction. In the case of pedestrian 1, the pedestrian returns to the field of view instead of continuing on its current trajectory, resulting in two alive tracklets corresponding to the same ground truth pedestrian(IDs 157 and 210). While this duplicate tracklet is undesirable, these lingering tracklets discourage the planning module from turning to the sides when pedestrians move into its blind spots. With the limited field of view set up in this thesis, we find that this lingering state tracklet extension is beneficial in this limited FOV navigation setting.

Figure 4-5 presents another set of pedestrian paths and tracklet paths during a second 15 second period. Similar to the previous example, the tracking scheme shows it is effectively able to track pedestrians in spite of occlusions is in the case of pedestrians 2, 3, and 5 while suffering from id switching and detection missing in the most severe cases of severe occlusion as shown in the case of pedestrian 1. While this scheme is still unable to provide 100% coverage of all pedestrians in spite of occlusion, the tracking scheme presented here provides high accuracy and precision, except in cases of the most severe occlusion where any tracking method would naturally struggle.

Table 4.4 summarizes the number of occurrences of matched and unmatched tracklets. In this case, Matched tracklets are defined as tracklets that can be matched 1:1 with ground truth pedestrians. Since there is a possibility that more tracklets will be present at any given frame than the number of pedestrians (due to duplicate tracklets that may occur when a tracklet enters the Lost/Lingering state and is not Re-

Table 4.4: Multiple Pedestrian Tracking Test - Unmatched Tracklet Evaluation

Tracklet Type	Number of Occurrences	Percentage of Tracklets
Unmatched - Tracked	132	1.79
Unmatched - Lost	67	0.91
Unmatched - Lingering	551	7.48
Unmatched - All States	750	10.17
Matched - All States	6613	89.81
Total	7363	100.00

IDed correctly when that pedestrian reappears), some tracklets must be unmatched, meaning they are an unnecessary duplicate tracklet that resulted from a failed Re-ID attempt. Thus, the ideal number of unmatched tracklets is zero. 89.81% of tracklets were matched, meaning only 10.17% of tracklets were unmatched or resulted from an error. However, the majority of these unmatched tracklets were in the lingering state, which only occurs when a tracklet is outside the estimated field of view. The purpose of these tracklets is to extrapolate the estimated position of a tracked pedestrian that has left the camera's FOV for longer such that the rover does not forget that a pedestrian was just there that they can no longer see. In the case of this particular test, with pedestrians leaving and re-entering the field of view frequently, it is not surprising that we see a large number of unmatched lingering tracklets. Additionally, since lingering tracklets exist outside the field of view, they have less impact on the trajectory outputs of the navigation module, and are thus more acceptable.

These results showcase the trade-off associated with keeping tracklets alive for longer periods of time when outside the field of view in the lingering state. On one hand, utilizing this tracking scheme allows the pipeline a longer memory of pedestrian it has recently seen, which is particularly useful when pedestrians are walking past the rover or when the rover is turning. However, these pedestrians aren't able to be localized accurately in these cases, with an average displacement of 1.724 m. Additionally, these lingering tracklets account for the majority of unmatched tracklets that have the potential to confuse the navigation stack and result in trajectories that don't make sense. However, in practice we observe that because these inaccuracies occur outside the field of view of the rover, these significant error are acceptable and

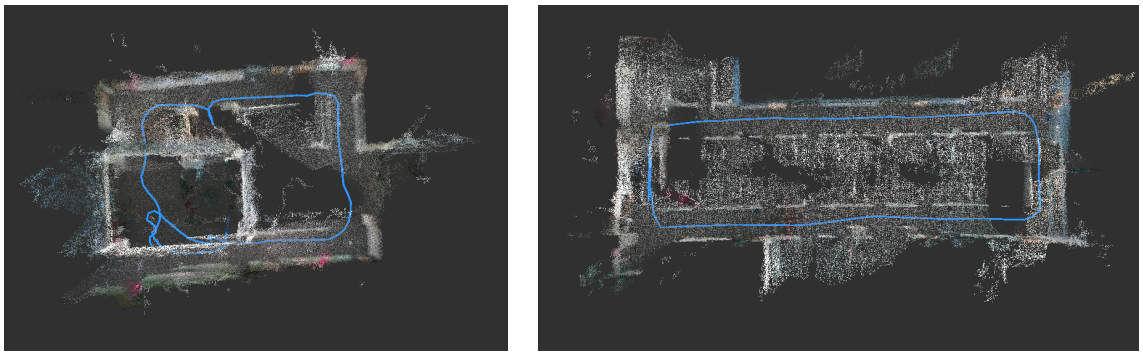


Figure 4-6: RTABMap Evaluation tests

Table 4.5: RTABMap Evaluation tests

Test Trial	Actual Dimensions (meters)	Estimated Dimensions (meters)	Percent Error
Kresa Center 2	12.2 x 9.0	12.50 x 9.19	2.28
Kresa Center 2R	24.7 x 7.9	24.20 x 7.67	2.53

result in more stable and continuous trajectories on average.

4.2 Mapping Evaluation

The performance of RTABMap in combination with the T265 tracking camera was evaluated in a series of indoor trials where the pipeline was manually driven in two indoor loops of varying sizes. In particular, these two indoor loop tests where done in two office areas in MIT building 31 on floor 2 and floor 2R. The maps generated by these tests can be seen in figure 4-6. The size of these maps were then compared against the actual dimensions of the building in question extracted from the building floor maps. The results of this comparison can be seen in table 4.5.

In addition to these indoor test trials, RTABMap was demonstrated to be able to correctly localize and map over long distances in a long running trial over the course of nearly 7 minutes from one indoor area, through an outdoor area, and into another indoor area (see figure 4-7). This approach is able to generate and localize over such distances without failure and in the presence of significantly different environments (indoor, restricted environments and sunny, open, outdoor environments).

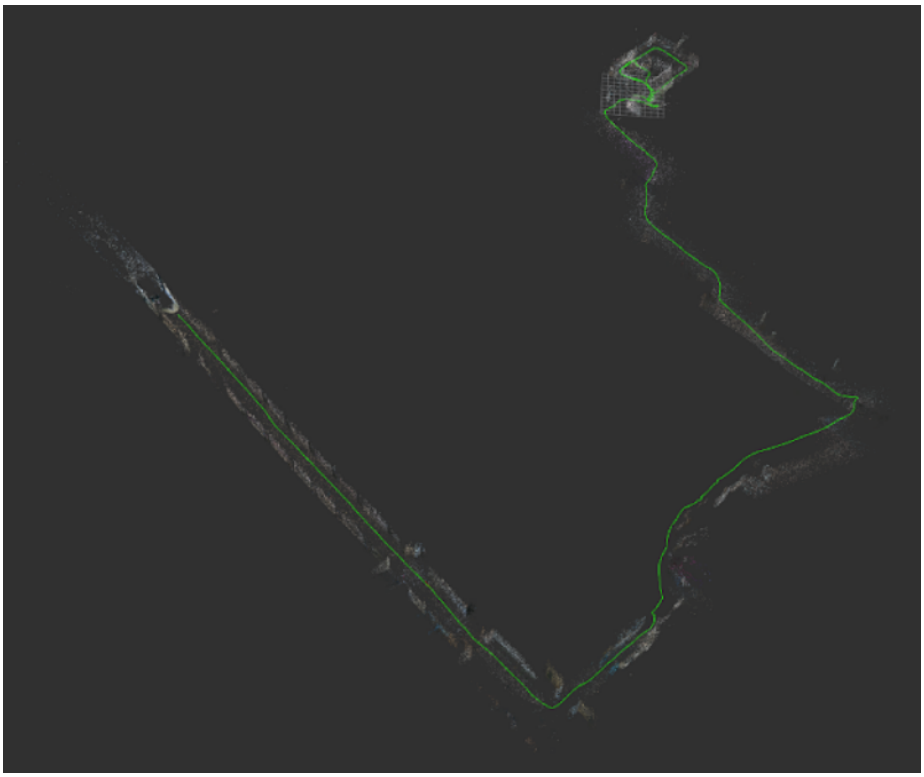


Figure 4-7: RTABMap Long Running Test

4.3 Full Navigation Tests

The performance of the full navigation stack was tested in a series of single-pedestrian use case tests and a series of multi-pedestrian “stress tests" to test the performance of the navigation stack in the presence of many pedestrians. Both types of tests were performed in the High Bay Space (Kresa Center for Autonomous Systems), which is an open area of about 11 m x 11 m. In both cases, the rover was tasked with navigating autonomously 10 m forward across the high bay space. Motion capture (Vicon) data was collected as ground truth to compare the estimated positions of the rover and estimated position of each of the pedestrians.

4.3.1 Single Pedestrian Test Cases

This section qualitatively evaluates the performance of the full navigation stack in controlled conditions with a single pedestrian moving in a simple fashion across the high bay. These tests are visualized in figures 4-8, 4-9, 4-10, and 4-11. In each case,

three visualizations are presented sequentially from left to right. In the visualizations, the coordinate frames represent the ground truth of the rover (RR03), ground truth of the pedestrian (PED1) as collected using the offboard Vicon tracking system, and the world frame origin(world). In these visualizations, each grid square corresponds to 1 m. Additionally, the estimated position of the rover is shown by the blue dot/arrow, and the estimated position of the pedestrian is shown by the red arrow (when present, indicates that the Kalman Bridge Node predicts that a pedestrian is there).

The pipeline was first demonstrated in the simple case of a stationary pedestrian, as shown in figure 4-8. In this instance, the pipeline detects the pedestrian, turns to give a wide berth to the pedestrian, and continues on a smooth trajectory to the goal.

The pipeline was then tested against an oncoming pedestrian moving directly towards the rover in figure 4-9. The rover swerves out of the way to avoid the pedestrian, then turns back around and continues to the goal. In this case, the rover makes a significantly larger turn than necessary, turning 180 degrees, continuing for a brief period, and then turning around. CADRL utilizes reinforcement learning to guide trajectories, so it is difficult to determine why this sub-optimal behavior occurs, but it could be in an attempt to "run away" from an agent that is coming directly towards it.

The pipeline was then tested against a single pedestrian crossing in front of the rover, but not obstructing it in figure 4-10. In this case, the pedestrian walks from right to left in front of the rover, and the rover is able to correctly predict that the pedestrian poses no threat to the rover and continues on a mostly straight trajectory.

In the final test case, the pedestrian approaches the rover diagonally across the rover's desired path from right to left in figure 4-11. The robot first attempts to go left, but then recognizes the best path is to allow the pedestrian to pass and goes to the right of the pedestrian instead.

4.3.2 Multi-Pedestrian Stress Testing

The rover was tested in the same task of navigation across the MIT AeroAstro High Bay Space (31-255) through a group of 5 pedestrians moving around the space. The

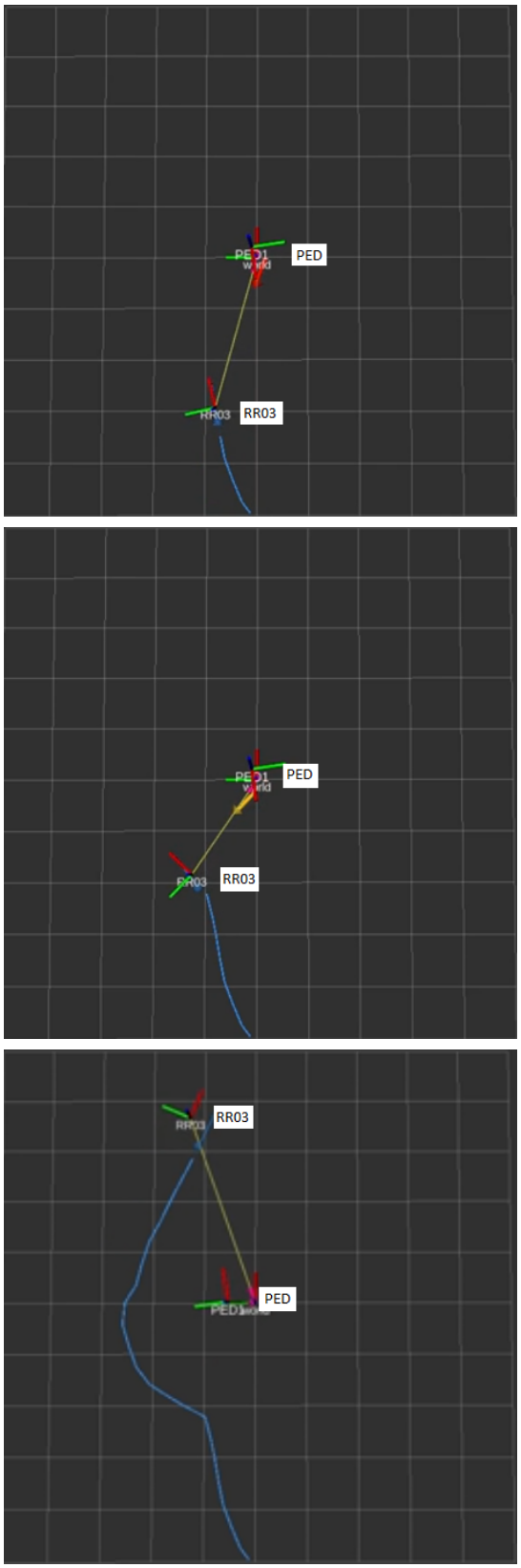


Figure 4-8: Stationary pedestrian test

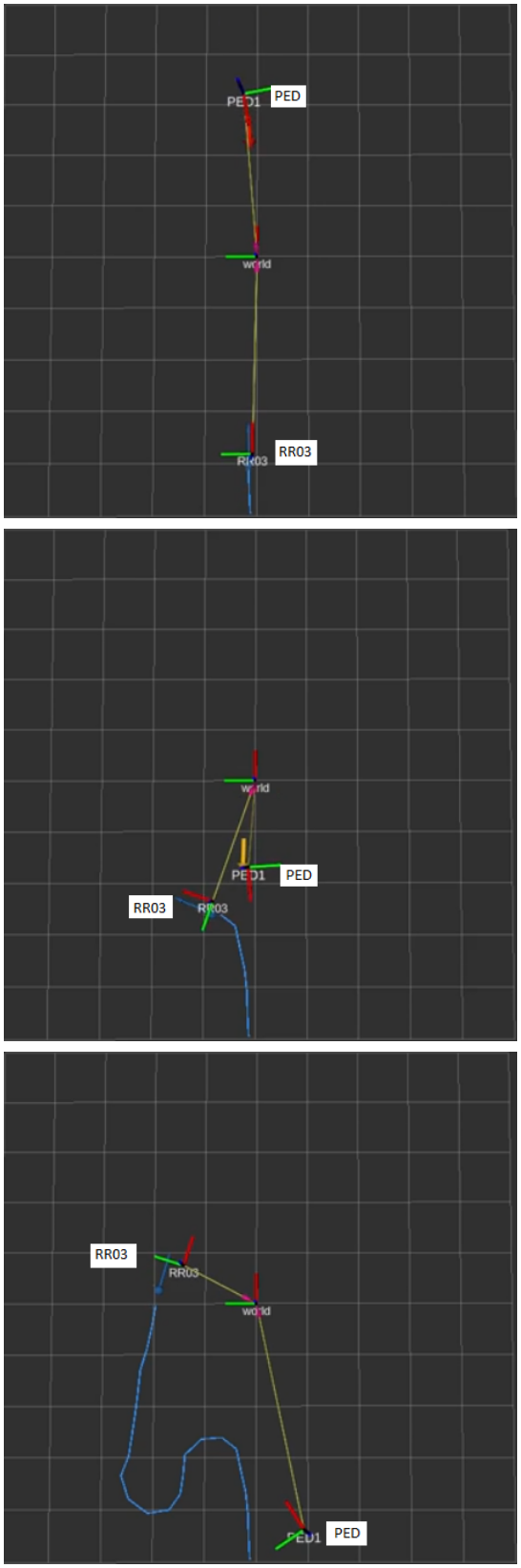


Figure 4-9: Oncoming pedestrian test

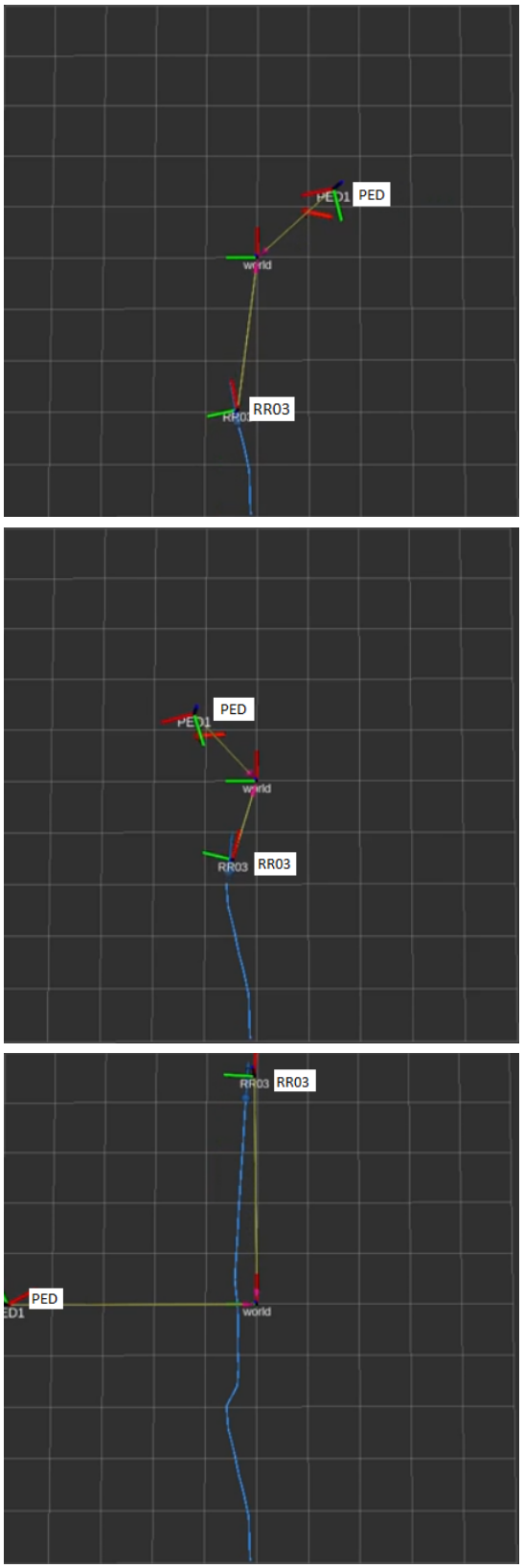


Figure 4-10: Crossing pedestrian test

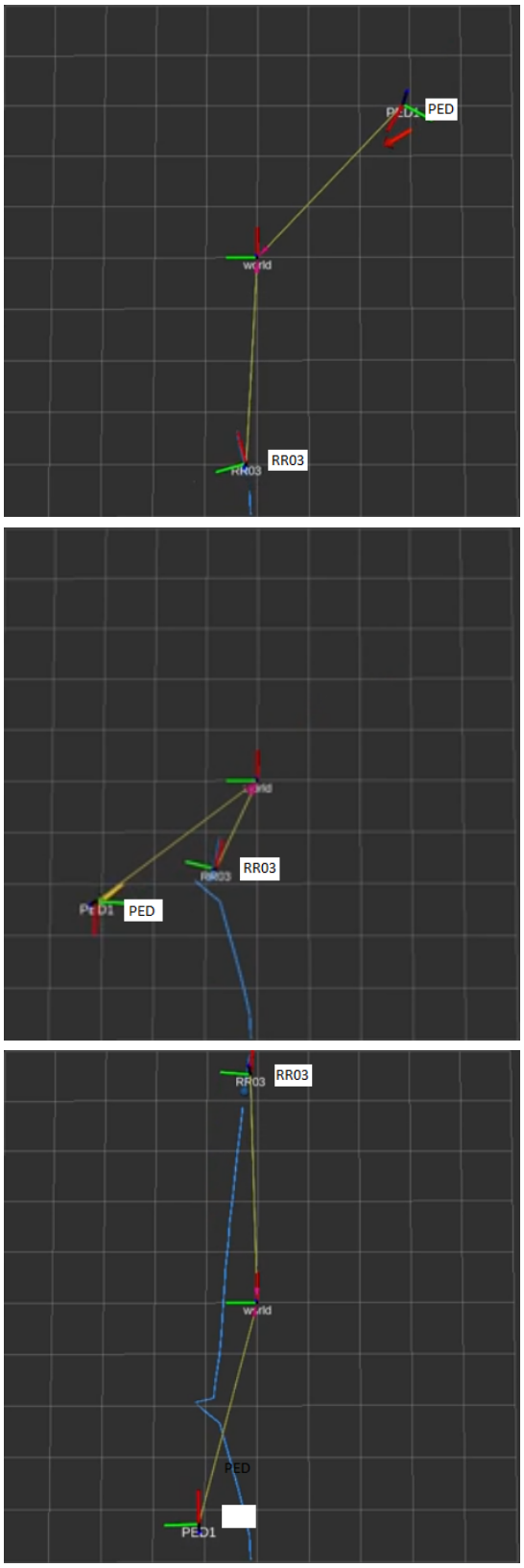


Figure 4-11: Diagonal pedestrian test

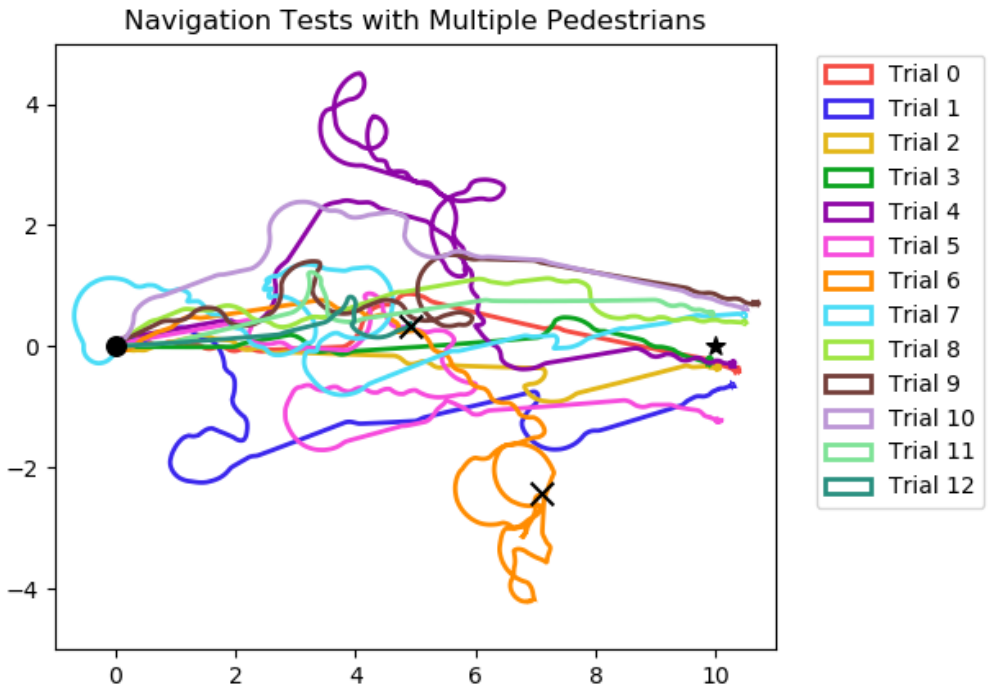


Figure 4-12: All Paths on the multi-pedestrian stress testing cases with navigation through a crowd of 5 pedestrians. The circle indicates starting point, star indicates the goal, and crosses indicate points of failure.

goal of these tests were to stress test the system against a real-world potential use case of a crowded pedestrian environment in which pedestrians will be moving in all directions, will face high levels of occlusion, and will be continuously moving in and out of frame.

In trials 1-8, the 5 pedestrians were instructed to move randomly between 5 waypoints places on the left and right sides of the room. In trials 9-13, the pedestrians were instructed to roam randomly around the center of the room. The resulting trajectories are shown in figure 4-12. The trajectories for each trial individually can be seen in figure 4-13.

In 2 of the 13 trials the rover was not able to successfully navigate across the room. The first of these cases was trial 7, in which the rover made a significant course correction to the right and made contact with the wall of the room and was unable to navigate correctly to the goal. This occurred because the navigation stack

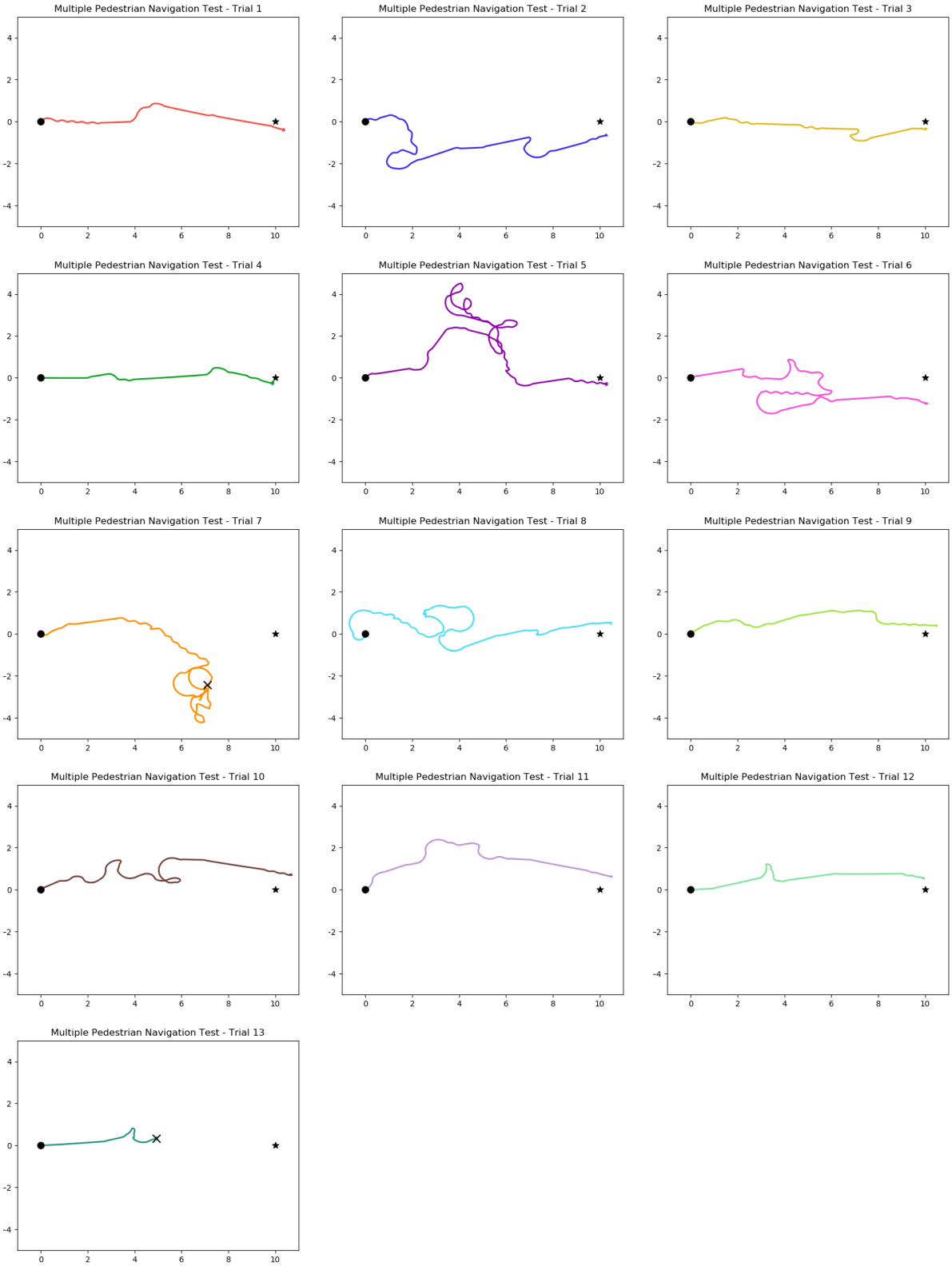


Figure 4-13: Paths taken by the rover through crowded 5 pedestrian environments. Each trial visualized as a different color

used in this pipeline does not consider static obstacle avoidance for this test. The second unsuccessful trial was trial 13, the rover made a significant turn to the right, before turning left again to head towards the goal. This quick turn made it unable to see a pedestrian that was heading directly toward the robot, and it then collided with that pedestrian. The trajectories for these cases can be seen visualized in figure 4-13.

In 11 of the 13 trials the rover was able to successfully navigate 10 m forward to the other side of the room to the vicinity of the goal. In the 11 successful cases, the rover was able to navigate to the goal in an average of 38.9 s at a speed of 0.5 m/s. The rest of this section will go into detail into two successful navigation trails: Trial 9 and Trial 4. In Trial 9, the rover was able to successfully navigate a challenging situation in only 28.4 s after making a series of minor corrective adjustments to avoid collisions while making progress to the goal. In Trial 4, the rover attempted to take a trajectory to the left of a group of pedestrians in the center of the room, only to find itself stuck in a challenging position when those pedestrians moved in front of the rover, blocking its path to the goal. The rover then takes several seconds unsuccessfully attempting to find a path around the pedestrians, before an opening occurs and the rover then navigates to the goal in a total of 72.3 s.

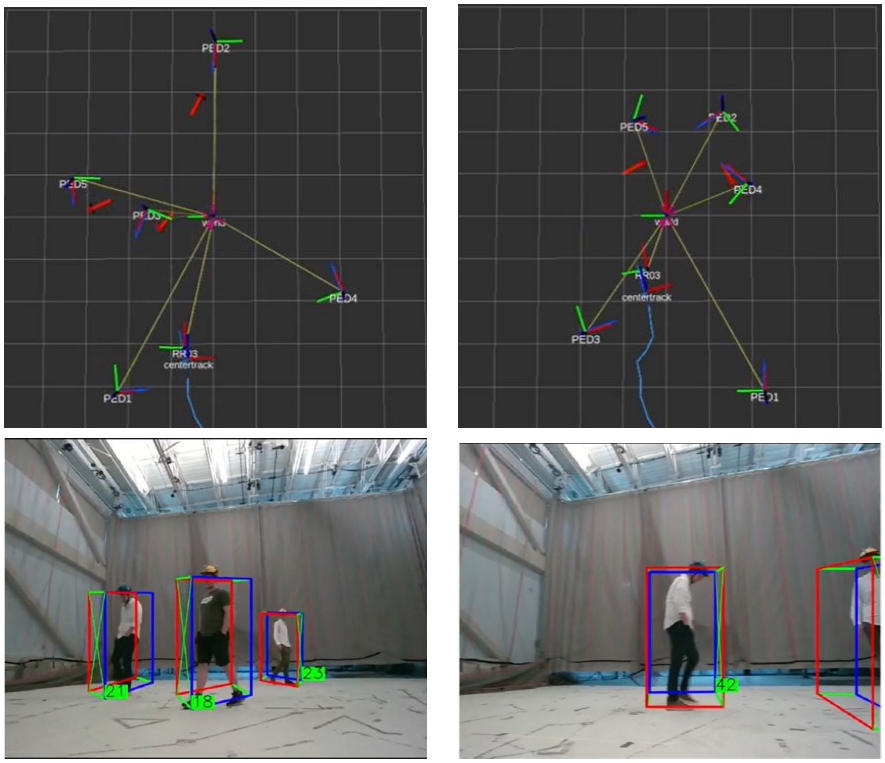
Trial 9

The behavior of the navigation stack in trial 9 is shown in the series of camera images and BEV visualizations shown in Figure 4-14. The figure visualizes 4 pairs of images representing different timesteps labeled A-D. At timestep A, the rover finds its direct path blocked by three pedestrians in between it and the goal. By timestep B, the rover chooses to navigate in between the pedestrians, making it past one pedestrian and finding the other two pedestrians moving to its right past the goal. The rover then continues towards the goal at timestamp C, choosing to follow behind the two pedestrians as they move to the right. In timestep C, another pedestrian crosses in front of the rover. The rover recognizes that the pedestrian is crossing its path and no course correction is necessary, choosing to continue its path to the goal, reaching

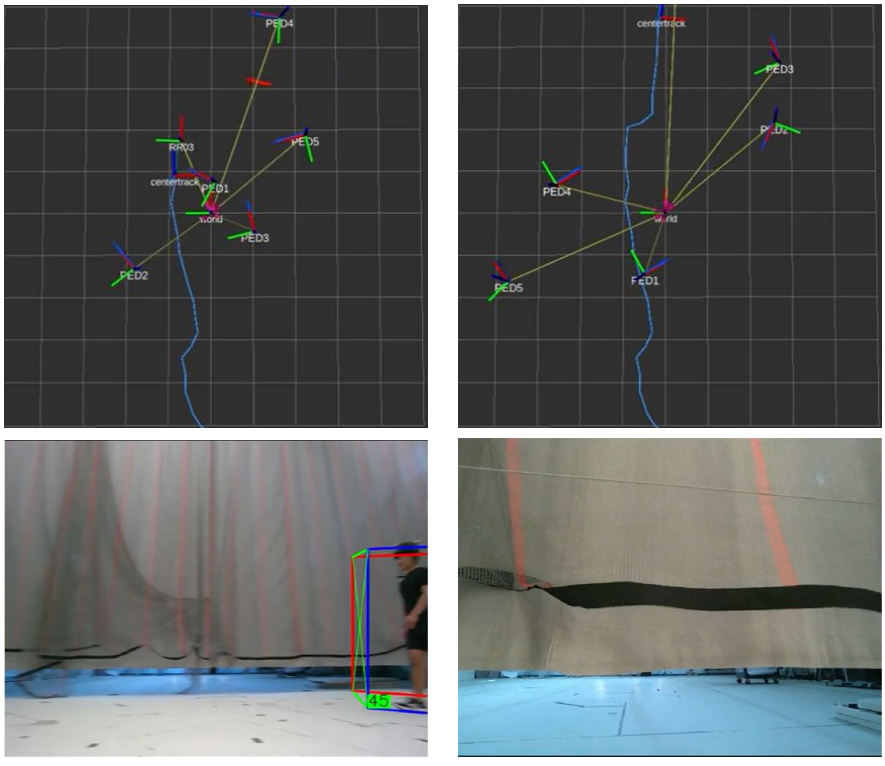
the goal by timestep D in a time of 28.4 s. This is an example of one of the successful trials that was able to safely and efficiently navigate across the crowded space my making minor but effective trajectory adjustments.

Trial 4

The behavior of the navigation stack in trial 4 is shown in the series of camera images and BEV visualizations shown in Figure 4-15. This trial proved challenging to the navigation stack, with roaming pedestrians in between the rover and the goal resulting in multiple loops as the rover attempts to find a path. In timestep A, the rover tracks a series of pedestrians in the middle of the room, and chooses to navigate to the left of the pedestrians. By timestep B, the rover passes the group of pedestrians in the center, and attempts to make a right turn past another pedestrian which has moved to the back left of the room. However, after turning right in response to the pedestrian from timestep B, the rover discovers 2 more pedestrians crossing in front of its direct path from the right. In response, the rover continues its right turn and retreats further back in the room in the opposite direction of the goal. The rover then turns back towards the goal, finding that the path to the goal is still crowded with the pedestrians from timestep C. In response, the rover performs another loop, before turning again and attempting to make progress to the goal. In timestep E, the tracking system identifies the two pedestrians previously occupying the path to the goal moving to the other side of the room, and identifies a safe path, reaching the goal by timestep F. This winding route contained many unnecessary loops, causing the rover to take 72.3 s to reach the goal. While this is a very long amount of time to traverse 10 meters, it occured because the rover chose a path to the left which wound up being a difficult spot with 3 pedestrians roaming the space between the rover and the goal. The rover was able to avoid collision, and quickly found a path to the goal once it became clear. Ideally, the rover would have been able to identify a safe path around the pedestrians rather than conservatively looping until the path was cleared. However, this strategy was primarily a result of the CADRL policy choosing to retreat in order to avoid a collision. It is likely that a more efficient route to the



A B



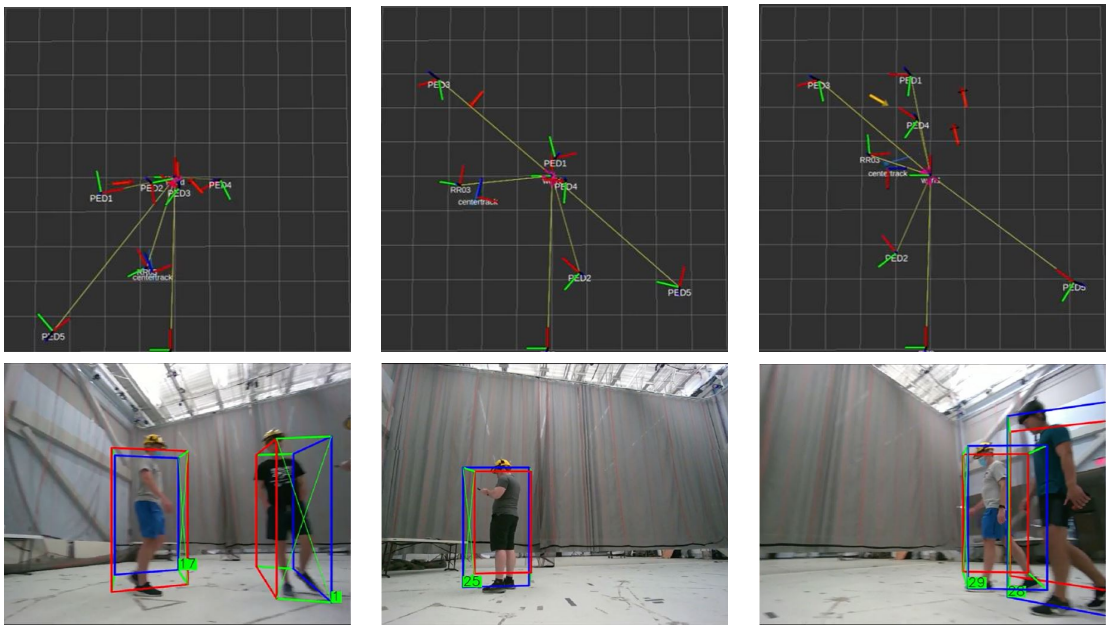
C D

Figure 4-14: Trial 9 Screenshots

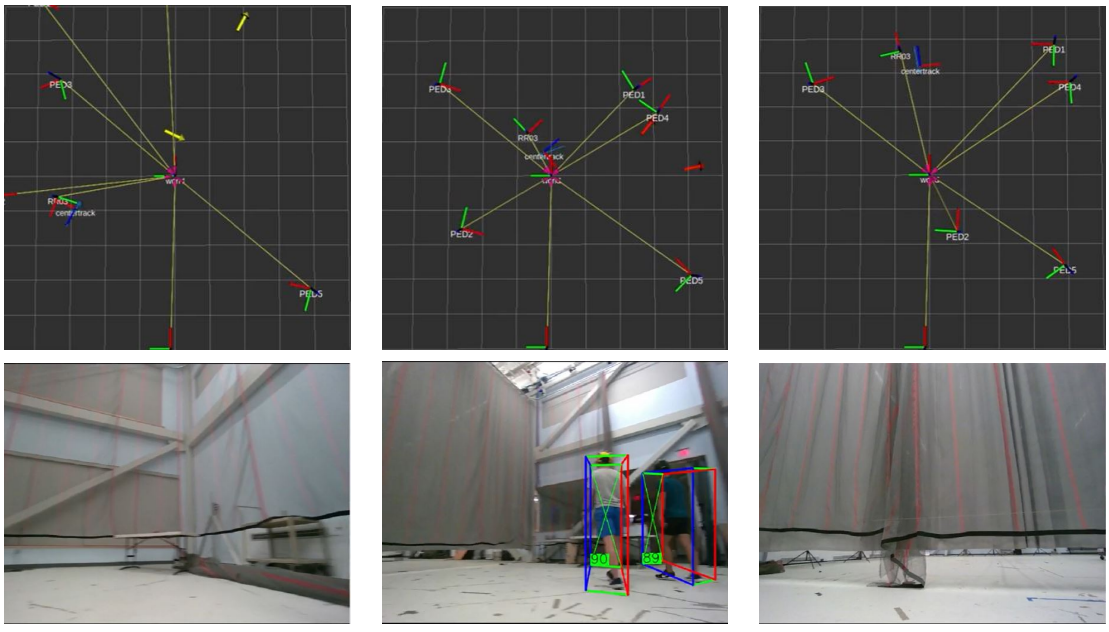
goal could be identified by a CADRL policy that was retrained on a dataset that uses a limited FOV, because it would be more likely to learn that simply retreating is an undesirable behavior in the limited FOV case.

Navigation Results Discussion

The end-to-end navigation system presented here shows many promising results in terms of 3D pedestrian tracking, localization/mapping, and navigation/planning. Despite using a very limited FOV and utilizing a CADRL policy trained on under a larger field of view, the navigation pipeline was able to identify and traverse paths through a challenging, crowded environment. The tracking scheme presented here enables strong performance in this case without explicit limited FOV training by utilizing a Kalman filter-based tracking scheme to infer information necessary for safe and efficient policy making by CADRL. In addition, the planning module was able to effectively navigate through the environment in 11 out of 13 cases, with only one pedestrian collision. It is likely that the pedestrian collision and undesirable looping behavior seen in trial 4 of the multi-pedestrian navigation tests could be avoided by training a new CADRL policy under limited FOV conditions where sharp turns and looping is less effective.



A B C



D E F

Figure 4-15: Trial 4 Screenshots

Chapter 5

Conclusion

5.1 Contribution

The work of this thesis presents a fully functional implementation of vision-based object detection, mapping, and navigation. Previous navigation methods for pedestrian environments rely on expensive sensor array including LiDaR, multi-camera arrays, or both. The navigation stack presented here is able to navigate effectively using only one forward-facing RGB-D camera and one forward-facing tracking camera, for a total cost of less than 700$ USD MSRP. The implementation presented in this thesis build off of previous work done at the Aerospace Controls Lab by integrating state of the art 3D monocular object detection methods, utilizing a robust FOV-aware kalman filter tracking approach, and by integrating full SLAM for more accurate localization predictions with loop closures.

The object tracking methodology presented here utilizes 3D monocular object detection using CenterTrack3D combined with a Kalman Filter based "bridge node" on top that enables accurate trajectory tracking, robustness to occlusion, and short-term Re-ID. The average 3D displacement is within .4 meters for single pedestrians and within .7 meters in challenging multi-pedestrian tracking experiments where high levels of occlusion are present.

Additionally, this pipeline utilizes a simultaneous localization and mapping(SLAM) solution using an intel RealSense T265 and RTABMap to localize itself and generate

maps with less than 2.5% error in real world tests. These dense maps provide useful information about static obstacles and can be used to perform loop closures.

Finally, these pedestrian detection and localization schemes were implemented in ROS and combined with CADRL navigation to navigate around dynamic obstacles using a reinforcement learning model. Despite being trained on a dataset with a significantly greater field of view, this implementation was demonstrated to be able to effectively navigate through crowded pedestrian environments with only one observed collision. The average final time to navigate through a crowded 10m by 10m space was 38.9 seconds at a speed of 0.5 meters/second.

5.2 Future Work

This thesis presented a vision-based navigation pipeline that demonstrates effective object tracking, localization, and navigation without having to fine tune object detection or navigation models on the specific hardware used for navigation. However, fine-tuned object detection models will likely increase the accuracy and consistency of detected 3D annotations. Additionally, the CADRL policy used to test full navigation was sufficient in most cases, but many of the movements triggered by the CADRL policy made turns which were too quick or showed a preference to turning around and moving in the opposite direction of nearby pedestrians instead of attempting to navigate through them. Part of the reason for this is that the policy interprets the lack of detections outside the field of view as there being free space to navigate through, when in reality it only means there are no pedestrians it can see. Ideally, a specifically trained navigation policy will be trained on a limited-FOV dataset to train the agent to avoid making dangerous moves into areas it cannot see.

Accurate and consistent depth estimation continues to be a challenge for vision-based object detection methods. In contrast to LiDaR, where depth is directly measured and available, vision-based methods for deriving depth is an inherently ill-posed problem because images only provide 2D RGB data and depth must be calculated using other means. This poses a significant challenge to vision-based navigation due

to the increased uncertainty in 3d localization. 3D monocular object detection has demonstrated that it is possible to extract object depth values using learned neural network models, but these estimates are not as accurate or consistent as other methods such as LiDaR. Stereo methods and depth cameras pose a promising alternative to achieving more accurate depth measurements, but these methods are not yet effective at long ranges and are also not as accurate at moderate ranges compared to LiDaR. Recently, some success has been experienced in investigation into pseudo-lidar approaches that could present a promising alternative to traditional monocular object tracking [13] [18]. These pseudo-lidar methods convert depth maps(either extracted from RGB-D cameras or learned models) into a lidar-like pointcloud and perform 3D object detection over this pointcloud data instead of the 2D image data. Future work to refine the accuracy of 3D vision-based object detection, whether pseudo-lidar or other methods, continues to be a requirement for more accurate vision-based navigation.

This thesis presented a navigation scheme that is able to navigate around dynamic agents. However, full deployment of vision-based navigation systems requires navigation around both static and dynamic obstacles. This pipeline collects and generates an occupancy map that could be utilized by such a policy, but lacks an available ROS implementation of a policy to enable simultaneous static and dynamic obstacle avoidance. Future work in this field should consider creation of navigation methods(implemented in ROS) than can incorporate available static information in addition to dynamic pedestrian data, such as Go-MPC.

In addition to navigation around static obstacles, practical deployment of vision-based navigation will be able to detect other classes of dynamic obstacles. Examples include traditional obstacles like cars, but also unexpected obstacles like carts or animals. Since it is impractical to run learned object detection on all possible classes of objects, it is desirable to be able to perform general moving object detection and tracking which does not require neural network training.

Future work in similar end-to-end pipelines should focus on improved performance of 3D monocular object detection through model fine tuning and/or the exploration

of pseudolidar models. In addition, future work should target the utilization of static information gathered in the form of maps and occupancy grids such as those generated by this pipeline in its navigation methods for static obstacle avoidance and global navigation.

Bibliography

[1] Lorenzo Bertoni, Sven Kreiss, and Alexandre Alahi. Monocular 3d pedestrian localiztion and uncertainty estimation. *ICCV*, 2019. arXiv:1906.06059.

[2] Lorenzo Bertoni, Sven Kreiss, Taylor Mordan, and Alexandre Alahi. Monstereo: When monocular and stereo meet at the tail of 3d human localization. In *the International Conference on Robotics and Automation (ICRA)*, 2021.

[3] Bruno Brito, Michael Everett, Jonathan P. How, and Javier Alonso-Mora. Where to go next: Learning a subgoal recommendation policy for navigation in dynamic environments. *IEEE Robotics and Automation Letters*, 6(3):4616–4623, 2021.

[4] Holger Caesar, Varun Bankiti, Alex H. Lang, Sourabh Vora, Venice Erin Liong, Qiang Xu, Anush Krishnan, Yu Pan, Giancarlo Baldan, and Oscar Beijbom. nuscenes: A multimodal dataset for autonomous driving. *CVPR*, 2020.

[5] Carlos Campos, Richard Elvira, Juan J. Gomez Rodriguez, Jose M. M. Montiel, and Juan D. Tardos. ORB-SLAM3: An accurate open-source library for visual, visual–inertial, and multimap SLAM. *IEEE Transactions on Robotics*, 37(6):1874–1890, dec 2021.

[6] Michael Everett, Yu Fan Chen, and Jonathan P. How. Motion planning among dynamic, decision-making agents with deep reinforcement learning. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3052–3059, 2018.

[7] David Gale and Lloyd Shapley a. College admissions and the stability of marriage. *The American Mathematical Monthly, Vol. 69*, 1962.

[8] Patrick Geneva, Kevin Eckenhoff, Woosik Lee, Yulin Yang, and Guoquan Huang. Openvins: A research platform for visual-inertial estimation. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4666–4672, 2020.

[9] Zechen Liu, Zizhang Wu, and Roland Tóth. SMOKE: Single-stage monocular 3d object detection via keypoint estimation. *arXiv preprint arXiv:2002.10111*, 2020.

[10] Roberto Martin-Martin, Mihir Patel, Hamid Rezatofighi, Abhijeet Shenoi, Junyoung Gwak, Eric Frankel, Amir Sadeghian, and Silvio Savarese. JRDB: A dataset and benchmark of egocentric robot visual perception of humans in built environments. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 1–1, 2021.

[11] Susan Ni. Hardware implementation of a complete vision-based navigation pipeline. Master's of engineering thesis, Massachusetts Institute of Technology. September 2020 - August 2021.

[12] Jiangmiao Pang, Linlu Qiu, Xia Li, Haofeng Chen, Qi Li, Trevor Darrell, and Fisher Yu. Quasi-dense similarity learning for multiple object tracking. *CVPR*, 2021. arXiv:2006.06664.

[13] Dennis Park, Rares Ambrus, Vitor Guizilini, Jie Li, and Adrien Gaidon. Is pseudo-lidar needed for monocular 3d object detection?, 2021.

[14] Tong Qin, Peiliang Li, and Shaojie Shen. Vins-mono: A robust and versatile monocular visual-inertial state estimator. *IEEE Transactions on Robotics*, 34(4):1004–1020, 2018.

[15] Tong Qin and Shaojie Shen. Online temporal calibration for monocular visual-inertial systems. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3662–3669. IEEE, 2018.

[16] Yun Su, Ting Wang, Chen Yao, Shiliang Shao, and Zhidong Wang. Gr-slam: Vision-based sensor fusion slam for ground robots on complex terrain. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5096–5103, 2020.

[17] Evgeny Tsykunov, Valery Ilin, Stepan Perminov, Aleksey Fedoseev, and Elvira Zainulina. Coupling of localization and depth data for mapping using intel realsense t265 and d435i cameras, 2020.

[18] Xinshuo Weng and Kris Kitani. Monocular 3d object detection with pseudo-lidar point cloud. In *IEEE International Conference on Computer Vision (ICCV) Workshops*, Oct 2019.

[19] Xingyi Zhou, Vladlen Koltun, and Philipp Krähenbühl. Tracking objects as points. *ECCV*, 2020. arXiv:2004.01177.