12-31-2022

Design Framework of UAV-Based Environment Sensing, Localization, and Imaging System

Yue Sun
University of Massachusetts Boston

DESIGN FRAMEWORK OF UAV-BASED ENVIRONMENT SENSING, LOCALIZATION, AND IMAGING SYSTEM

A Dissertation Presented

by

Yue Sun

Submitted to the Office of Graduate Studies, University of Massachusetts Boston, in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

December 2022

Computer Science Program

DESIGN FRAMEWORK OF UAV-BASED ENVIRONMENT SENSING, LOCALIZATION, AND IMAGING SYSTEM

A Dissertation Presented

by

Yue Sun

Approved as to style and content by:

Honggang Zhang, Professor
Chairperson of Committee

Dan Simovici, Professor
Member

Marc Pomplun, Professor
Member

Xiaohui Liang, Associate Professor
Member

Bo Sheng, Associate Professor
Member

Dan Simovici, Program Director
Computer Science Program

Marc Pomplun, Chairperson
Computer Science Department

ABSTRACT

DESIGN FRAMEWORK OF UAV-BASED ENVIRONMENT SENSING, LOCALIZATION, AND IMAGING SYSTEM

December 2022

Yue Sun,
B.S., Sun Yat-Sen University
B.S., Hong Kong Polytechnic University
M.S., University of Hong Kong
M.S., University of Massachusetts Boston
Ph.D., University of Massachusetts Boston

Directed by Professor Honggang Zhang

In this dissertation research, we develop a framework for designing an Unmanned Aerial Vehicle or UAV-based environment sensing, localization, and imaging system for challenging environments with no GPS signals and low visibility. The UAV system relies on the various sensors that it carries to conduct accurate sensing and localization of the objects in an environment, and further to reconstruct the 3D shapes of those objects. The system can be very useful when exploring an unknown or dangerous environment, e.g., a disaster site, which is not convenient or not accessible for humans. In addition, the system can be used for monitoring and object tracking in a large scale environment, e.g., a smart manufacturing factory, for the purposes of workplace management/safety, and maintaining optimal system performance/productivity.

In our framework, the UAV system is comprised of two subsystems: a sensing and localization subsystem; and a mmWave radar-based 3D object reconstruction subsystem.

The first subsystem is referred to as LIDAUS (Localization of IoT Device via Anchor UAV SLAM), which is an infrastructure-free, multi-stage SLAM (Simultaneous Localization and Mapping) system that utilizes a UAV to accurately localize and track IoT devices

in a space with weak or no GPS signals. The rapidly increasing deployment of Internet of Things (IoT) around the world is changing many aspects of our society. IoT devices can be deployed in various places for different purposes, e.g., in a manufacturing site or a large warehouse, and they can be displaced over time due to human activities, or manufacturing processes. Usually in an indoor environment, the lack of GPS signals and infrastructure support makes most existing indoor localization systems not practical when localizing a large number of wireless IoT devices. In addition, safety concerns, access restriction, and simply the huge amount of IoT devices make it not practical for humans to manually localize and track IoT devices. Our LIDAUS is developed to address these problems. The UAV in our LIDAUS system conducts multi-stage 3D SLAM trips to localize devices based only on Received Signal Strength Indicator (RSSI), the most widely available measurement of the signals of almost all commodity IoT devices. Our simulations and experiments of Bluetooth IoT devices demonstrate that our system LIDAUS can achieve high localization accuracy based only on RSSIs of commodity IoT devices.

Build on the first subsystem, we further develop the second subsystem for environment reconstruction and imaging via mmWave radar and deep learning. This subsystem is referred to as 3DRIMR/R2P (3D Reconstruction and Imaging via mmWave Radar/Radar to Point Cloud). It enables an exploring UAV to fly within an environment and collect mmWave radar data by scanning various objects in the environment. Taking advantage of the accurate locations given by the first subsystem, the UAV can scan an object from different viewpoints. Then based on radar data only, the UAV can reconstruct the 3D shapes of the objects in the space. mmWave radar has been shown as an effective sensing technique in low visibility, smoke, dusty, and dense fog environment. However, tapping the potential of radar sensing to reconstruct 3D object shapes remains a great challenge, due to the characteristics of radar data such as sparsity, low resolution, specularity, large noise, and multi-path induced shadow reflections and artifacts. Hence, it is challenging to reconstruct 3D object shapes based on the raw sparse and low-resolution mmWave radar signals.

To address the challenges, our second subsystem utilizes deep learning models to extract features from sparse raw mmWave radar intensity data, and reconstructs 3D shapes of

objects in the format of dense and detailed point cloud. We first develop a deep learning model to reconstruct a single object's 3D shape. The model first converts mmWave radar data to depth images, and then reconstructs an object's 3D shape in point cloud format. Our experiments demonstrate the significant performance improvement of our system over the popular existing methods such as PointNet, PointNet++ and PCN. Then we further explore the feasibility of utilizing a mmWave radar sensor installed on a UAV to reconstruct the 3D shapes of multiple objects in a space. We evaluate two different models. Model 1 is 3DRIMR/R2P model, and Model 2 is formed by adding a segmentation stage in the processing pipeline of Model 1. Our experiments demonstrate that both models are promising in solving the multiple object reconstruction problem. We also show that Model 2, despite producing denser and smoother point clouds, can lead to higher reconstruction loss or even missing objects. In addition, we find that both models are robust to the highly noisy radar data obtained by unstable Synthetic Aperture Radar (SAR) operation due to the instability or vibration of a small UAV hovering at its intended scanning point. Our research shows a promising direction of applying mmWave radar sensing in 3D object reconstruction.

ACKNOWLEDGEMENTS

First of all, I would like to express my sincere gratitude to my Ph.D. research advisor, Prof. Honggang Zhang, who is a professor of the engineering department. I am deeply grateful of his instructive advice and helpful suggestions to complete this dissertation. It is my fortune to have him not only as my study mentor but also as my life mentor, I may not complete my research and study without his constant encouragement and guidance. It is my great honor to work with him for four and half years of my Ph.D. study. From him, I learn responsibility, enthusiasm, professionalism and persistence for research. I believe all those good qualities and the experiences I gathered from him will also help me succeed in my future life.

I should also thank Prof. Dan Simovici, who is the program director of the computer science program. I took his courses *Theory of Formal Languages* and *Logical Foundations in Computer Science*. I really appreciate his clear and detailed explanation of so many difficult theories during the class, and I also learned a lot when we discussed some hard questions after class. And thanks to him for being willing to serve on my dissertation committee.

I would like to convey my appreciation to Prof. Marc Pomplun, who is the chairperson of the computer science department. I took many courses of him during my study at UMass Boston, and I learned so much basic and necessary knowledge about artificial intelligence and computer vision from his courses, like *Artificial Intelligence*, *Neural Networks*, and *Computer Vision*. These interesting courses also stimulate my enthusiasm for learning more AI-related knowledge and applying it to my own research. And thanks to him for being willing to serve on my dissertation committee.

Let me also express my appreciation to Prof. Xiaohui Liang, who is an associate professor in the computer science department. Many thanks to him for his advice and supports on my research study. His enthusiasm and hard work for research inspire me a lot. And also thanks to him for being willing to serve on my dissertation committee.

I also want to thank Prof. Bo Sheng, who is an associate professor in the computer science department. I took his course *Wireless Network*, and I gained a lot of related knowledge during the class. I also would like to thank him for his encouragement and suggestions on my research and study. His rigorous attitude to the teaching and research influences me a lot. And also thanks to him for being willing to serve on my dissertation committee.

Then let me express my sincere thankfulness to my colleagues, friends and classmates: Zhuoming Huang, Deqiang Xu, Dr. Zhi Cao, Xiaoqian Zhang, Haoyu Wang, Chenglin Ba, Huiyi Ding, Shuairui Yao, Xidan Zheng, Tiantian Xie, Bang Tran and many others. I truly appreciate their kind help and supports during my Ph.D. study.

Last but not least, I want to deliver my great gratitude to my family. Many thanks to my mother Meizhen Zhang, my father Jianjun Sun and my dog Luobo for their unlimited love, understanding, support and encouragement.

TABLE OF CONTENTS

LIST OF TABLES

Table Page

LIST OF FIGURES

Figure Page

CHAPTER 1

INTRODUCTION

We develop a framework for designing an Unmanned Aerial Vehicle or UAV-based system for conducting environment sensing, localization, and imaging in the environments that pose great challenges to exploration and investigation by traditional techniques, e.g., weak or no GPS signals, low visibility, or access difficulty for humans or mobile ground robots. To address those challenges, we introduce a UAV-based system that utilizes the various sensors mounted on a UAV and a collection of learning algorithms to localize and track the objects in a space and reconstruct their 3D shapes in point cloud format, despite no GPS signals and low visibility.

Our UAV system is comprised of two subsystems: a subsystem for sensing and localization, and a subsystem for 3D reconstruction and imaging of objects via mmWave radar. The first subsystem is referred to as LIDAUS (Localization of IoT Device via Anchor UAV SLAM), which is an infrastructure-free, multi-stage SLAM (Simultaneous Localization and Mapping) system that utilizes a UAV to accurately localize and track IoT devices in a space. The second subsystem is referred to as 3DRIMR/R2P, i.e., 3D Reconstruction and Imaging via mmWave Radar, and Radar to Point Cloud. The 3DRIMR/R2P is a deep neural network architecture (based on conditional GAN) that takes as input raw mmWave radar sensing signals scanned from multiple different viewpoints of an object, and then outputs the 3D shape of the object in the format of point cloud.

In the rest of this chapter, we will first provide the background for designing the UAV-based sensing, localization, and imaging system in Section 1.1. Then we will present the contributions of the dissertation research in Section 1.2. The organization of the dissertation will be discussed in Section 1.3.

1.1 Background

1.1.1 Background of LIDAUS

The first subsystem LIDAUS is an infrastructure-free IoT device localization system based only on RSSI, due to its ubiquitous presence and wide availability in the existing commodity IoT devices, especially Bluetooth devices.

In recent years, indoor localization of wireless devices or sensor nodes in general has been an active research area [BVH16, AQ20, VKK16, SRC12, SKW, LZP11, XJS14, YYR06, SHZ15, AVB18, KJB15, ABF, WXY12, WL98, KPS14, VW07]. For example, [BVH16] applied Received Signal Strength Indicator (RSSI) in localizing smart devices. However, directly utilizing RSSIs can result in large estimation errors due to shadow fading and multi-path fading in complex indoor environments [YZL13]. To address the problem, research has been done on improving RSSI-based techniques, e.g., via better fingerprinting [SHZ15, ABF]. Furthermore, Channel State Information (CSI) has been utilized to get more accurate localization, but usually these methods need to build customized equipment and/or require some coordination between wireless devices in a space [WXY12, SKW, LZP11, XJS14, YYR06, WL98, KPS14]. The recent trend of applying neural networks to indoor localization relies on a pre-built model based on extensive data collection and model training [AQ20][AAW], which is not applicable to a dynamic changing space. Furthermore as of today, CSI information cannot be easily obtained from Bluetooth signals (unlike RSSI) even though some recent research [AVB18] has made some effort in this direction. Since Bluetooth has been the dominating technology in IoT device market and its global market size is expected to reach $58.7 billion by 2025 [Gra17], a universal IoT device localization system should certainly include Bluetooth devices.

LIDAUS is applicable to various IoT devices with RSSIs. We focus on Bluetooth RSSIs. We now discuss the background of RSSI-based distance estimation and the challenges of applying it to wireless IoT device localization.

Distance estimation based on RSSIs. The functional relationship between the distance (between a signal transmitter and a receiver) and the RSSI measured by a receiver is given

by the following equation [SR92]:

$$RSSI = -10\alpha \log_{10} d/d_0 + \beta \tag{1.1}$$

where α is the signal propagation exponent, β is a reference *RSSI* value at d_0, and d is the distance between a signal transmitter and a receiver. We usually set d_0 to be $1m$ so that we can get the value of β, which is *RSSI* measured at a distance of $1m$ from the node. Then the above equation can be simplified as:

$$RSSI = -10\alpha \log_{10} d + \beta \tag{1.2}$$

The parameters α and β in Eqn. (1.2) are different for different wireless transmitters, and they can be estimated in practice. We can use Least Square Method (LSM) [EH05] to get the estimated values of α and β based on experimentally collected RSSIs and distance values. Specifically, we can place a signal receiver at various distances away from a transmitter and measure the *RSSI*s of received signals, denoted by $RSSI_m$. Let d_m denote a measured distance between the signal transmitter at position (x_t, y_t, z_t) and the receiver at (x_r, y_r, z_r). Based on Eqn. (1.2), we can get an expected value $RSSI_e$ as a function of a measured distance d_m:

$$RSSI_e = -10\alpha \log_{10} d_m + \beta \tag{1.3}$$

where $d_m = \sqrt{(x_r - x_t)^2 + (y_r - y_t)^2 + (z_r - z_t)^2}$. Then the estimated parameters $\hat{\alpha}$ and $\hat{\beta}$ can be obtained by using N measured pairs $(d_m, RSSI_m)$ in solving the following optimization problem:

$$\hat{\alpha}, \hat{\beta} = \arg\min_{\alpha,\beta} \sum_{i=1}^{N} (RSSI_m - (-10\alpha \log_{10} d_m + \beta))^2 \tag{1.4}$$

In this dissertation, we use the LSM to derive the estimated α and β.

1.1.2 Background of 3DRIMR/R2P

Our second subsystem 3DRIMP/R2P reconstructs the 3D shapes of objects in a space by processing raw Frequency Modulated Continuous Wave (FMCW) mmWave radar sensing data with deep learning models. We now present the background for this part of our research.

FMCW Millimeter Wave Radar Sensing and Imaging. Frequency Modulated Continuous Wave (FMCW) mmWave radar sensor works by periodically transmitting continuous chirps that each linearly sweeps through a certain frequency band [Texb]. Transmitted signals reflected back from an object in a 3D space will be received by a receiver. Range Fast Fourier Transform (FFT) is conducted on the received waveforms to detect the distance of the object from the radar sensor. In addition, multiple receivers can be arranged horizontally and vertically to form virtual antenna arrays along both dimensions. Then two additional FFTs can be applied to the data to calculate an object's relative angles from the sensor horizontally and vertically, referred to as azimuth angle ϕ and elevation angle θ. Those three FFTs together can generate a 3D heatmap or intensity map of the space that represents the energy or radar intensity per voxel, which is written as $x(\phi, \theta, \rho)$.

The process of electronically or mechanically steering an antenna array to get high azimuth and elevation angle resolutions is referred to as Synthetic Aperture Radar (SAR) operation. Usually higher resolutions along these two dimensions requires longer SAR operation time given fixed number of transmitters and receivers. However, high range resolution can be achieved even with commodity mmWave radar sensors without time consuming SAR process. In our work, we use IWR6843ISK [Texc] operating at 60 GHz frequency.

Different from LiDAR and camera sensors, the data generated by mmWave radar sensors is usually sparse, of low resolution, and highly noisy. Though SAR can help improve resolution, it is a very slow process, which may not be practical in many application scenarios that require short application response time. The specularity characteristic makes an object's surface behave like a mirror, so the reflected signals from a certain portion of the object will not be received by a receiver (hence missing data). In addition, the multi-path effect can cause ghost points which give incorrect information on the object's shape. For detailed discussions on FMCW mmWave radar sensing, please see [GMJ, FN, LRZ].

3D Reconstruction Representations. We use point cloud, a widely used format in robotics, to represent 3D objects' geometry (e.g., [LRZ]). This format has been used in recent work on learning-based 3D reconstruction, e.g., [FSG, QSM16, QYS17]. The point cloud representation of an object is a set of unordered points, with each point is a sample

point of the object's surface, written as its 3D Cartesian coordinates. Unlike voxelization of 3D objects, point cloud can represent an object with high resolution but without high memory cost. However, CNN convolutional operation cannot be applied to an unordered point cloud set.

In addition, though we can generate the point cloud of an object by directly filtering out mmWave radar 3D heatmaps, such a resulting point cloud usually has very low resolution, being sparse, and with incorrect ghost points due to multi-path effect. Therefore, although it may be acceptable to just use such a point cloud to detect the presence of an object, it is impossible to reconstruct the shape of the object. Our work attempts to solve this problem by using two generator neural networks to produce a smooth and dense point cloud based on raw radar data.

Other than point cloud, voxel representations are commonly used in 3D reconstruction to represent 3D objects in learning based approaches, for example, [KHM17, PUS, JGZ17, WZX16], and 3D CNN convolutional operations can be applied to such data models. However, such representations have cubic growth rate in the number of voxel grids, so they are limited to representing low resolution objects. In addition, mesh representations of 3D objects are considered in existing work, e.g., [KLL, WZL18], but they are also limited by memory cost and are prone to self-intersecting meshes.

1.2 Dissertation Contributions

In this dissertation, we aim to develop a design framework of UAV-based environment sensing, localization and imaging system, which can not only accurately localize objects in an 3D space, but also reconstruct objects' 3D shapes and hence reconstruct the target 3D scene. The whole system consists of two subsystems: sensing and localization of IoT devices via UAV SLAM and 3D reconstruction and imaging via mmWave radar and deep learning. In addition, we also explore the feasibility of two different models of reconstructing the 3D shapes of multiple objects. The contributions of this dissertation can be summarized as follows.

1. We first develop and evaluate our sensing and localization subsystem, LIDAUS (Localization of IoT Device via Anchor UAV SLAM), an infrastructure-free, multi-stage SLAM system that utilizes an Unmanned Aerial Vehicle (UAV) to accurately localize IoT devices in a 3D indoor space. LIDAUS consists of two stages: exploring stage, which derives an initial estimation of the positions of the IoT devices (referred to as target beacons) in a 3D space, and multi-round searching stages, which accurately localize all devices. It discretizes a 3D space into multiple horizontal layers, with each layer being modeled as a grid graph. The exploring stage adopts a UAV path planning based on an Eulerian cycle that covers all edges of the grid graph of each layer, in order to resolve direction or angle ambiguity that is inherent in RSSI-only distance estimation. The UAV's path in each searching stage is based on a Steiner tree that allows the UAV to approach each beacon's estimated position with minimum cost of deploying anchor beacons, which are dynamically deployed by the UAV along its path to mitigate the impact of the unreliable noisy RSSIs of target beacons in its SLAM computation. We also design *U-SLAM*, a SLAM algorithm based on FastSLAM [Mon03] that can do SLAM in a 3D search space. We introduce a weighted entropy-based clustering algorithm that selects a cluster of positions where the RSSIs observed for a specific target beacon can be used in the location estimation of the beacon in an offline U-SLAM replay to improve the beacon's localization accuracy. This contribution is reported in a publication [SXH].

2. We further introduce 3DRIMR/R2P (3D Reconstruction and Imaging via mmWave Radar/Radar to Point Cloud), for 3D object shape reconstruction and imaging via mmWave radar and deep learning. It applies a novel conditional GAN architecture that exploits 3D CNN for local structural information extraction and point cloud based neural networks for highly efficient 3D shape generation with detailed geometry. This subsystem is a fast 3D object reconstruction system that works on the signals of two radar snapshots of an object received by a commodity mmWave radar sensor, instead of a slow full-scale Synthetic Aperture Radar (SAR) scan. It can also works directly on sparse and noisy raw radar data without any structural assumption

or annotation. Our proposed point cloud based network architecture can generate smooth, dense, and highly accurate point cloud representation of a 3D object with fine geometry details, based on rough and sparse point clouds with incorrect points. These input point clouds are directly converted from the 2D depth images of an object that are generated from raw mmWave radar sensor data, and thus characterized by mutual inconsistency and orientation/shape errors, due to the imperfect process to generate them. We further demonstrate with extensive experiments the importance of loss function design in the training of models for reconstructing point clouds. We also show the limitations of Chamfer Distance (CD) and Earth Mover's Distance (EMD), the two state of art point clouds evaluation metrics, in the evaluation of the shape similarity of two point clouds. This contribution is reported in publications [SHZ21][SZH].

3. In the third part of this dissertation, we further explore the feasibility of utilizing a mmWave radar sensor installed on a UAV to reconstruct the 3D shapes of multiple objects in a space, rather than a single object. We propose and evaluate two different models. Model 1 is our recently proposed 3DRIMR/R2P model [SHZ21][SZH], and Model 2 is formed by adding a segmentation stage in the processing pipeline of Model 1. We demonstrate that it is feasible to utilize a multi-stage deep neural network model to reconstruct multiple objects in a 3D space based on mmWave radar data. In addition, we find that Model 1 has better quantitative results than Model 2 even though Model 2 has an extra segmentation stage and can result in denser/smoother point clouds. This is because any segmentation error can cause cascading failures in the following reconstruction stage, hence making the model less robust and more error-prone. Furthermore, we find that both models are fairly robust to the highly distorted and noisy radar data collected by unstable SAR operation due to the vibration/instability of a commodity UAV when hovering. This finding shows that the inherent intricate characteristics of radar energy signature of a space is still retained in the imperfect SAR data, and shows that it is feasible to use low-cost small

UAV in an environment sensing/reconstruction mission under low visibility. This contribution is reported in a publication [SXZ].

1.3 Dissertation Organization

This dissertation introduces the design framework of a UAV-based environment sensing and imaging system. The following chapters will present our research work in details.

Chapter 2: We first introduce some related work and challenges of indoor localization of IoT devices. Then we present our sensing and localization system design of LIDAUS and give detailed explanation of our techniques used. After that, we present our evaluation results via simulations and experiments. Finally, we draw a conclusion and discuss our future work.

Chapter 3: At the beginning, we introduce some related work and challenges to reconstruct an object's 3D shape in a low-visibility environment. Then we introduce the architecture and technical details of the design of 3DRIMR/R2P, our 3D shape reconstruction system. Finally, we present our evaluation results and findings via implementation and experiments.

Chapter 4: We first introduce the related work and challenges to further reconstruct multiple objects in a 3D space. Then we present design details of our two models to solve this problem. After that, we present our findings.

Chapter 5: In this chapter, we summarize this dissertation, and explain how we will extend our work in the future.

CHAPTER 2

LIDAUS: LOCALIZATION OF IOT DEVICE VIA ANCHOR UAV SLAM

In this chapter, we introduce the first subsystem of our design framework, LIDAUS (**L**ocalization of **I**oT **D**evice via **A**nchor **U**AV **S**LAM), an infrastructure-free, multi-stage SLAM[1] system that utilizes a UAV (Unmanned Aerial Vehicle) to search and accurately localize IoT devices in a space. LIDAUS is an infrastructure-free IoT device localization system based only on RSSI, due to its ubiquitous presence and wide availability in the existing commodity IoT devices, especially Bluetooth devices. We aim to design this subsystem that (1) can work in an indoor space where GPS is not available or a space that is not accessible for humans, (2) does not require any fingerprinting or pre-trained model and can deal with dynamic changing environment, (3) is easily deployable (without any customized signal processing hardware). IoT devices are also referred to as *beacons* in this dissertation.

2.1 Related Work

Indoor localization of wireless sensors, devices, or humans has been an active research topic in recent years [VKK16, SRC12, SKW, LZP11, XJS14, YYR06, SHZ15, AVB18, KJB15, ABF, WXY12, WL98, KPS14, VW07]. RSSI has been utilized in many range-based indoor localization system, but RSSI exhibits high temporal and spatial variance due to the multipath effect. To address this issue, research has been done on improving RSSI-based techniques, e.g., a gradient-based fingerprinting method in [SHZ15] and a neural-network based approach in [ABF]. In addition, CSI-based method has also been studied extensively, e.g., [WXY12].

[1] Simultaneous Localization and Mapping (SLAM) refers to a class of techniques used by a mobile robot to explore an environment with unknown landmarks, with a goal to develop a map of the environment (i.e., to localize the landmarks) and in the mean time to localize itself within the environment during its trip.

One of the most recent work on localizing IoT devices is iArk [AQ20]. But unlike our work, iArk needs to design a large customized antenna array with pre-trained neural network models, which makes iArk not applicable for a dynamically changing environment. Another related work, HumanSLAM [BVH16], also utilizes SLAM based on the RSSIs received by smartphones carried by human users walking around to locate IoT devices. Different from [BVH16], our system can be applied in an environment which is dangerous for human to access, e.g., a nuclear plant site or a disaster site.

Similar to our idea of using anchor beacons, [VW07] proposes a method to let sensor nodes to compute their own positions based on the received signals from a set of pre-deployed and position-known anchor nodes. But in our work, anchor nodes are dynamically deployed with only estimated positions in a UAV search process. Our work is also related to the rich literature in robotics on applying mobile robots in search and rescue. FastSLAM [Mon03] is adopted as an important component in our system. Application of UAVs in various mobile scenarios has also been active research area. For example, [PBS] introduces a method that lets a UAV to automatically collect CSI measurements for finger-printing based localization. [MBT17] discusses a vision-based UAV system for crowd surveillance.

2.2 Methodology

In this section, we present the design of LIDAUS. We first discuss the design challenges. Then we give an overview of the system's architecture. We further illustrate our key design ideas of the whole system. Then we discuss UAV's path planing and the various components of the system.

2.2.1 Challenges

There are two major design challenges of applying the RSSI-based distance estimation (Section 1.1.1) in the localization of a wireless IoT device.

First, the RSSIs of IoT devices are highly variable [YZL13], which is in sharp contrast to the reliable and precise sensor data (e.g., LIDAR data) used in the conventional SLAM

in robotics. Therefore, in a SLAM-based localization system (which we intend to design), directly feeding RSSI data to a robot's SLAM algorithm can result in large estimation errors of both the robot itself and the IoT device to be localized.

Second, RSSIs alone can only be used to estimate the distance between a robot or UAV from an IoT device, but not the direction or angle of the device relative to the robot. For example, if a device is on the north side of a UAV, the UAV flies on a straight line from west to east cannot decide whether the device is on its north side or south side. This ambiguity is referred to as *angle or direction ambiguity*.

2.2.2 Architecture Overview

LIDAUS is a *multi-stage SLAM system* that consists of a UAV and a set of anchor beacons. The UAV is equipped with a wireless signal receiver module and implements a software system with its architecture shown in Figure 2.1.

The architecture includes the following main modules.

1. **Exploring stage module.** This module generates the initial estimates of the positions of the IoT devices, referred to as *target beacons*, in the space to be explored. The UAV's path planning is done through an algorithm based on a Eulerian cycle. The estimated 3D coordinates of target beacons are derived via a weighted entropy-based clustering algorithm.

2. **Searching stage module.** The UAV may need to conduct multiple searching stages in order to finalize its localization of all target beacons. In each searching stage, the UAV flies along a path based on a Steiner tree that connects the estimated positions of those target beacons that have not been identified as *found* yet. The UAV utilizes a dynamic anchor deployment mechanism to help localize itself during its SLAM flight.

3. **Weighted entropy-based clustering algorithm**. This algorithm runs at the end of every stage in order to find a cluster or set of high quality location points at which the

observed RSSIs of a specific beacon should be used to estimate the position of the beacon.

4. **U-SLAM algorithm**. The UAV uses this 3D SLAM algorithm in its exploring and searching stages, and it also conducts a selective U-SLAM replay of the RSSIs found by the weighted entropy-based clustering algorithm, at the end of each stage.

5. **UAV flight control and sensing**. The module collects RSSIs from IoT devices and the sensor data from the UAV's onboard height sensors.

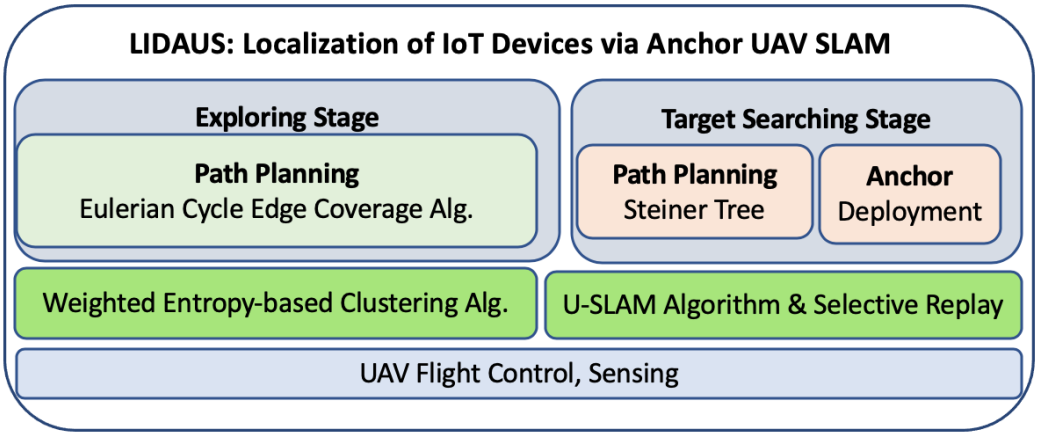


Figure 2.1: LIDAUS system architecture.

2.2.3 Some Key Design Ideas

Our LIDAUS subsystem introduces a few novel design ideas. We now illustrate two of them through simple examples. The first one is that after each SLAM trip of a UAV, for a particular beacon, the UAV conducts a *SLAM replay* based only on a selected set of observation location points that gives high quality RSSIs for the beacon (discussed in Section 2.2.7). The second idea is to utilize *anchor beacons* in a SLAM process. We let the UAV deploy anchor beacons[2] on its path to help improve the accuracy of localizing itself during its trip, described later in Algorithm 2.2.4 in Section 2.2.9.

[2]Deploying or releasing a low-cost coin-sized Bluetooth beacon can be done through a simple mechanism attached to the UAV. In addition, a low-cost NRF51822 beacon only costs around $4 [ALI].

2.2.3.1 Replay SLAM Based On High Quality RSSIs

We now use a simple 2D example to illustrate this idea as shown in Figure 2.2. There are 11 target beacons: $B\_0, B\_1, ..., B\_10$ placed on the floor of a room. Black dots show the actual locations of those target beacons. Red dots show the estimated positions of the targets when the drone returns its starting position. Green dots show the estimated positions of the targets by the drone in its SLAM process. Blue color dotted lines show the estimated locations of the UAV itself during its trip. The UAV first does a FastSLAM [Mon03] in a pre-determined

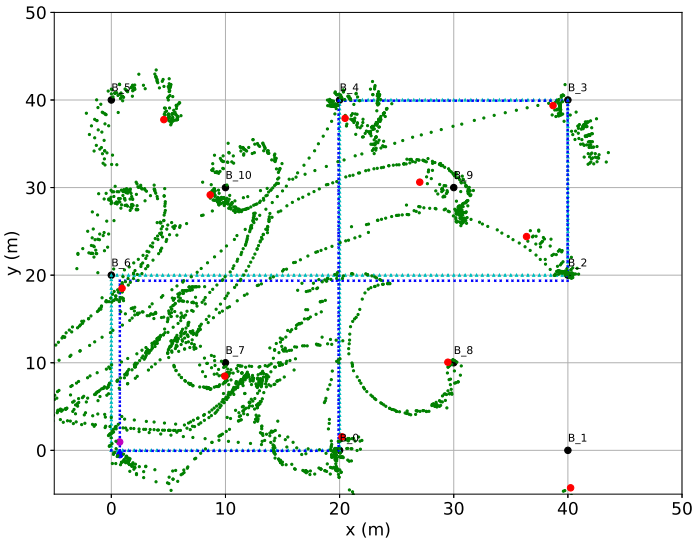


Figure 2.2: A 2D example to illustrate our key design idea.

path: $(0,0) \rightarrow (20,0) \rightarrow (20,40) \rightarrow (40,40) \rightarrow (40,20) \rightarrow (0,20) \rightarrow (0,0)$ in the room. This initial trip gives us an estimation of the locations of those target beacons. The blue lines in Figure 2.2 shows the UAV's path. The red dots show the final estimated positions of the beacons, and the green dots show the changing estimates during the trip. We see that many beacons cannot be accurately localized as indicated by the distance between the black dots and their corresponding red dots in Figure 2.2. To address this problem, for each beacon, we choose a set of RSSIs that are higher than or equal to a threshold RSSI that corresponds to 10 meters. We replay FastSLAM for each beacon based on its set of selected RSSIs, then get a further estimation of its location. Out of 11 beacons, 8 beacons' estimations

are significantly improved, with an average 57.7% improvement. Among the improved beacons, a set of RSSI observation locations that enclose a beacon usually gives the most accurate estimate. In our system design, we introduce a weighted entropy-based clustering algorithm to select a set of RSSI observation points for each beacon, shown in Section 2.2.7. This SLAM replay of beacon-specific high quality RSSIs, together with a Eulerian cycle based path planning (in Section 2.2.8), collectively address the aforementioned design challenges.

2.2.3.2 Anchor beacons improve SLAM performance

A typical SLAM algorithm of a robot highly depends on the accuracy levels of its sensor readings (that measure the robot's distance/orientation from surrounding landmarks) in order to get accurate estimations of its own location and the landmarks' locations. Unfortunately, the sensor readings used by our system, i.e., RSSIs of IoT devices, are highly noisy and unreliable. To address this problem, we let the UAV in our system to deploy additional beacons to aid its SLAM, which are referred to as *anchor beacons*. The IoT devices to be localized are referred to as *target beacons*. We now use a simple example to illustrate the benefits of anchor beacons. The deployment of anchor beacons is described in Algorithm 2.2.4 in Section 2.2.9.

Consider an $39m$ by $39m$ area that is divided into a grid of 39 by 39 cells with 40 by 40 nodes, as shown in Figure 2.3. Those 40×40 nodes form a grid graph. The distance between any two neighboring nodes is $1m$. We place 5 target beacons at 5 randomly chosen grid nodes. As shown in Figure 2.3, red dots represent IoT beacons, and the yellow lines show the UAV's path. Blank blocks represent obstacles, e.g., rooms or furniture, in the area. The UAV uses FastSLAM while flying along the planned route.

We simulate the environment with three different noise levels, i.e., standard deviations (std), of the RSSIs generated by the target beacons: small ($std = 3$), medium ($std = 5$) and large ($std = 8$). For each noise level, we do the simulation to estimate the 5 target beacons under the following 3 cases: (1) Place one anchor beacon every one meter; (2) Place one

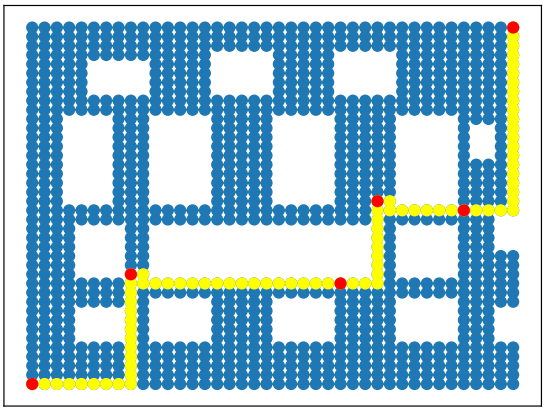


Figure 2.3: The locations of the five target beacons, and the UAV's path.

anchor beacon every four meters; (3) No anchor beacon used. For each experiment, we repeat it 10 times using different random seeds.

The simulation results show that deploying anchor beacons always gives much smaller estimation errors than the cases without using any anchor beacon. This improvement can be very significant when RSSIs are quite noisy or even when a target beacon is very far away from the UAV's starting point. For example, Figure 2.4 shows the results of the beacon at $(28, 20)m$. In the figure, L means we use anchors in the experiment, while N means we do not use them. Numbers 3, 5, and 8 represent the three RSSI noise levels. The $1m$ and $4m$ respectively corresponds to one anchor per meter and one anchor per 4 meters on the path. We can see that compared with the case that no anchor is used, deploying one anchor per meter can improve the average estimation accuracy by 133%, 381% and 229% for RSSI noise levels being 3, 5 and 8 respectively. In each RSSI noise level, we can get smaller estimation error with higher anchor density. When noise level is as large as 8, the estimation error in the case of deploying one anchor per meter is only 85% of that when deploying one anchor every 4 meters and 22% of that without using anchor beacons.

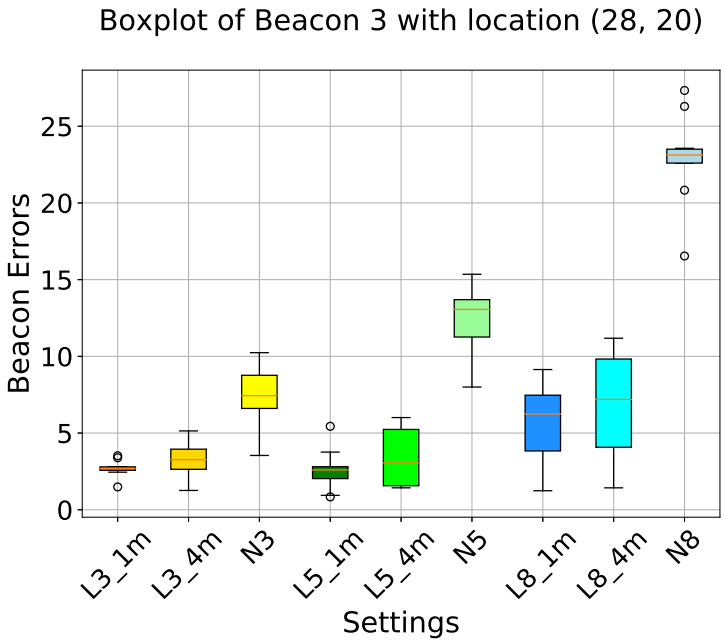


Figure 2.4: Boxplots of estimation errors of a target beacon.

2.2.4 Overview of Path Planning

We now give an overview of the UAV's path planning. The basic idea is to divide an indoor 3D space into multiple horizontal layers, on which the UAV conducts SLAM trips and collects RSSI data. The UAV's facing direction is always aligned with the x-axis or y-axis directions of the space, in order to minimize the movement errors caused by rotation. As for the z-axis direction, the quadcopter UAV used in our system always go straight up or down. The overall workflow of LIDAUS is presented in Algorithm 2.2.1. The UAV's trip includes an *exploring stage* and one or multiple *searching stages*.

In the exploring stage, the UAV flies on all the layers, one by one from a low layer to a high layer. On each layer, its path is determined by Algorithm 2.2.3 (in Section 2.2.8). Figure 2.5 gives an example where the space is discretized into 4 layers. On each layer, its path covers all edges of a grid graph representation of the layer and is determined by a Eulerian cycle. The UAV starts from the origin $(0,0,0)$ and returns to the origin at the end of its trip. Its path consists of 8 segments, among which segments $1,3,5$ and 7 are on the four horizontal layers. Since a typical indoor space has a ceiling height of ≤ 3 meters, thus

Algorithm 2.2.1: System Workflow

Input: A 3D space where IoT beacons to be localized.

Output: Estimated locations of all IoT beacons in the space.

1: *Exploring stage:* Find initial estimations of all IoT target beacons' locations. The UAV invokes Algorithm 2.2.3 for path planning. It conducts U-SLAM (Section 2.2.5) along the path and collects RSSIs. At the end of the stage, it invokes weighted entropy-based clustering algorithm (in Algorithm 2.2.2) to find a cluster of observation locations for each target beacon, and performs a selective replay of U-SLAM on those clusters to derive estimated coordinates of all target beacons.

2: *Searching stage(s):* Multiple repeated stages may be needed in order to finalize the localizations of all target beacons. In each stage, the UAV invokes Algorithm 2.2.4 for a Steiner-tree-based path planning, and then it performs U-SLAM along the path. A target beacon is labeled as *found* if the UAV receives a certain number of observed high RSSIs (above a threshold). At the end of the stage, it performs clustering and U-SLAM replay to update the estimated positions of the *found* target beacons. Conduct a new search stage if there are still not-found target beacons.

this layered discretization along the z-axis will not significantly increase the complexity of our system.

In a searching stage, the estimated positions of the target beacons whose final locations have not been decided yet are projected on the ground layer, and a Steiner tree is built to connect all the projected positions. Since UAV deploys anchor beacons to help its SLAM process and they can only be deployed on the ground, thus the tree is constructed on the graph formed by the nodes projected to the ground layer. The UAV keeps a record of the estimated height of each beacon, and when it reaches a target beacon's projected position on the ground layer, it flies up to reach the estimated height of the beacon. The algorithm to find a path in this stage is given in Algorithm 2.2.4 in Section 2.2.9. Figure 2.6 shows an example.

In the next three subsections, we will discuss U-SLAM algorithm, exploring stage, and searching stage.

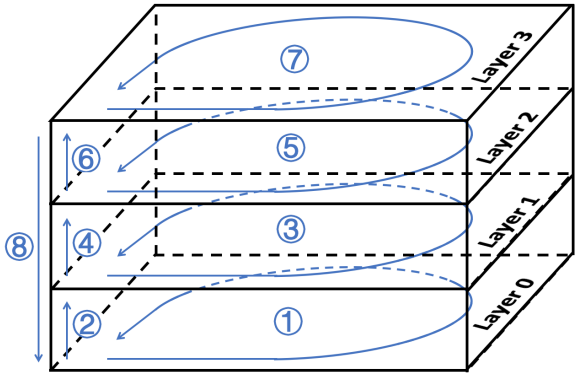


Figure 2.5: Illustration of the path planning in the exploring stage.

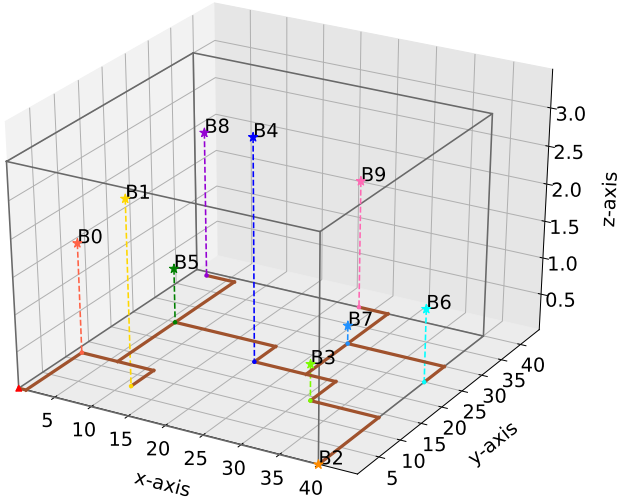


Figure 2.6: An example of the path planning in a searching stage.

2.2.5 UAV SLAM (U-SLAM)

A key component of this subsystem is U-SLAM. In the context of the problem we study, a UAV is a robot, and the wireless IoT devices to be localized are landmarks. U-SLAM extends FastSLAM 1.0 [Mon03] algorithm[3] by (1) adding a Kalman Filter (KF) to estimate the elevation of the UAV through a barometer and a laser sensor (ToF sensor VL53L1x)[4],

[3]We have found that FastSLAM 2.0's performance is not as good as FastSLAM 1.0 in the problem studied in this paper, as FastSLAM 2.0 relies more on the sensing data of landmarks (i.e., IoT beacons' RSSIs) which are highly noisy and unreliable.

[4]This can be easily extended by adding more accurate and longer range sensors such as mmWave sensors [Tex20].

(2) expanding the Extended Kalman Filters (EKF) of FastSLAM to estimate the three-dimensional coordinates of the target IoT devices, and (3) letting the UAV dynamically deploy *anchor beacons* on its path and use the RSSIs of the anchor beacons in updating the posterior of its own position.

Recall the IoT devices that are to be localized are called *target beacons*. Let $\mathbb{T}$ denote the set of target beacons. Let $\mathbb{A}$ denote the set of anchor beacons, and let $\mathbb{B} = \mathbb{T} \bigcup \mathbb{A}$. Let b_i denote the position of beacon i with $b_i = (b_{i,x}, b_{i,y}, b_{i,z})$, and vector $\mathbf{b}$ denote the positions of all beacons. The UAV's position is denoted by $s = (s_x, s_y, s_z)$. A beacon's relative position with respect to the UAV is given by $g = (r, \varphi, \gamma)^T$, i.e., the Euclidean distance, and the azimuth and elevation angle differences, between the UAV and the beacon, as shown in Figure 2.7.

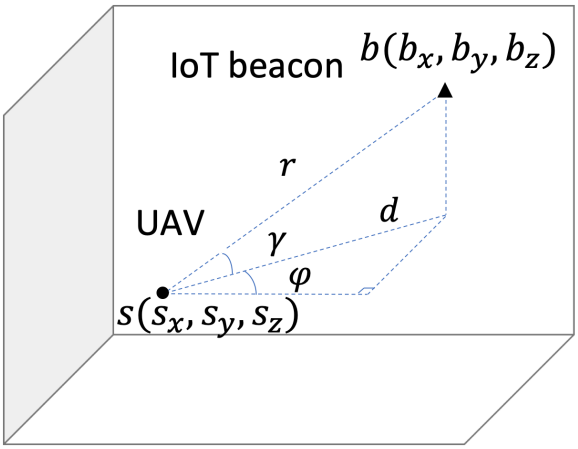


Figure 2.7: An IoT beacon's relative position with respect to the UAV.

Observed and estimated parameters. The observed distance r_i of a beacon i from the UAV is calculated by using Eqn. (1.2) and the observed RSSIs of beacon i. The observed height of the UAV, denoted by h, is from the height sensors of the UAV. The UAV uses a KF to estimate s_z based on h. Based on observed r_i and estimated s_z and the motion command of the UAV (denoted by u), the UAV uses a particle filter to estimate its s_x, s_y. Inside each particle, the UAV uses an EKF to estimate beacon i's $b_i = (b_{i,x}, b_{i,y}, b_{i,z})$. Note that the RSSIs of a beacon do not give us observed γ and φ. However, we can easily derive γ after we derive the estimated b_i and s_i. In addition, since we keep the UAV's facing direction

always along the x-axis and the y-axis of the space, the azimuth angle difference φ can also be derived from estimated b_i and s.

Let $r_{i,t}$ denote the observed distance between beacon i and the UAV at time t. Vector $\mathbf{r}_t = (r_{1,t}, r_{2,t}, ..., r_{i,t}, ..., r_{|\mathbb{B}|,t})$ includes all beacons at time t. The UAV's location at time t is denoted by s_t, and an IoT beacon i's position at time t is denoted by $b_{i,t}$. Let u_t denote the motion command given by the UAV's controller to the UAV at time t. The UAV's history of positions from time 0 to time t is denoted by $s_{0:t}$. Similarly, we have $\mathbf{r}_{0:t}$, $h_{0:t}$, and $u_{0:t}$.

2.2.5.1 Calculate the posteriors of the positions of UAV and beacons

The UAV updates its estimation of the following probability during its flight[5].

$$p(s_{0:t}, \mathbf{b}) | \mathbf{r}_{0:t}, h_{0:t}, u_{0:t}) \tag{2.1}$$

At the end of the UAV's flight, the final estimate of the mean of $\mathbf{b}$ tells us the estimated locations of all beacons.

The UAV solves (2.1) via the conditional independence between beacons based on the knowledge of the UAV's path. That is,

$$\begin{aligned} & p(s_{0:t}, \mathbf{b} | \mathbf{r}_{0:t}, h_{0:t}, u_{0:t}) \\ & = p(s_{0:t} | \mathbf{r}_{0:t}, h_{0:t}, u_{0:t}) p(\mathbf{b} | s_{0:t}, \mathbf{r}_{0:t}, h_{0:t}, u_{0:t}) \\ & = p(s_{0:t} | \mathbf{r}_{0:t}, h_{0:t}, u_{0:t}) \prod_{i=1}^{|\mathbb{B}|} p(b_i | s_{0:t}, r_{i,0:t}, h_{0:t}, u_{0:t}) \end{aligned} \tag{2.2}$$

The algorithm estimates the UAV's s_x and s_y by using a particle filter that has L particles, and it estimates s_z by using a KF with observed height h from height sensors. In addition, the algorithm estimates beacon i's posterior $p(b_i | s_{0:t}, r_{i,0:t}, h_{0:t}, u_{0:t})$ by using EKFs, with one EKF for each beacon.

[5]We do not need to solve the data association problem as in traditional robotics SLAM algorithms, as each beacon's signal contains its unique ID.

Consider a beacon i. The algorithm uses the following formula to compute the posterior of its position b_i:

$$p(b_i|s_{0:t}, r_{i,0:t}, h_{0:t}, u_{0:t})$$
$$= \eta \cdot p(r_{i,t}|b_i, s_t) \cdot p(b_i|s_{0:t-1}, r_{i,0:t-1}, h_{0:t}, u_{0:t-1}) \tag{2.3}$$

where η is a normalizing factor.

The algorithm maintains a set of particles $\mathbf{Y}_t$ and a KF at time t. Each particle ℓ contains its estimated x- and y-coordinates of the UAV at time t (i.e., $s_x^{[\ell]}, s_y^{[\ell]}$). A KF contains estimated UAV's height s_z. Let $s_t^{[\ell]}$ denotes the vector collection of these three estimates. In addition, particle ℓ contains the estimated mean and covariance of the position of each beacon i at time t, denoted by $(\mu_{i,t}^{[\ell]}, \Sigma_{i,t}^{[\ell]})$. For ease of exposition, we drop notations i, t, ℓ and consider a beacon's position distribution with mean and covariance (μ, Σ), then we have $\mu = (\mu_x, \mu_y, \mu_z)$ and

$$\Sigma = \begin{pmatrix} \sigma_x^2 & \sigma_{x,y} & \sigma_{x,z} \\ \sigma_{x,y} & \sigma_y^2 & \sigma_{y,z} \\ \sigma_{x,z} & \sigma_{y,z} & \sigma_z^2 \end{pmatrix}$$

Let $\mathbf{Y}_t^{[\ell]}$ denote particle ℓ at time t, then $\mathbf{Y}_t^{[\ell]} = \{s_t^{[\ell]}, (\mu_{1,t}^{[\ell]}, \Sigma_{1,t}^{[\ell]}), ..., (\mu_{|\mathbb{B}|,t}^{[\ell]}, \Sigma_{|\mathbb{B}|,t}^{[\ell]})\}$, and let $\mathbf{Y}_t = \{\mathbf{Y}_t^{[1]}, ..., \mathbf{Y}_t^{[L]}\}$.

2.2.5.2 The iterative steps

The algorithm generates $\mathbf{Y}_t$ from $\mathbf{Y}_{t-1}$ at time t based on the latest motion control u_t and observed values $\mathbf{r}_t$ and h_t by running the following steps:

1. **Step 1**. The height KF estimates the UAV's $s_{z,t}$ at t.
2. **Step 2**. Each particle ℓ predicts the new position $s_t^{[\ell]}$ based on u_t, $\mathbf{s}_{t-1}^{[\ell]}$, and $s_{z,t}$.
3. **Step 3**. Update the EKF's estimation of each beacon in every particle.
4. **Step 4**. Calculate the importance weight of each particle.
5. **Step 5**. Conduct an importance re-sampling to derive a new particle set.

2.2.5.3 Impacts of RSSI noise and UAV's motion noise

There are two types of measurements that can impact the estimation accuracy of the beacons' positions and the UAV's position over time. They are the beacons' RSSI noise and the UAV's motion noise. We next examine all 5 steps carefully.

Step 1's height estimation is affected by the height sensors' noises and the UAV's motion command noise.

Step 2 of the algorithm utilizes a probabilistic motion model to predict the UAV's position, i.e., $s_t^{[\ell]} \sim p(s_t|u_t, s_{t-1}^{[\ell]})$, so this prediction is affected by the motion control noise, not the noise of the measured RSSIs of beacons.

Step 3 updates the position estimation of a beacon i by using an EKF. Consider particle ℓ, its EKF first calculates a predicted distance $\hat{r}_i$ from beacon i with the beacons's position estimated at the previous time step $t-1$ and the predicted new position of the UAV, i.e.,

$$\hat{r}_{i,t}^{[\ell]} = g(s_t^{[\ell]}, \mu_{i,t-1}^{[\ell]}) \tag{2.4}$$

where $g(s, b_i)$ is the Euclidean distance function between s and b_i.

Then the EKF derives the estimated mean and covariance of beacon i at time t according to the following equations:

$$\mu_{i,t}^{[\ell]} = \mu_{i,t-1}^{[\ell]} + K_t^{[\ell]}(r_{i,t} - \hat{r}_{i,t-1}^{[\ell]}) \tag{2.5}$$

$$\Sigma_{i,t}^{[\ell]} = (I - K_{i,t}^{[\ell]} G_{i,t}^{[\ell]})\Sigma_{i,t-1}^{[\ell]} \tag{2.6}$$

where coefficients $K_{i,t}^{[\ell]}$ and $G_{i,t}^{[\ell]}$ both are functions of $s_t^{[\ell]}, \mu_{i,t-1}^{[\ell]}, \Sigma_{i,t-1}^{[\ell]}$. Therefore, we can write the following functions to calculate the estimated mean and covariance of beacon i in particle ℓ at time t:

$$\mu_{i,t}^{[\ell]} = f(r_{i,t}, s_t^{[\ell]}, \mu_{i,t-1}^{[\ell]}, \Sigma_{i,t-1}^{[\ell]}) \tag{2.7}$$

$$\Sigma_{i,t}^{[\ell]} = f(s_t^{[\ell]}, \mu_{i,t-1}^{[\ell]}, \Sigma_{i,t-1}^{[\ell]}) \tag{2.8}$$

Step 4 calculates the importance weight of each particle based on the measured RSSIs of the beacons at time t. Specifically, the weight of particle ℓ is a function of $r_{i,t}$.

$$w_t^{\ell} = f(r_{i,t}, s_t^{[\ell]}, \mu_{i,t-1}^{[\ell]}, \Sigma_{i,t-1}^{[\ell]}) \tag{2.9}$$

We follow the standard procedure to design the KF, the EKFs, and the importance sampling for particle filters.

Remarks. Equations (2.7), (2.8), and (2.9) show that the accuracy of estimated positions of the UAV and of beacons are highly dependent on the accuracy of the measured RSSIs. Thus, we let the UAV deploy anchor beacons on its path and only use anchor beacons' RSSIs in the estimation of its own locations in its U-SLAM, which helps improve its U-SLAM's localization accuracy.

2.2.6 SLAM Selective Replay

At the end of each stage (either the exploring stage or each searching stage), for a beacon i, the UAV selects a set of RSSI points according to a weighted entropy-based clustering method (i.e., Algorithm 2.2.2 in Section 2.2.7), denoted by C_i. Then the UAV conducts an offline replay of U-SLAM based on C_i to update its estimation of beacon i's position.

2.2.7 Weighted Entropy-based Clustering Algorithm

We design this algorithm to select a set or cluster of position points (denoted by C_i) for a beacon i, at which the observed RSSIs of beacon i can be used for a SLAM selective replay to get an accurate estimate of beacon i's position. This algorithm is shown in Algorithm 2.2.2. In addition, the center of cluster C_i is also used as the initial estimate in the SLAM selective replay for beacon i at the end of the exploring stage.

Let $P_{init} = \{p_1, ..., p_N\}$ denote the set of the observation points or positions where RSSIs are collected in a trip of the UAV. For an observation point p_j, the UAV derives a set of the medians of observed RSSIs, denoted by $R^j = \{R^j_1, ..., R^j_M\}$ for all M IoT beacons, denoted by set $T = \{T_1, ..., T_M\}$.

For a beacon i, we use two thresholds to control the selection of RSSI observation points to be added into cluster C_i: RSSI difference level R_{dL} and RSSI cluster threshold $R_{th,c}$. These two thresholds are dynamically adjusted during the execution of the clustering

Input	Explanation
$P_{ini} = \{p_1, ..., p_N\}$	Initial path
$R^j = \{R^j_1, ..., R^j_M\}$	M IoT target beacons $T = \{T_1, ..., T_M\}$
$R_{dL} = 0$	Initial RSSI difference level
$R_{th,c}$	Initial RSSI cluster threshold
S_{th}	Entropy threshold
$\{G_1, ..., G_q\}$	Edge weight look-up table of each 3D cube
L_G	The total number of edges of the multi-layer grid graph representation of the search space
$L_{i,k}$	The number of observation points in an edge e_k that is selected for beacon i

Table 2.1: Input of Algorithm 2.2.2 - Weighted Entropy-based Clustering Algorithm.

algorithm in order to get a sufficient number of points in C_i and in the mean time to ensure only location points with high quality RSSIs for the beacon are added into C_i.

Recall that based on the relationship between RSSI and distance, in general the closer a location is to a beacon, the stronger its RSSI can be observed. Thus we decide that the points where the RSSIs observed for a beacon is no less than $R_{th,c}$ can be candidate points for the beacon. However, due to multi-path and other factors, a strong RSSI of a beacon at a location that is not so close to the beacon might also be observed. To mitigate this impact, we assign a location point as a high quality observation location only to the beacon that has the strongest RSSI among all beacons with observed RSSIs at the location, i.e., the location is only used for the beacon with strongest RSSI, not used for other beacons' estimation even though their RSSIs are also observed. In case we cannot collect sufficient observations for a beacon, we lessen this strongest assignment requirement by assigning a location to a beacon if its RSSI at the location is within a difference threshold R_{dL} from the strongest RSSI at the location. This is shown on line 12 of Algorithm 2.2.2.

In addition, we use Shannon entropy to evaluate the quality of the selected observation points for a beacon. Recall all observation points are along the edges of a grid graph. For a beacon i, recall C_i denotes the set of the selected points for this beacon. Those points in C_i are distributed on various edges. We use the following two methods to evaluate the quality of those points in terms of estimating beacon i's position. **First**, note that the more evenly distributed the points on those edges, the better location estimation we can get for this beacon. For example, consider Case 5 in Figure 2.9. If all observation points are evenly distributed on all four edges instead of on only one edge, our SLAM algorithm can get high estimation accuracy. We use an entropy metric to measure the level of evenness or balance of the points assigned to beacon i. **Second**, for a particular beacon, we associate a weight w_k to each edge e_k; the w_k of an edge gets smaller if the edge is further away from the beacon. Specifically, we propose to use the following weighted entropy of beacon i's RSSI observation points to evaluate the quality of cluster C_i:

$$S_i = \Sigma_k(-w_k \frac{L_{i,k}}{L_i} \log(\frac{L_{i,k}}{L_i}))/\log(L_G) \tag{2.10}$$

where $L_{i,k}$ denotes the number of observation points in an edge e_k that is selected for beacon i, $L_i = \Sigma_k L_{i,k}$, and L_G denotes the total number of edges of the multi-layer grid graph representation of the search space. The goal of Algorithm 2.2.2 is to find a C_i for a beacon i so that the C_i's weighted entropy S_i is above a quality threshold S_{th}. Note that since the edges are weighted, we need to multiply the above S_i with another scaling factor to scale an entropy in the range of $[0,1]$. Note that a simple clustering to find location estimates of iBeacons is used in [LVN18], but their method is simply a direct application of K-means clustering.

Figure 2.8 shows a toy example to illustrate the calculation of the weighted entropy of the cluster of positions (shown as the red dots in the figure) for a beacon (shown as the black star in the figure). There are 48 observation points in this beacon's cluster. There are 30 edges in this space. Given a target estimated located in the middle cube of the graph, 12 cyan edges are the nearest (or core) edges to the beacon, with weight w_C; 12 blue edges are adjacent to the core edges, and their weights are w_A; 6 yellow edges belong to the Other

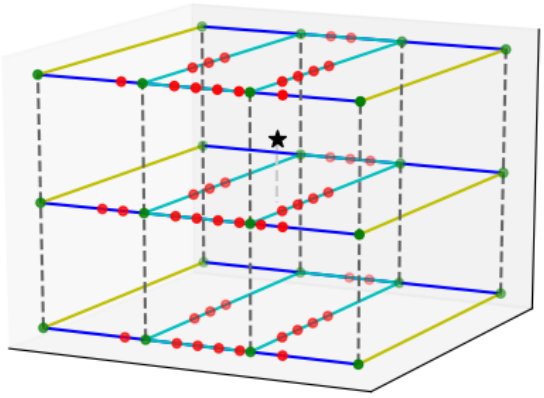


Figure 2.8: A toy example to calculate weighted entropy of the cluster of points of a beacon.

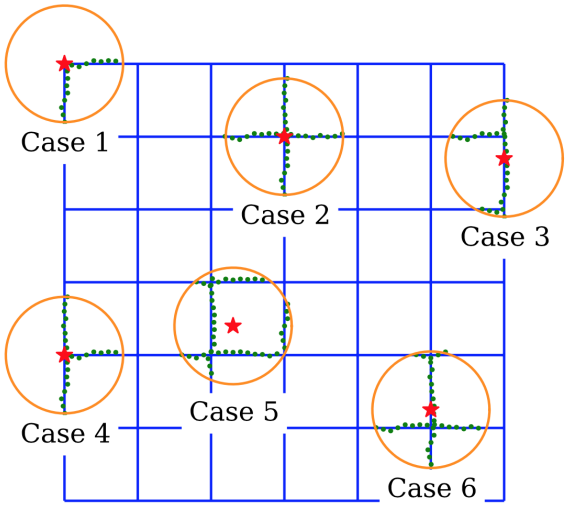


Figure 2.9: Example locations when a beacon is on a 2D grid graph.

class with weight w_O. The 12 core edges have 40 observation locations in total, the 12 adjacent edges have 8 observations, and there are 0 observation locations in other edges.

2.2.8 Exploring Stage Path Planning

In this stage, the UAV models each horizontal layer as a grid graph. Then it travels along a path based on an Eulerian cycle to cover all edges of the grid graph on each horizontal layer. It visits all layers, one by one from lower layers to higher layers, as shown in Figure 2.5. It runs U-SLAM during its trip and in the mean time it collects RSSIs of all IoT devices while

flying. After returning back, the UAV uses a *weighted entropy-based clustering algorithm* to derive initial estimated positions of the IoT devices in the space.

Path planning based on Eulerian cycle. On each layer, the UAV follows a path determined by Algorithm 2.2.3, which is a minimum distance path that *covers all the edges* of the squares of the layer. Let G_{in} denote the grid graph on a layer. Assume that each edge of the graph has a unit cost. The basic idea of Algorithm 2.2.3 is to first convert G_{in} to an even-degree graph (i.e., every node has even degree), and then utilize an Eulerian cycle algorithm to find the least cost (i.e., shortest distance) tour that covers all edges of the graph. The tour starts from the origin of G_{in} and returns back to the origin at the end of the tour.

Rationale for collecting RSSIs on all edges of the grid graph representation of a horizontal layer. This design addresses the problem of direction ambiguity of a beacon due to the fact that RSSIs can only indicate distance, but no direction or angle information. For ease of exposition, we take a look at a 2D plane shown Figure 2.9, where the red dots show the possible locations of an IoT beacon and the points inside a circle and along edges are the locations that can be used by SLAM replay. We see that in all cases, an edge covering tour can collect RSSI data of a beacon along both x-axis and y-axis directions. In addition, for Case 5 (a very common case), all the positions where we get the beacon's RSSI data enclose the beacon. This design can greatly improve the localization accuracy of U-SLAM.

2.2.9 Searching Stage

Following the exploring stage, the UAV goes through a limited number of target searching stages: $Stage_1, Stage_2, ..., Stage_K$. The limit K is a user tunable parameter, depending on the specific application scenario. This stage is described in Algorithm 2.2.4.

In each stage, the UAV constructs a Steiner tree [HR92] to reach all remaining not-yet-found target devices, based on the latest estimated positions of them from previous stages. It travels along the shortest branch of the tree to fly close to the targets on that branch. While flying along the tree nodes, the UAV also deploys anchor beacons to help its U-SLAM computation. During this stage, the UAV updates its estimation of all targets

through U-SLAM. The rationale of utilizing anchor beacons is to improve the estimation accuracy of U-SLAM, as the algorithm (and in fact all SLAM algorithms based on our knowledge) is highly dependent on the measured signals of beacons, i.e., RSSIs, but RSSIs are not reliable.

In addition, since a Steiner tree is a minimum-weight tree that spans all target device nodes in the graph of the area, the Steiner tree path can minimize the the deployment cost of anchor beacons. Note that we let the UAV fly close to a target, as RSSIs are more reliable when a receiver is very close to a transmitter. A target is labeled as *found* when the UAV observes a sufficient number of very strong RSSIs.

After exploring the shortest branch of the Steiner tree in *Stage i*, for all targets that are labeled as *found*, the UAV runs a U-SLAM selective replay based on the set of RSSI observation locations selected by the entropy clustering algorithm. If there are still targets that are not labeled as found yet, the UAV starts a new stage again. The total number of stages is bounded by an empirically determined number[6].

2.3 Implementation and Experiments

We simulate the operation of LIDAUS in a 3D space with dimensions $40m \times 40m \times 3m$. We discretize it into 4 horizontal layers, and their heights are of $0m$, $1m$, $2m$ and $3m$ respectively. We model each layer as a grid graph with squares of size $10m \times 10m$. There are 10 IoT devices (i.e., target beacons) scattered in the space as shown in Figure 2.6, and their estimated positions are projected on the ground layer where a Steiner tree is built to connect those projected nodes. The UAV has a Gaussian motion noise with standard deviation (std) of $0.05m$ in horizontal direction and $std = 0.03m$ in vertical direction, and the RSSI data received from a target beacon is generated through Eqn. (1.2) with a Gaussian noise $std = 0.1$. In simulations, the UAV always starts at the origin $(0,0,0)$, and it collects RSSIs at every $1m$ step.

[6]Note that in our simulations and experiments, the UAV can always quickly label all targets as found within 5 stages.

We compare LIDAUS with two baseline methods: *Random SLAM Fly* and *Naive SLAM Search*. With these two baseline methods, the UAV also starts at the origin and measures RSSIs every $1m$, and it also utilizes the 3D U-SLAM algorithm. But these two baseline methods do not deploy any anchor beacon. For all these three methods, the RSSI threshold to label a target beacon as *found* is -51, which corresponds to a distance of $1.7m$ (i.e., about the length of the diagonal line of a cube with $1m$ side length[7]). The UAV in LIDAUS follows the design given in Section 2.2.

With *Random SLAM Fly*, the UAV randomly chooses a direction to move at each step of $1m$. The UAV will stop if all targets have been labeled as found, or it will stop if its total flying step reaches $10k$ steps. Note that this Random SLAM Fly is similar to HumanSLAM [BVH16], but the former works in a 3D space and the motion control estimation is more accurate than the smartphone-based human motion estimation in HumanSLAM.

With *Naive SLAM Search*, the UAV also conducts a multi-stage U-SLAM flight. In its exploring stage, it flies layer by layer in the same discretized space as LIDAUS, but it follows the 8-like shape path (same as the one shown in Figure 2.2) on each layer in order to resolve direction or angle ambiguity. After that, the UAV sets the estimated position of the closest *unfound* target as its destination and flies toward it. During this trip, it keeps updating the estimates of all *unfound* target beacons via U-SLAM as new RSSIs are being observed constantly, and when the UAV finds the received RSSI from a target is greater than -51 at least three times, it will consider that target as *found*. When UAV reaches the destination, it will find the closest estimated position of an *unfound* target as the next destination and repeat the above process until all target beacons are labeled as found, or the total flying steps reach 10 thousand steps. To avoid keeping searching the same target all the time or getting into an infinite loop, we set a counter for each target. Whenever a target's estimated position is chosen as a destination, its counter increments by one. If a target's counter reaches 5, it will not be chosen as a destination in future.

[7]The relationship between RSSI and distance varies across different types of devices. A future work of ours is to design a learning module to deal with a mixture of various different types of wireless signals.

2.3.1 Comparison between Methods

We now compare the performance of these three methods. Figure 2.10 shows the localization errors (i.e., the distance between the true location of a beacon and its estimated location) of all three methods over all those 10 target beacons. We see that LIDAUS in general can achieve significantly higher localization accuracy than the other two methods. LIDAUS's estimation errors are always no more than $1m$ (except that one beacon's error is slightly higher than $1m$).

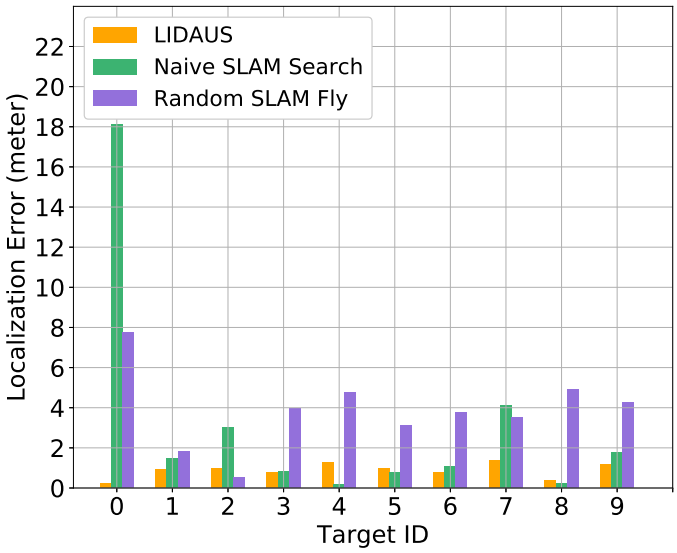


Figure 2.10: Bar charts of all targets' localization errors of the three methods.

Figure 2.11 shows the boxplots of localization errors of the three methods. We observe that the average localization errors of LIDAUS, Naive SLAM Search, and Random SLAM Fly are 0.88, 3.15 and 3.84 meters respectively. LIDAUS's average performance is significantly better than the other two methods. Figure 2.12 illustrates LIDAUS's localization errors along x, y and z directions. We can see that, except that target beacon 4 has a slightly large error along y-direction (1.05 meters), all other targets' localization errors along these three directions are limited within 1 meter. As a comparison, neither the other two methods can achieve this level of accuracy. Overall, we see that LIDAUS has the lowest average localization error and the lowest localization variance, and hence has the best performance.

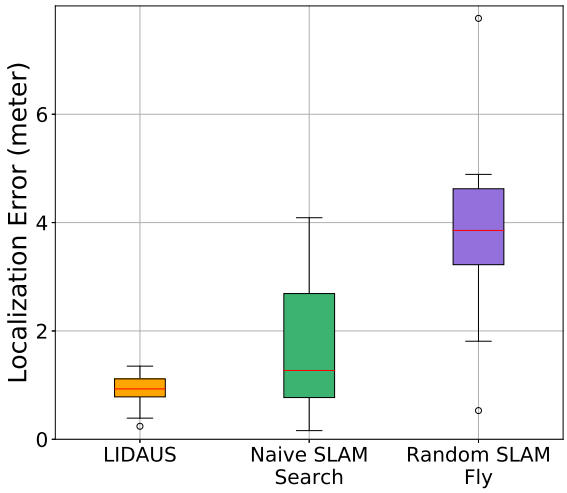


Figure 2.11: Boxplots of all targets' localization errors of the three methods.

To further illustrate LIDAUS, Figure 2.13 shows the Steiner tree built on the ground layer in the final searching stage of the UAV, and the locations of anchor beacons. The red triangle denotes the current position of UAV. In ths figure, star symbols denote the real locations of target beacons, diamond symbols denote the current estimated locations of the not-yet-found target beacons, cross symbols denote the final estimated locations of found target beacons, and light green dots on the branch of the tree denote the deployed anchor beacons. During the searching stage, the UAV flies along the shortest branch of the Steiner tree built upon the estimated locations of target beacons and deploys anchors every $4m$ on its path.

2.3.2 Experiments

We now demonstrate our system design via a real-world experiment. We build a UAV with Bluetooth receiver module based on Crazyflie [Bit20] and use it in our experiments. Eight IoT beacons (i.e., targets) are placed on the ground of an office space as shown in Figure 2.14. All targets and anchors (i.e., anchor beacons) advertise at a rate of 2Hz with a transmission power set to 0 dBm. The UAV is initially placed at the origin (i.e., the main entrance).

Initially the UAV take a Eulerian cycle path to explore the whole space. Due to the furniture in the space, the grid graph derived by our space discretization contains various

Figure 2.12: LIDAUS's localization errors of target beacons along x, y and z directions.

sized grid squares or rectangles, and most of the grid squares have a side length of $2m$. The UAV collects 20 sets of RSSI data from beacons in the space every 50 cm along its flying path. During the exploring stage, no anchor is deployed. Then it takes three *searching stages* to finish its localization of all target beacons. In each searching stage, the UAV follows the shortest branch of the Steiner tree built in that stage when searching for target beacons, and it deploys anchors every $2m$. In addition, when it reaches a target's estimated location as planned, it flies along an extra square path ($1m$ side length) with the estimated target position as center in order to receive stronger RSSI data for the target. In addition, we also conducts experiments with FastSLAM 1.0 and FastSLAM 2.0, following the same flying path as mentioned above.

Figure 2.15 shows that our system's localization estimation error for each target is no more than 1 meter, with an average $0.68m$. Note that for those estimated locations that were out of the range of the space (when using FastSLAM 1.0 or 2.0), they were rounded to the nearest position in the space. This experiment result shows that our system can achieve a sub-meter localization accuracy. Note that both FastSLAM 1.0 and 2.0 performed very

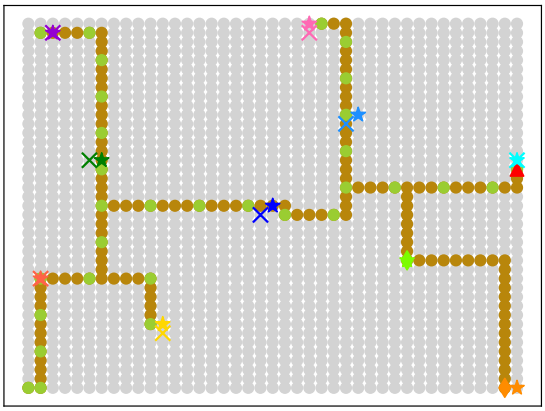


Figure 2.13: Steiner trees built by LIDAUS in its final searching stage.

poorly. In addition, FastSLAM 2.0 is even worse in most cases than FastSLAM 1.0, due to the high noise level of sensor data (i.e., RSSI).

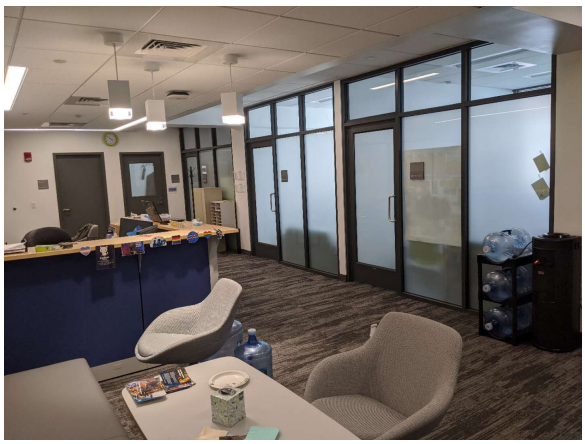


Figure 2.14: The office space where experiments were performed.

2.4 Conclusion

In this chapter, we have discussed our sensing and localization subsystem LIDAUS, an infrastructure-free, multi-stage SLAM system that utilizes a UAV to search and accurately localize IoT devices in a 3D space. The system is based only on the RSSIs of the existing

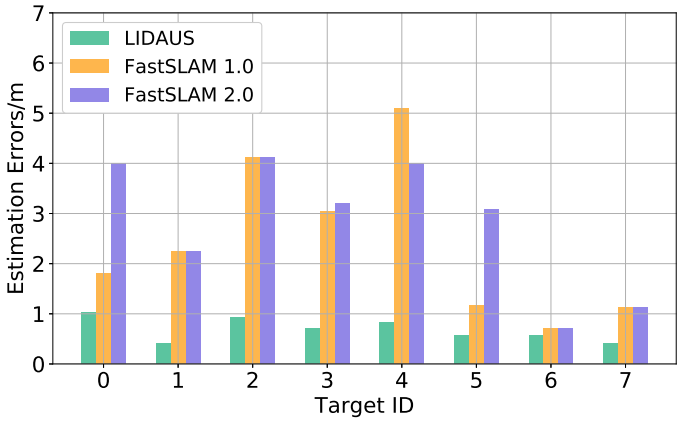


Figure 2.15: Estimation errors of LIDAUS, FastSLAM 1.0 and 2.0.

commodity IoT devices. It can be easily deployed without any customized signal processing hardware and without requiring any fingerprinting or pre-trained model. It can deal with dynamic changing environment. Furthermore, it can work in a complex indoor space where GPS is not available or a space where it is inaccessible for humans. The system contains several novel techniques such as a weighted entropy-based clustering of RSSI observation locations, 3D U-SLAM (and its selective replay) with dynamic deployed anchors, and path planning based on edge covering Eulerian cycles and Steiner tree route for cost minimization. Our extensive simulations and real-world experiments have demonstrated the system's effectiveness of IoT device localization.

Algorithm 2.2.2: Weighted Entropy-based Clustering Algorithm

Input: Table 2.1

Output: $\forall i$, cluster C_i and its center $\hat{b}_i$

INI: Entropy of target beacons $\{S_1 = 0, ..., S_N = 0\}$, $C_i = \emptyset$, $L_{i,k} = 0$, $E_{i,k} = \emptyset$.

for *beacon i in T* **do**

  $\forall i, S_{i,prev} = 0, S_i = 1,$

  while $S_i < S_{th}$ *and* $C_i \subset P_{init}$ **do**

    if $0.5 < Random(0,1)$ **then**

      Decrements $R_{th,c}$ by 1.

    else

      if *not the first iteration* **then**

        Increments R_{dL} by 1.

    Reset entropy $S_i = 0$.

    for p_j *in* P_{ini} **do**

      $R^j_{max} = \max_i\{R^j_1, ..., R^j_i, ..., R^j_N\}$. **if** $R^j_i == R^j_{max}$ *or* $|R^j_i - R^j_{max}| \leq R_{dL}$ **then**

        if $R^j_i \geq R_{th,c}$ **then**

          $C_i = C_i \bigcup \{p_j\}$.

    Calculates the 1-means center position c_i for C_i and locates the cube G_q where c_i is in.

    for p_j *in* C_i **do**

      Find the edge e_k where p_j is located and $E_{i,k} = E_{i,k} \bigcup \{p_j\}$, $L_{i,k} = L_{i,k} + 1$.

    $L_i = \Sigma_k L_{i,k}$.

    for $E_{i,k}$ *of* C_i **do**

      Find the weight w_k of $E_{i,k}$ from the weight look-up table of G_q. Update entropy: $S_i = S_i + (-w_k \frac{L_{i,k}}{L_i} \log(\frac{L_{i,k}}{L_i}))/\log(L_G)$.

  $\hat{b}_i = c_i$.

  return $C = \{C_1, ..., C_M\}$ *and* $\{\hat{b}_1, \hat{b}_2, ..., \hat{b}_M\}$.

Algorithm 2.2.3: Path Planning of Exploring Stage

Input: Grid graph G_{in} on a plane

Output: UAV's initial fly path P_0

if G_{in} *is not an even-degree graph* **then**
    Form a graph G^* from all odd-degree nodes in G_{in}.
    forall *node pairs* u, v *in* G^* **do**
        if $e_{uv} \notin G_{in}$ **then**
            Add e_{uv} in G^*.
            Set weight $w(e_{uv}) = d_{min}(u, v)$.
    Use Blossom V alg. [Kol09] to find the minimum perfect matching M^* in G^*.
    Add edges in M^* into G_{in} to get an even-degree graph G^*_{in}.
Use an Eulerian cycle alg. (e.g., Fleury's alg.) to find a least-cost tour P_0 in G^*_{in}.

Input	Explanation
$\hat{\mathbf{b}}^0 = \{\hat{b}^0_1, ..., \hat{b}^0_M\}$	Initial estimated positions of target beacons' coordinates
R_{th}	Threshold RSSI value
n_{th}	Threshold, eg. a target beacon is labeled as *found* if the number of its RSSIs that is above R_{th} reaches n_{th}

Table 2.2: Input of Algorithm 2.2.4 - Searching Stage with Anchor Steiner Tree.

Algorithm 2.2.4: Searching Stage with Anchor Steiner Tree

Input: Table 2.2

Output: Target beacons' estimated coordinates: $\hat{\mathbf{b}} = \{\hat{b}_1, ..., \hat{b}_M\}$.

INITIALIZE: UAV starts at origin: $(0,0,0)$, let $\hat{\mathbf{b}} = \hat{\mathbf{b}}^0$, found target set $T_{found} = \emptyset$, not-yet-found target set $T_{non} = \mathbb{T}$, set of the coordinates of expected anchors $A_{exp} = \emptyset$, set of the coordinates of deployed anchors $A_{dep} = \emptyset$, set of tree nodes $N_{tree} = \emptyset$.

Discretize the ground plane as a grid graph G with each cell's $SideLength = 1m$.

while $T_{non} \neq \emptyset$ **do**

  Execute Sub-Algorithm 2.2.5.

  forall *step point* $p_j \in P$ **do**

    if $p_j \in A_{exp}$ *and* $p_j \notin A_{dep}$ **then**

      UAV deploys one anchor beacon on the ground at p_j.

      $A_{dep} = A_{dep} + \{p_j\}$.

    if p_j *and a node* $v_i \in T_{non}^{proj}$ *have the same x and y coordinates* **then**

      UAV flies up to $(\hat{b}_{i,x}, \hat{b}_{i,y}, \hat{b}_{i,z},)$ of $\hat{b}_i \in T_{non}$, as v_i is $\hat{b}_i$'s projection, then flies around, and finally down to p_j.

    if *there are* n_{th} *collected RSSIs* R_i^j *for target beacon* t_i *such that* $R_i^j > R_{th}$ **then**

      $F = F + \{t_i\}$.

  $T_{found} = T_{found} + F$.

  Use Algorithm 2.2.2 based on all RSSIs collected so far to get cluster C_i for each target beacon t_i.

  forall $t_i \in T_{non}$ **do**

    Use C_i and R_i^j to replay U-SLAM algorithm.

    Update $\hat{b}_i$ in $\hat{\mathbf{b}}$.

  $T_{non} = T_{non} - F$.

return $\hat{\mathbf{b}}$

Algorithm 2.2.5: Sub-Algorithm for Algorithm 2.2.4.

- Project every target beacon t_i's estimated position $(\hat{b}_{i,x}, \hat{b}_{i,y}, \hat{b}_{i,z})$ in T_{non} to its closest node on G, denoted by $(v_{i,x}, v_{i,y})$, and put all these nodes in T_{non}^{proj}.
- $N_{tree} = T_{non}^{proj} + A_{dep}$.
- Use UAV's current position as root to build a Steiner tree $Tree$ out of G that connects all nodes in N_{tree}.
- Let path $P =$ the shortest branch of $Tree$.
- Generate A_{exp}, the set of expected positions of anchors to be deployed on P.
- $F = \emptyset$.
- UAV flies on the ground plane, following path P and performing U-SLAM algorithm.

CHAPTER 3

3DRIMR/R2P: 3D RECONSTRUCTION AND IMAGING VIA MMWAVE RADAR / RADAR TO POINT CLOUD

In this chapter, we present the design of our second subsystem 3DRMIR/R2P. It is a deep neural network based system that takes as input raw mmWave radar sensing signals scanned from multiple different viewpoints of an object, and then outputs the 3D shape of the object in the format of point cloud.

Recent advances in Millimeter Wave (mmWave) radar sensing technology have made it a great tool in autonomous vehicles [GMJ] and search/rescue in high risk areas [LRZ]. For example, in the areas of fire with smoke and toxic gas and hence low visibility, it is impossible to utilize optical sensors such as LiDAR and camera to find a safe route, and any time delay can potentially cost lives in those rescue scenes. To construct maps in those heavy smoke and dense fog environments, mmWave radar has been shown as an effective sensing tool [LRZ, GMJ].

However, it is challenging to use mmWave radar signals for object imaging and reconstruction, as their signals are usually of low resolution, sparse, and highly noisy due to multi-path and specularity. A few recent work [GMJ, FN, LRZ] have made some progress in generating 2D images based on mmWave radar. In this chapter, we move one step further to tackle a more challenging problem: reconstructing 3D object shapes based on raw sparse and low-resolution mmWave radar signals.

To address the challenge, we propose our reconstruction and imaging subsystem, 3DRIMR/R2P [SHZ21, SZH], a deep neural network architecture based on conditional Generative Adversarial Network (GAN). The architecture consists of two generators in two stages. In Stage 1, generator $\mathbf{G_{r2i}}$ takes 3D radar intensity data as input and generates 2D depth images. In Stage 2, generator $\mathbf{G_{p2p}}$ takes as input a set of multiple 2D depth images (results from Stage 1) of the object (from k different viewpoints) and generates a dense and smooth 3D point

cloud of the object. Generator $\mathbf{G_{r2i}}$ is jointly trained with a discriminator $\mathbf{D_{r2i}}$, which fuses 3D radar intensity data and 2D depth images (either generated or ground truth images). In addition, generator $\mathbf{G_{p2p}}$ is also jointly trained with a discriminator $\mathbf{D_{p2p}}$.

This architecture is designed to combine the advantages of Convolutional Neural Network (CNN)'s convolutional operation and the efficiency of point cloud representation of 3D objects. Convolutional operation can capture detailed local neighborhood structure of a 3D object, and point cloud format of 3D objects is more efficient and of higher resolution than 3D shape representation via voxelization. Specifically, $\mathbf{G_{r2i}}$ applies 3D convolution operation to 3D radar intensity data to generate depth images. Generator $\mathbf{G_{p2p}}$ accepts a 3D object in point cloud format (unordered point set hence convolutional operation not applicable), so that it can process data highly efficiently (in terms of computation and memory) and can express fine details of 3D geometry.

In addition, because commodity mmWave radar sensors (e.g., TI's IWR6843ISK [Texc]) usually have high resolution along range direction even without time consuming Synthetic Aperture Radar (SAR) operation, a 2D depth image generated by $\mathbf{G_{r2i}}$ can give us high resolution depth information from the viewpoint of a radar sensor. Therefore, we believe that combining multiple such 2D depth images from multiple viewpoints can give us high resolution 3D shape information of an object. Our architecture design takes advantage of this observation by using multiple 2D depth images of an object from multiple viewpoints to form a raw point cloud and uses that as the input to generator $\mathbf{G_{p2p}}$.

Although 3DRIMR [SHZ21] introduces an architecture that generates 3D object shapes based on mmWave radar, there is still room for significant improvement in order to produce more satisfactory end results, especially the design of network architecture in 3DRIMR's Stage 2 (named PNG, PointNet [QSM16] based GAN model) can be further improved. Hence, we further introduce Radar to Point Clouds (R2P), to replace the generator network of PNG, and we find that R2P significantly outperforms PNG both quantitatively and visually.

3.1 Related Work

There have been active research in recent years on applying Frequency Modulated Continuous Wave (FMCW) Millimeter Wave (mmWave) radar sensing in many application scenarios, for example, person/gesture identification [VKJ18, YLC], car detection/imaging [GMJ] and environment sensing [LRZ, FN] in low visibility environments. To achieve high resolution radar imaging, usually mmWave radar systems need to work at close distance to objects or they rely on SAR process [MMA14, SM18, GHD16, SMH07].

Our work is inspired by a few recent researches on mmWave radar imaging, mapping, and 3D object reconstruction [GMJ, FN, LRZ, YKH18, QSM16]. Both [GMJ, LRZ] have shown the capability of mmWave radar in sensing/imaging in low visibility environments. Their findings motivate us to adopt mmWave radar in our design. In addition, we intend to add the low-visibility sensing capability on our UAV SLAM system [SXH] for search and rescue in dangerous environments.

Learning-based approaches have been adopted in recent research on 3D object shape reconstruction [YWW17, DRN, SGF16, SM17]. Most of them use voxels to represent 3D objects, as CNN convolutional operation can be easily applied to such data format. However, voxelization of 3D objects or space cannot achieve high resolution due to cubic growth of memory and computation cost. Our architecture's generator at Stage 2 uses point cloud to represent 3D objects which can give us detailed geometric information with efficient memory and computation performance. In addition, in Stage 1 of our architecture, we are able to take advantages of convolutional operation to directly work on 3D radar intensity data.

HawkEye [GMJ] generates 2D depth images of an object based on conditional GAN architecture with 3D radar intensity maps obtained by multiple SAR scans along both elevation and azimuth dimensions. Our architecture adopts this design to generate intermediate results used as inputs to a 3D point cloud generator. Different from [GMJ], we use data of only two snapshots along elevation dimension when using commodity TI IWR6843ISK sensor [Texc]. This is similar to the input data used in [FN], but the network of [FN] out-

puts just a higher dimensional radar intensity map along elevation dimension, not 2D depth images.

The research in [YKH18] and [QSM16] uses point cloud to reconstruct 3D object shapes, which motivates us to adopt PointNet structure in our 3D point cloud generator. Different from those existing work, our architecture's Stage 2 utilizes a conditional GAN architecture to jointly train a generator and a discriminator to achieve high prediction accuracy.

Our proposed reconstruction and imaging subsystem significantly outperforms the existing methods such as PointNet [QSM16], PointNet++ [QYS17], and PCN[YKH18].

3.2 Methodology

3.2.1 Challenges

As we mentioned above, it is a great challenge to use radar sensing to reconstruct 3D object shapes.

First, the characteristics of radar data include sparsity, low resolution, specularity, high noise, and multi-path which can induce shadow reflections and artifacts. As we can see in Figure 3.1, it actually shows a received radar signal of a car, but we cannot see any shape from that. It is very time-consuming to collect full-scale mmWave radar data even use Synthetic Aperture Radar (SAR) technique. Thus, we need to design our system to handle the input data with as few snapshots as possible.

Second, point clouds are unordered and in irregular format, thus, changing the permutations of points in a point cloud will not change its geometry property. For this reason, we cannot find the correspondence between each point in a generated point cloud and its ground truth point cloud.

Third, existing evaluation metrics like Chamfer Distance (CD) and Earth Mover's Distance (EMD), are quite limited when evaluating a point cloud's overall accuracy.

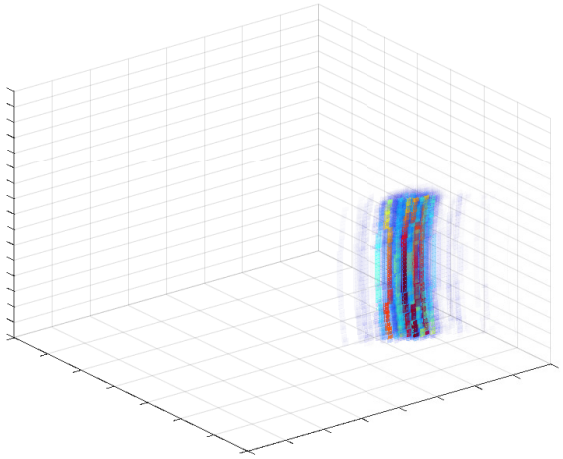


Figure 3.1: An example of the 2-snapshot radar intensity map of a car captured from one view.

3.2.2 Architecture Overview

The architecture consists of two generator networks in two stages. In Stage 1, generator network $\mathbf{G_{r2i}}$ takes 3D radar intensity data as inputs and generates 2D depth images. In Stage 2, generator network $\mathbf{G_{p2p}}$ takes as input a set of k 2D depth images (results from Stage 1) of the object (from k different viewpoints) and generates a dense smooth 3D point cloud of the object. Generator $\mathbf{G_{r2i}}$ is jointly trained with a discriminator network $\mathbf{D_{r2i}}$, which fuses 3D radar intensity data and 2D depth images (either generated or ground truth images). In addition, generator $\mathbf{G_{p2p}}$ is jointly trained with a discriminator network $\mathbf{D_{p2p}}$. These two stages are shown in Figures 3.2 and 3.3.

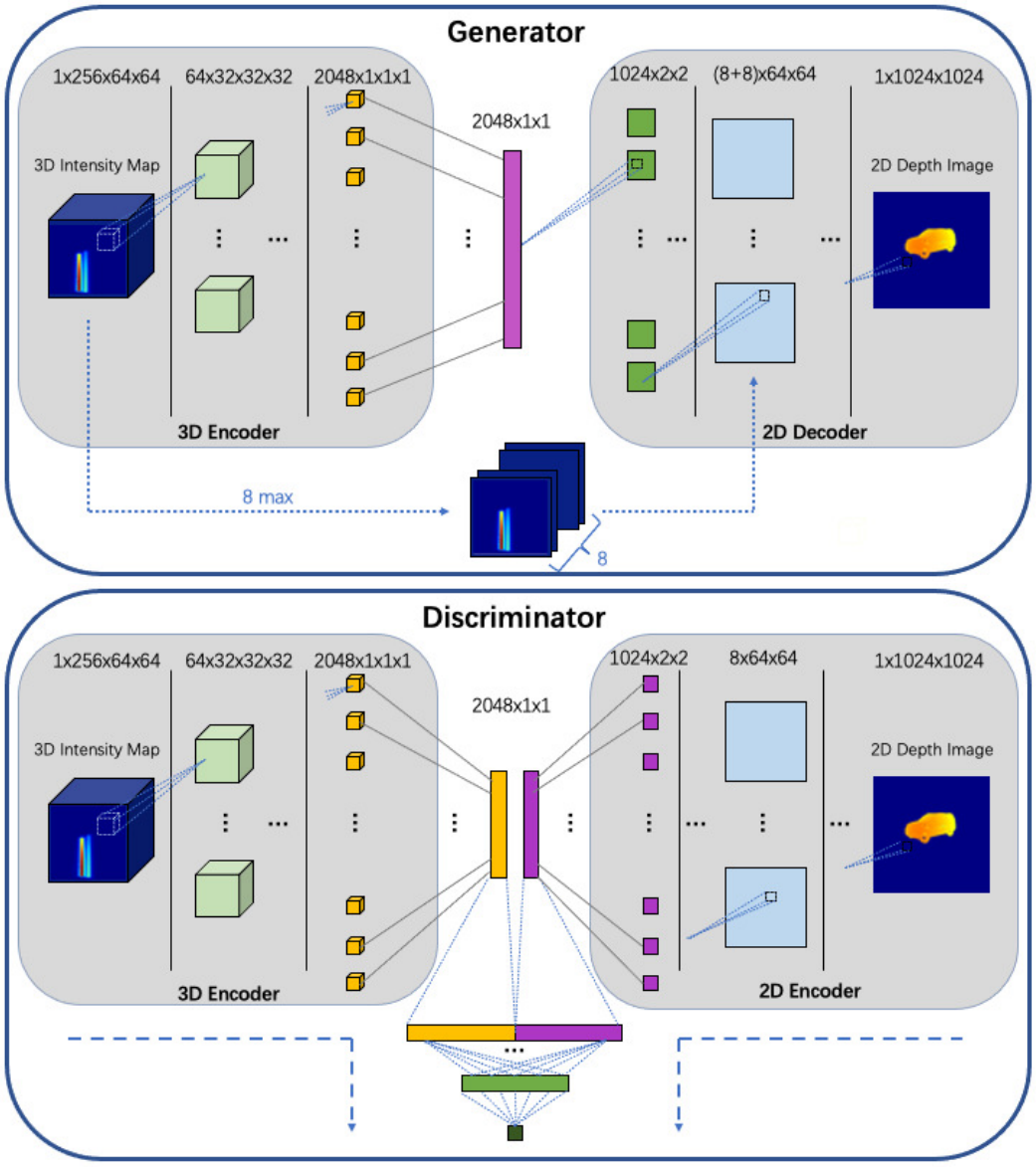


Figure 3.2: 3DRIMR/R2P's Stage 1.

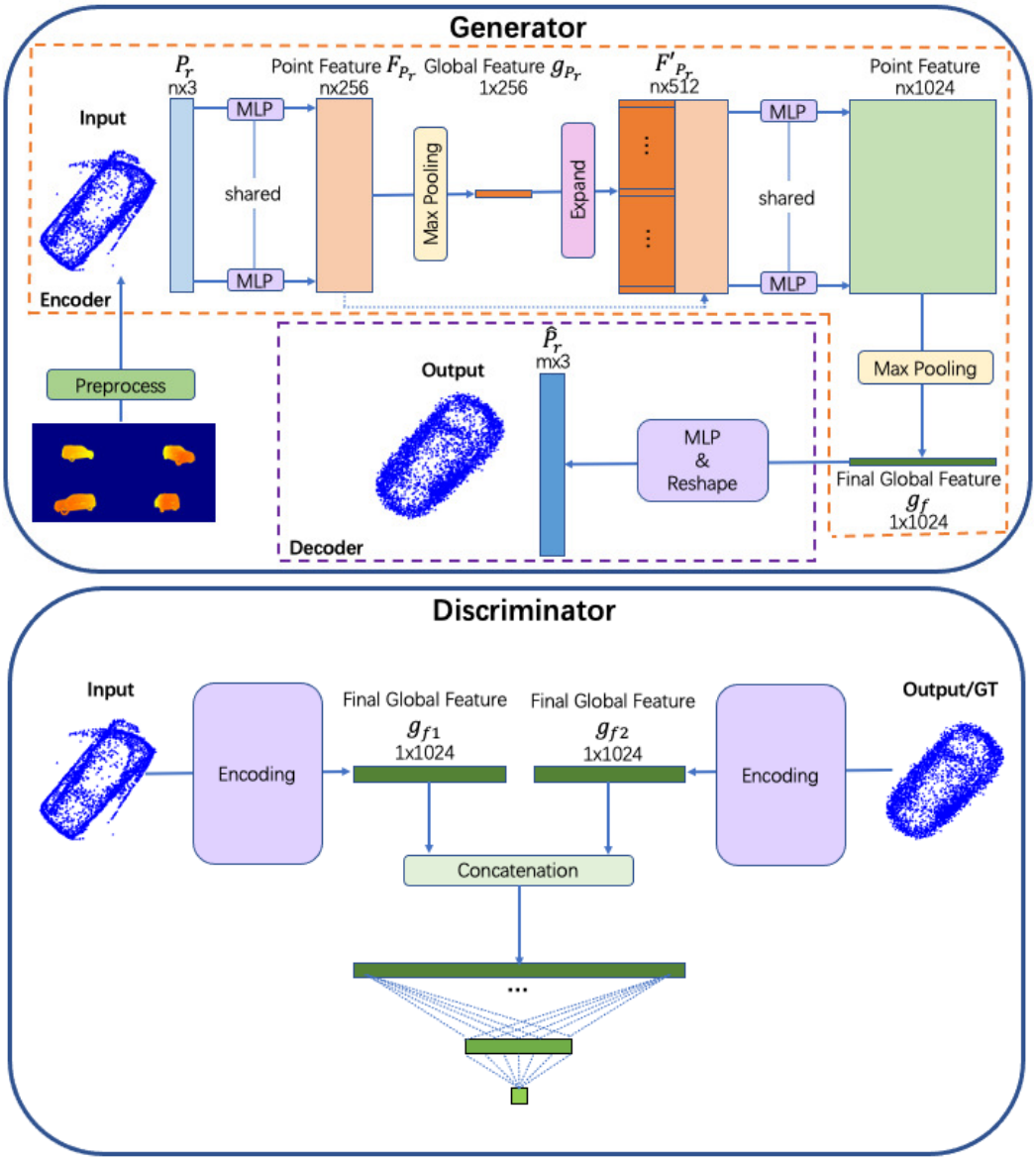


Figure 3.3: PNG's architecture.

Generator $\mathbf{G_{r2i}}$ takes as input a 3D radar energy intensity map of an object and generates a 2D depth image of the object. Let m_r denote a 3D radar intensity map of an object captured from viewpoint v. Let g_{2d} be a ground truth 2D depth image of the same object captured from the same viewpoint. $\mathbf{G_{r2i}}$ generates $\hat{g}_{2d}$ that predicts or estimates g_{2d} given m_r.

Given a set of 3D radar intensity maps $\{m_{r,i}|i = 1,...,k\}$ of an object captured from k different viewpoints $v_1,...,v_k$, generator $\mathbf{G_{r2i}}$ predicts their corresponding 2D depth images

$\{\hat{g}_{2d,i}|i = 1,...,k\}$. Each predicted image $\hat{g}_{2d,i}$ can be projected into 3D space to generate a 3D point cloud.

This subsystem generates a set of k coarse point clouds $\{P_{r,i}|i = 1,...,k\}$ of the object from k viewpoints. It first unions the k coarse point clouds to form an initial estimated coarse point cloud of the object, denoted as P_r, which is a set of 3D points $\{p_j|j = 1,...,n\}$, and each point p_j is a vector of Cartesian coordinates. We choose $k = 4$ in our experiments. Generator $\mathbf{G_{p2p}}$ takes P_r as input, and predicts a dense, smooth, and accurate point cloud $\hat{P}_r$.

Since the prediction of $\mathbf{G_{r2i}}$ may not be completely correct, a coarse P_r may likely contain many missing or even incorrect points. The capability of our method handling incorrect information differs our work from [YKH18] which only considers the case of missing points in an input point cloud.

By leveraging conditional GAN, the architecture's training process consists of two stages: $\mathbf{G_{r2i}}$ and $\mathbf{D_{r2i}}$ are trained together in Stage 1; similarly, $\mathbf{G_{p2p}}$ and $\mathbf{D_{p2p}}$ are trained together in Stage 2. The network architecture and training process are described in more details in Section 3.2.3.

The design of $\mathbf{G_{r2i}}$ is similar to HawkEye [GMJ], but $\mathbf{G_{r2i}}$'s each input radar intensity map only contains two snapshots whereas HawkEye's input radar intensity map has 64 SAR snapshots along elevation (which gives higher elevation resolution but it takes much longer time to generate those 64-snapshots data). In addition, 2D depth images are the final outcomes of HawkEye, but in our architecture, they are only our intermediate results to be used as input for generating 3D object shapes. The design of $\mathbf{G_{p2p}}$ in Stage 2 learns the idea from [YKH18], but with different network architecture design. And unlike [YKH18], we use a conditional GAN architecture. In addition, our Stage 2's generator networks are designed to operate on sparse and only partially correct point clouds.

Next we discuss the two stages of the architecture of 3DRIMR/R2P.

3.2.3 Stage 1: 3D Radar Intensity Map to 2D Depth Image

3.2.3.1 Input and output representation

We first pre-processes a set of 2-snapshot raw mmWave radar data of an object by using three FFTs: *range FFT, azimuth FFT, and elevation FFT*, to generate a 3D radar intensity map m_r of the object from a particular viewpoint. Input m_r is a $64 \times 64 \times 256$ tensor. $\mathbf{G_{r2i}}$'s output is a high-resolution 2D depth image $\hat{g}_{2d}$.

Similar to the hybrid 3D to 2D network architecture of HawkEye [GMJ], our $\mathbf{G_{r2i}}$ is also a 3D-encoder-2D-decoder network. Different from HawkEye [GMJ], each of our input data sample only contains 2 snapshots of SAR radar signals. Therefore, our system can run much faster than HawkEye in practice. Recall that fewer snapshots means lower radar resolution, thus our network handles more challenging inputs, but our network can still produce high-quality outputs.

3.2.3.2 Generator $\mathbf{G_{r2i}}$

This generator follows a standard encoder-decoder architecture [BKC17], which maps a sparse 3D radar intensity map to a high-resolution (same level as that of the depth image taken by a stereo camera) 2D depth image. As shown in Figure 3.2, its encoder consists of six 3D convolution filters, followed by Leaky-ReLU and BatchNorm layers. The 3D feature map is down-sampled and the number of feature channels increases after every convolution layer. Hence, the encoder converts a 3D radar intensity map into a low-dimensional representation. The decoder consists of nine 2D transpose convolution layers, along with ReLU and BatchNorm layers. The 2D feature map is up-sampled and the number of feature channels is reduced after every transpose convolution layer. Passing through the decoder network, low-dimensional vectors are finally converted to a high-resolution 2D depth image.

3.2.3.3 Discriminator $\mathbf{D_{r2i}}$

The discriminator takes in two-stream inputs, 3D radar intensity maps m_r and 2D depth images (either g_{2d} or $\hat{g}_{2d}$). As shown in Figure 3.2, we design two separate networks to encode both of them into two 1D feature vectors. The encoder for 3D radar intensity maps is the same as $\mathbf{G_{r2i}}$'s encoder architecture. The encoder for 2D depth images is a typical convolutional architecture, which consists of nine 2D convolution layers, followed by Leaky-ReLU and BatchNorm layers. The outputs of these two encoders are two 1D feature vectors, which are concatenated and then are passed into a simple convolution network with two 2D convolution layers and Leaky-ReLU and BatchNorm layers in between, and followed by a sigmoid function to output the final classification result.

3.2.3.4 Skip connection

We also apply skip connection in $\mathbf{G_{r2i}}$ to help avoid the gradients vanishing problems in deep networks, and to make the best use of the input data since the range or depth information of an input 3D intensity map can be directly extracted and passed to higher layers. We choose 8 max values of a 3D radar intensity map along range dimension to form a 8-channel 2D feature map. Then, we concatenate this feature map with the feature map produced by the 6-th layer of the decoder, and they together are passed through the remaining decoder layers.

3.2.3.5 Loss function

Our subsystem trains $\mathbf{D_{r2i}}$ to minimize discriminator loss $\mathscr{L}_{\mathbf{D_{r2i}}}$, and trains $\mathbf{G_{r2i}}$ to minimize generator loss $\mathscr{L}_{\mathbf{G_{r2i}}}$ simultaneously. $\mathscr{L}_{\mathbf{D_{r2i}}}$ is the mean Mean Square Error (MSE) of $\mathbf{D_{r2i}}$'s prediction error when the input of includes $(m_r, \hat{g}_{2d})$ and (m_r, g_{2d}). $\mathscr{L}_{\mathbf{G_{r2i}}}$ is a weighted sum of vanilla GAN loss $\mathscr{L}_{GAN}$, $\mathscr{L}_1$ loss between $\mathbf{G_{r2i}}$'s prediction and ground truth, and a perceptual loss $\mathscr{L}_p$. The perceptual loss is calculated by a pre-trained neural network VGG[SZ15] on $\mathbf{G_{r2i}}$'s prediction and ground truth.

$$\mathscr{L}_{\mathbf{G_{r2i}}} = \mathscr{L}_{GAN}(\mathbf{G_{r2i}}) + \lambda_1 \mathscr{L}_1(\mathbf{G_{r2i}}) + \lambda_p \mathscr{L}_p(\mathbf{G_{r2i}}) \tag{3.1}$$

Note that λ_1 and λ_p are two hand-tuned relative weights, and in our simulations, we find that it performs well when the values of them are 1000 and 20 respectively.

3.2.4 Stage 2: Multi-View 2D Depth Images to 3D Point Clouds

For Stage 2, we have developed two different models: PNG - PointNet [QSM16] based GAN Model; and R2P - PCN [YKH18] based Generator Model. We will next discuss these two models.

3.2.4.1 PNG: PointNet based GAN model

As shown in Figure 3.3, Stage 2 consists of another conditional GAN based neural network $\mathbf{G_{p2p}}$ that generates a point cloud of an object with continuous and smooth contour $\hat{P}_r$, based on P_r, a union of k separate coarse point clouds observed from k viewpoints of the object $\{P_{r,i}|i = 1,...,k\}$, which are the outputs from Stage 1. We choose $k = 4$ in our experiments. Note that P_r may contain very noisy or even incorrect points due to the prediction errors in Stage 1.

Both input P_r and output $\hat{P}_r = \mathbf{G_{p2p}}(P_r)$ are 3D point clouds represented as $n \times 3$ matrices, and each row is the 3D Cartesian coordinate (x, y, z) of a point. Note that the input and output point clouds do not necessarily have the same number of points. Generator $\mathbf{G_{p2p}}$ is an encoder-decoder network in which an encoder first transforms an input point cloud P_r into a k-dimensional feature vector, and then a decoder outputs $\hat{P}_r$. The discriminator $\mathbf{D_{p2p}}$ of this stage takes (P_r, P_{true}) or $(P_r, \hat{P}_r)$ pairs and produces a score to distinguish between them.

3.2.4.2 PNG's generator $\mathbf{G_{r2i}}$

The generator also adopts an encoder-decoder structure. The encoder has two stacked PointNet [QSM16] blocks. This point feature encoder design follows similar designs in PointNet [QSM16] and PCN [YKH18], characterized by permutation invariance and tolerance to noises.

The **first block** of $\mathbf{G_{p2p}}$ takes the input P_r, and uses a shared multi-layer perceptron (MLP) to produce a high-dimensional local feature $f_i (i = 1, ..., n)$ for each point p_i in P_r, and hence we can obtain a local feature matrix F_{P_r} with each row being the local feature of the corresponding point. Then, it applies a point-wise maxpooling on F_{P_r} and extracts a high-dimensional global feature vector g_{P_r}. To produce a complete point cloud for an object, we need both local and global features, hence, the encoder concatenates the global feature g_{P_r} with each of the point features f_i and form another matrix F'_{P_r}. The **second block** also consists of a shared MLP and point-wise maxpooling layer, which takes F'_{P_r} as input and produces the final global feature g_f.

The decoder consists of 3 fully connected layers with proper non-linear activation layers and normalization layers. It expands g_f to a $3m \times 1$ dimensional vector, and then reshapes it to a $m \times 3$ matrix. The matrix has m rows and each row represents the Cartesian coordinate (x, y, z) of a point of the predicted point cloud. The fully-connected decoder design is good at predicting a sparse set of points which represents the global geometry of a shape.

3.2.4.3 PNG's discriminator $\mathbf{D_{p2p}}$

Note that the number of points of input P_r and output $\hat{P}_r$ and ground truth P_{true} can be different, and the points of a point cloud are unordered. Therefore, we need to design a discriminator with two-stream inputs. As shown in Figure 3.3, these two inputs pass through the same architecture as used in $\mathbf{G_{p2p}}$ and are converted into their own final global feature, which are concatenated and fed into 2 fully connected layers to produce a score, which is used to indicate whether the input is real or generated point cloud.

3.2.4.4 PNG's loss function

For this GAN based network, we train $\mathbf{D_{p2p}}$ to minimize $\mathscr{L}_{\mathbf{D_{p2p}}}$, and train $\mathbf{G_{p2p}}$ to minimize $\mathscr{L}_{\mathbf{G_{p2p}}}$ simultaneously. $\mathscr{L}_{\mathbf{D_{p2p}}}$ is calculated in the same way as described in Stage 1, but $\mathscr{L}_{\mathbf{G_{p2p}}}$ is different due to the characteristics of point clouds. $\mathscr{L}_{\mathbf{G_{p2p}}}$ is a weighted sum, consisting of $\mathscr{L}_{GAN}(\mathbf{G_{p2p}})$, Chamfer loss $\mathscr{L}_{cf}$ between predicted point clouds and the ground truth, and IoU loss $\mathscr{L}_{iou}(\mathbf{G_{p2p}})$.

Chamfer distance [YKH18] is defined as:

$$d_{cf}(S_1,S_2)=\frac{1}{|S_1|}\sum_{x\in S_1}\underset{y\in S_2}{min}\|x-y\|_2+\frac{1}{|S_2|}\sum_{y\in S_2}\underset{x\in S_1}{min}\|y-x\|_2 \tag{3.2}$$

In our case, Chamfer loss is calculated as:

$$\mathscr{L}_{cf}(\mathbf{G_{p2p}})=d_{cf}(\hat{P}_r,P_{true}) \tag{3.3}$$

We define IoU of two point clouds as the ratio of the number of shared common points over the number of points of the union set of both two point clouds. First, we voxelize the space, and all the points in the same voxel will be treated as the same point. We define the number of voxels occupied by $\hat{P}_r$ and P_{true} as $\hat{V}_r$ and V_{true} respectively. Then, the IoU can be calculated as:

$$IoU=\frac{\hat{V}_r\cap V_{true}}{\hat{V}_r\cup V_{true}+\varepsilon} \tag{3.4}$$

Note that ε is a very small number such as 10^{-6} to avoid dividing by zero. Since IoU is within range 0 and 1, we define our IoU loss as:

$$\mathscr{L}_{IoU}(G)=1-IoU \tag{3.5}$$

$\mathscr{L}_{\mathbf{G_{p2p}}}$ is given by Eqn. (3.6), and $\lambda_{d_{cf}}$ and λ_{iou} are also need to be hand-tuned. In our experiments, we set the values of them to 100 and 10 respectively.

$$\mathscr{L}_{\mathbf{G_{p2p}}}=\mathscr{L}_{GAN}(\mathbf{G_{p2p}})+\lambda_{d_{cf}}\mathscr{L}_{cf}(\mathbf{G_{p2p}})+\lambda_{iou}\mathscr{L}_{iou}(\mathbf{G_{p2p}}) \tag{3.6}$$

3.2.4.5 R2P: PCN based generator model

Even though PNG can give some promising results, its design of generator network and loss function are still not quite satisfactory. Specifically, the edges of generated point clouds are still blurry and their points tend to be evenly distributed in the space, and thus do not give a clear sharp shape structure. To address those issues, we further introduce R2P (Radar to Point Cloud). Compare it with PNG, we find that R2P significantly outperforms PNG both quantitatively and visually.

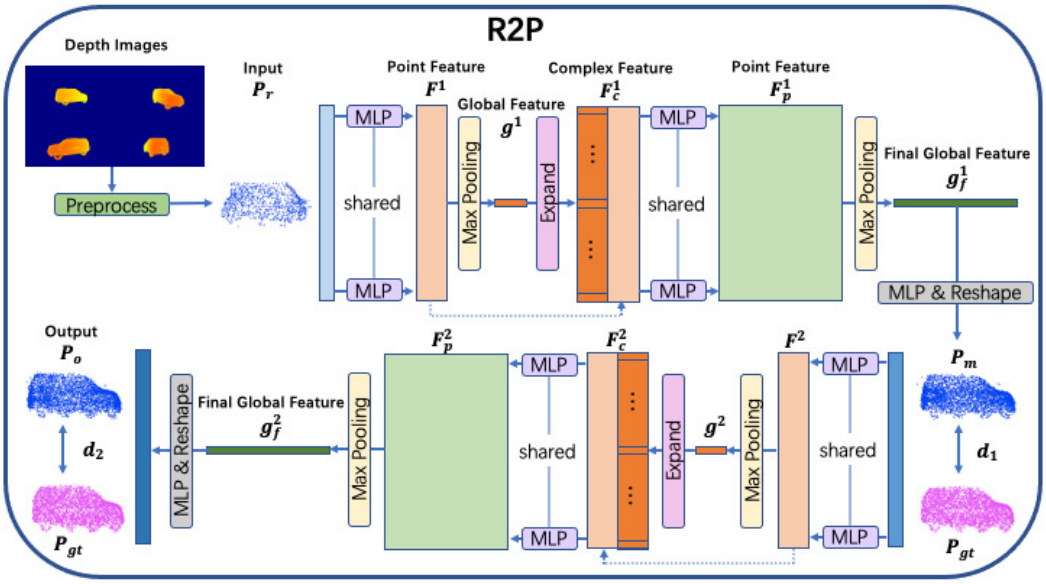


Figure 3.4: R2P network architecture.

R2P's input P_r and output P_o are 3D point clouds represented as $n \times 3$ matrices, where n is the number of points in the point cloud, and each row represents the 3D Cartesian coordinate (x, y, z) of a point. Note that the output point cloud generated from R2P has a larger number of points than the input point cloud, i.e., R2P can reconstruct a dense and smooth point cloud from a sparse one. R2P consists of two sequential processing blocks, and both blocks share the same encoder-decoder network design. The first block takes the raw input point cloud P_r to produce an intermediate point cloud P_m, and then the second block processes P_m to generate the final output P_o. In each block, the encoder takes its input point cloud and converts it to a high dimensional feature vector, and the decoder takes this feature vector and converts it to the intermediate or final output point cloud.

Encoder. As shown in the upper part of Figure 3.4, in the first block, the encoder gets P_r as input, and passes it to a shared MLP to extend every point p_i in P_r to high dimensional feature vector f_i and form the point feature matrix F^1. This shared MLP is a network with two linear layers with BatchNorm and ReLU in between. Then, the encoder applies a point-wise maxpooling on F^1 and extracts a global feature vector g^1. To produce a complete point cloud for an object, we need both local and global features, therefore, the encoder concatenates the global feature with each of the point features f_i^1 (of F^1) and form the

complex feature matrix F_c^1. Then another shared MLP is used to produce point feature matrix F_p^1. Then, another point-wise maxpooling will perform on F_p^1 to extract the final global feature g_f^1.

Decoder. Decoder takes the final global feature g_f^1 as input, and then passes it to a MLP, which consists of three fully-connected layers with ReLU in between. After this MLP, the final global feature is converted to $3m \times 1$ vector (where m is the number of points in P_m), and then reshaped to $m \times 3$ matrix which represents the point cloud P_m.

As shown in the lower part of Figure 3.4, the second block's design is similar to the first block. The encoder of the second block takes P_m as input to produce a final global feature g_f^2, and then the decoder generates the final output point cloud P_o.

3.2.4.6 R2P's loss function

Due to the irregularity of point clouds, it is quite difficult to choose an effective loss function to indicate the difference between a generated point cloud and its corresponding ground truth point cloud. There are two popular metrics to evaluate the difference between two point clouds: Chamfer Distance (CD) [YKH18] and Earth Mover's Distance (EMD) [YKH18].

CD calculates the average closest distance between two point clouds S_1 and S_2. The symmetric version of CD is defined as:

$$CD(S_1,S_2) = \frac{1}{|S_1|}\sum_{x\in S_1} \min_{y\in S_2} \|x-y\|_2 + \frac{1}{|S_2|}\sum_{y\in S_2} \min_{x\in S_1} \|y-x\|_2 \tag{3.7}$$

EMD can find a bijection $\phi : S_1 \to S_2$, which can minimize the average distance between each pair of corresponding points in two point clouds S_1 and S_2. EMD is calculated as:

$$EMD(S_1,S_2) = \min_{\phi:S_1\to S_2} \frac{1}{|S_1|}\sum_{x\in S_1} ||x-\phi(x)||_2 \tag{3.8}$$

Note that the generator generates the intermediate point cloud P_m after its first encoder-decoder block, and then generates the final output point cloud P_o after the second block. Hence, we design our loss function to evaluate both generated point clouds. That is, our loss function is defined as a weighted sum of the loss of the first block $d_1(P_m, P_{gt})$ and the

loss of the second block $d_2(P_o, P_{gt})$.

$$\mathscr{L}(R2P) = d_1(P_m, P_{gt}) + \alpha d_2(P_o, P_{gt}) \tag{3.9}$$

Both d_1 and d_2 can be either CD or EMD, or some weighted combination of them. Note that unlike CD, EMD needs to find a bijection relationship between two point clouds, which is an optimization problem and hence computationally expensive, especially when the point clouds have large amounts of points. Moreover, this bijection also requires these two evaluated point clouds have the same number of points.

3.3 Implementation and Experiments

We implement our models and conduct experiments to test 3DRIMR/R2P with real and synthesized data.

3.3.1 Data Collection and Generation

To the best of our knowledge, there is no publicly available dataset of mmWave radar sensing, therefore, we conduct experiments to collect real data. However, since real data collection is time consuming, we also augment our dataset via synthesizing mmWave data as done in [GMJ].

Real radar data collection. We collect real radar data with Synthetic Aperture Radar (SAR) operation using an IWR6843ISK [Texc] sensor and a data capture card DCA1000EVM [Texa]. To generate a full-scale 3D energy intensity data as a baseline, we first conduct SAR radar scans with a 24×64 virtual antenna array of the sensor by sliding the sensor horizontally and vertically through a customized slider. Note that the each sensor's scan has a range or depth of 256 units. We expand a $24 \times 64 \times 256$ 3D data cube (obtained via a SAR scan) to a full-scale $64 \times 64 \times 256$ 3D data cube, which can be regarded as consisting of 64 snapshots stacked vertically, with each snapshot being a data plane of size 64×256. Note that the full-scale data is only used as a base reference in validating our deep learning system's effectiveness. When training and testing model, each actual input data sample has

only 2 out of 64 snapshots. Getting 2-snapshot data is a much faster process than getting full-scale 64-snapshot data.

Synthesized radar data. Similar to [GMJ], we use 3D CAD models of cars [FDU12] to generate synthesized radar signals. We first generate 3D point clouds based on those CAD models, and then translate and rotate them to simulate different scenarios. Then we select points in each point cloud as radar signal reflectors. Next we simulate received radar signals in a receiver antenna array based on our radar configuration.

Generating 3D radar energy intensity data. For both real and synthesized radar data, we perform FFT along all three dimensions, namely azimuth ϕ, elevation θ, and range r. Note that a radar 3D data cube is measured in degree along azimuth ϕ and elevation θ dimension. We convert a data cube into Cartesian coordinate system so that it matches the coordinate system of the depth camera. This is different from [GMJ], which directly used original data in spherical coordinate system (hence introduced additional errors).

Ground truth 2D depth images and point clouds. For real data, ground truth depth images are obtained via a ZED mini camera [STE]. Note that ZED mini camera is widely used in mobile robots and VR sets. For synthesized data, based on the derived point clouds of CAD models, we generate ground truth depth images after perspective projection with appropriate camera settings and viewpoints positions. Similarly, we generate ground truth 3D point clouds.

Real data collection environment setup. We collect radar data for a L-shape box placed in our lab, in which we let a mobile robot carry a tall flag and it moves around in the room to simulate humans walking around. We also augment this dataset by generating synthesized data for the same L-Box via CAD models placed in the same settings as used in the real data collection. Figure 3.5 shows the scene where our real experiments are performed.

3.3.2 Model Training and Testing

We conduct experiments of 3DRIMR on cars (large objects with average size of $445cm \times 175cm \times 158cm$), and a L-shape box (a small object with size of $95cm \times 73cm \times 59cm$). Furthermore, we conduct experiments of R2P on a larger dataset including 5 different cat-

Figure 3.5: A lab space where our real experiments are performed.

egories of objects, namely industrial robot arm, car, chair, desk, and L-shape box (L-box). All their datasets consists of both synthesized data and real data collected from experiments, and we use real data only in the experiment of L-shape box.

3.3.2.1 Stage 1

The training data of *car* consists of 8 different categories of car models, and for each car model, we collect data of 300 different car orientations from 4 views, so in total $300 \times 4 \times 8 = 9600$ data samples are used in training, and another 6400 data samples are used to test the model. We train the network of *L-box* based on 320 real data samples and 4800 synthesized data samples and test the model using 80 real data samples and 6400 synthesized data samples. We extend the dataset to include more data from *other categories* (industrial robot arms, chairs, and desks) of objects to compare the models in Stage 2. We train each model based on 4000 data samples and test each model using 6000 data samples. We train each model with batch size 4 for 200 epochs. The learning rate is 2×10^{-4} for the first 100 epochs, and then linearly decreased to 0 in the rest 100 epochs.

3.3.2.2 Stage 2

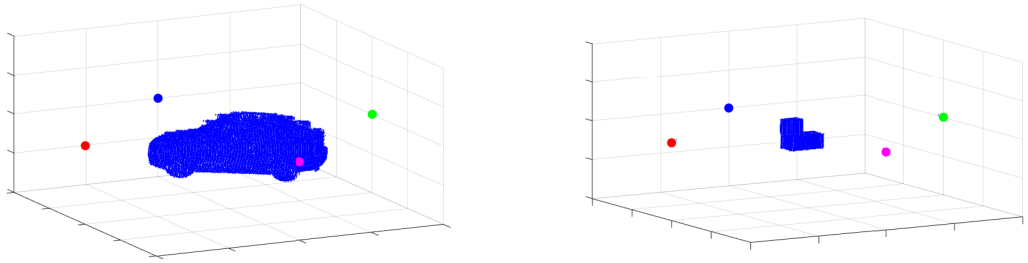
PNG. We use the 2D depth images generated in Stage 1 to form a dataset in Stage 2 of coarse point clouds. The dataset of *car* includes 1600 point clouds with 200 point clouds of each car model. We train the model using 1520 point clouds and test it using the remaining 80 point clouds. For the model of *L-box* in Stage 2, we train it based on 10 point clouds generated by Stage 1's output of real data and 1500 point clouds generated by Stage 1's output of synthesized data. We test the model using the rest 110 data samples (10 from real and 100 from synthesized data). Similarly, we train each model with batch size 4 for 200 epochs. The learning rate is 2×10^{-4} for the first 100 epochs, and then linearly decreased to 0 in the rest 100 epochs.

R2P. As mentioned above, we have a extended dataset of coarse and sparse input point clouds for the 5 different categories of objects, which are industrial robot arm, car, chair, desk, and L-shape box (L-box). Each input point cloud has 1024 points and an intermediate/final output point cloud has 4096 points. We train our proposed network model for each category independently. For each object category, we train the model based on 1400 pairs of point clouds for 200 epochs with batch size 2. The learning rate for the first 100 epochs is 2×10^{-4} and linearly decreases to 0 in the rest 100 epochs. Then we test the model using the remaining 100 point clouds.

3.4 Results

3.4.1 Results of Stage 1

We compare 3DRIMR/R2P's results in Stage 1 against HawkEye [GMJ] as HawkEye's goal is to generate 2D depth images only. Here we only list the results of objects in 2 categories, car and L-box. Example scenes of car and L-box experiments are shown in Figure 3.6(a) and Figure 3.6(b) respectively. Four colored dots in each figure show four viewpoints of radar sensor or camera.



(a) An example scene of car experiment.

(b) An example scenes of L-box experiment.

Figure 3.6: Example scenes of car and L-box experiments.

We show our model's performance in car experiments in Figure 3.7. The top row shows the 3D radar intensity data from 2 snapshots only, the middle row shows the outputs from our Stage 1, and the bottom row shows the ground truth depth images. We see that visually our model can accurately predict an object's shape, size, and orientation, and it can also estimate the distances between an object's surface points and radar receiver. Although most of our training data is synthesized data, our model can still predict depth images well based on the real data. We observe similar performance from the experiments with L-box.

Note that we only use 2-snapshot 3D intensity data in our experiments whereas HawkEye [GMJ] uses 64 SAR snapshots along elevation, but our system's Stage 1 results are still comparable with those of HawkEye. To prove that, we evaluate the same metrics as those in [GMJ]. Table 3.1 shows the median errors of 3DRIMR/R2P's Stage 1 results compared against those in [GMJ]. Since the size of L-box is different from the size of cars, we scale it to the average size of cars when calculating length, width, and height errors. In the comparison, we calculate the average errors of HawkEye's three experiment settings (i.e., clean air, fog, and synthesized data). Table 3.1 shows that our system outperforms HawkEye in terms of range and orientation prediction. Compared with HawkEye, our car predication's length error is 29% larger, and our L-box prediction's length error is about 50% smaller. The errors in height and % Fictitious Reflections in both our car prediction and HawkEye's

are very similar, but these errors in our L-box prediction are 67% and 89% smaller. The errors in width and % Surface Missed of ours and HawkEye are similar.

In sum, our Stage 1's performance is comparable with that of HawkEye. Recall that Stage 1's results are just our intermediate results. Next we evaluate our final prediction results in Section 3.4.2 .

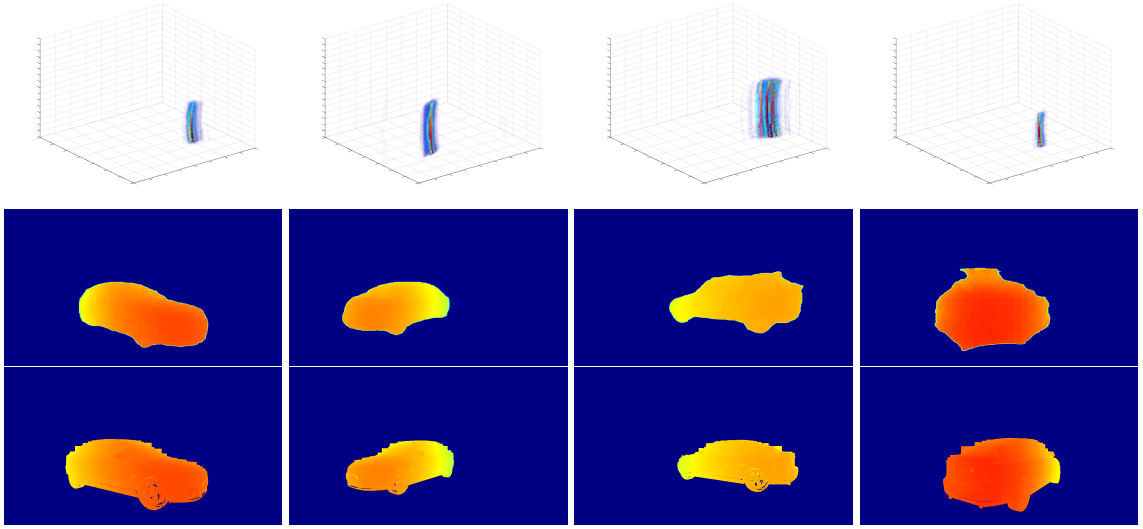


Figure 3.7: Reconstruction and imaging subsystem 3DRIMR/R2P. Stage 1's qualitative performance in car experiments.

Method	Range Err.	Length Err.	Width Err.	Height Err.	Orientation Err.	FR	SM
our-Cars	16 cm	84 cm	37 cm	10 cm	**4.8**°	1.9 %	15.4 %
our-Lbox-scaled	**8 cm**	**32 cm**	**34 cm**	**3 cm**	12.4°	**0.2** %	16.1 %
HawkEye-avg	34 cm	65 cm	37 cm	9 cm	28.7°	1.8 %	**12.8 %**

Table 3.1: Quantitative results of 3DRIMR/R2P's Stage 1, compared with HawkEye [GMJ]. (FR: % Fictitious Reflections; SM: % Surface Missed.)

3.4.2 Results of Stage 2

Recall that in Stage 2, we have proposed two models: PNG and R2P. We then discuss both of their results in the following.

3.4.2.1 PNG's results.

Figure 3.8 shows our final output point clouds compared with input point clouds and groud truth point clouds. In Figure 3.8, top row shows the input point clouds, middle row shows the output point clouds, and bottom row shows the ground truth point clouds. We can see that due to predict errors in Stage 1, the input point clouds to Stage 2, which are simply the unions of coarse point clouds from 4-view depth images, are discontinuous and have many incorrect points. To the best of our knowledge, the state of art research only studies reconstructing point clouds from sparse and partial point clouds with missing points, but not dealing with systematic incorrect points which might give slanted or even wrong shapes. However, our system can perform reasonably well even based point clouds with incorrect points. As shown in Figure 3.8, the point clouds after PNG's are continuous and complete, compared with the input point clouds.

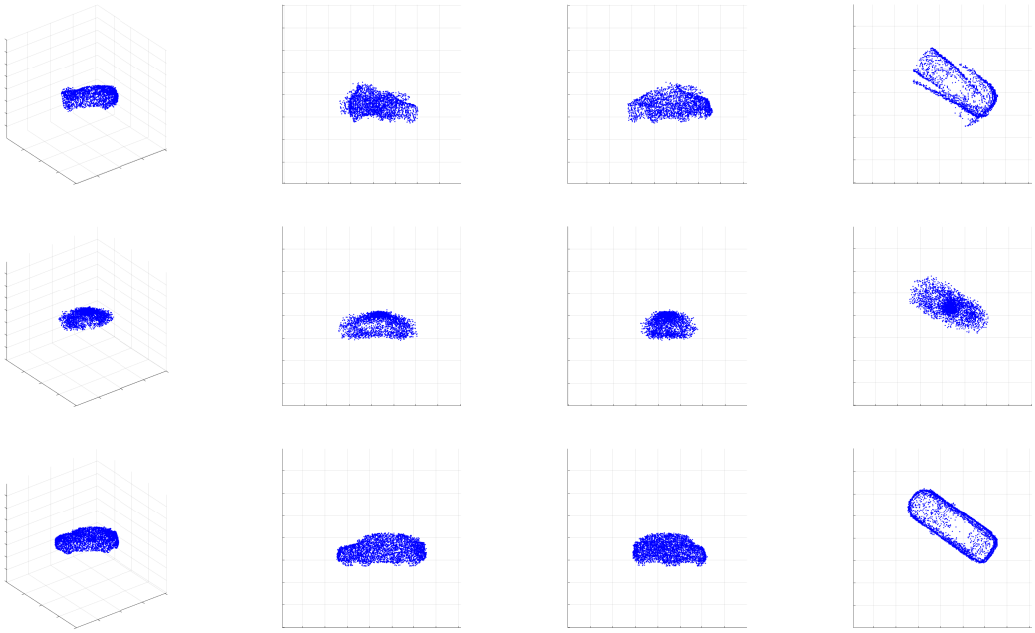


Figure 3.8: PNG's point clouds in car experiments.

We compare PNG's performance against two baseline methods.

Method	Chamfer Distance		IoU		F-score	
	avg.	std.	avg.	std.	avg.	std.
PNG-Cars	**0.0789**	**0.0411**	**0.0129**	**0.0052**	**0.0841**	**0.0322**
DPN-Cars	0.1164	0.1358	0.0121	0.0061	0.0806	0.0389
CLN-Cars	0.1485	0.1843	0.0120	0.0071	0.809	0.0447
PNG-Lbox	**0.0205**	**0.0036**	**0.0937**	**0.0124**	**0.5575**	**0.0961**
DPN-Lbox	0.0303	0.0265	0.0862	0.0268	0.4808	0.1576
CLN-Lbox	0.0367	0.0558	0.0845	0.0281	0.5103	0.1774

Table 3.2: Quantitative results of PNG, compared with two baseline methods.

- **Double-PointNet Network (DPN)**. This is modified from PNG's structure by removing $\mathbf{D_{p2p}}$, i.e., this is not a GAN architecture. This one is used to examine the effectiveness of GAN in training.

- **Chamfer-Loss-based Network (CLN)**. This is the network modified from PNG's structure by removing IoU loss from training process. This one is used to examine whether IoU loss can help with the training.

Table 3.2 shows the average and standard deviation of Chamfer Distances, IoUs, and F-Scores of PNG and two baseline methods. We can see that in both cars and L-box experiments, PNG always performs best among three methods in terms of all the evaluation metrics. These results show that GAN architecture can indeed improve the generator's performance. In addition, Chamfer-Loss-based Network has the worst performance, which shows that our IoU loss can significantly improve the network's performance.

3.4.2.2 R2P's results.

To the best of our knowledge, except for our previous model PNG, there are no other point cloud-based networks to reconstruct smooth, dense, and accurate point clouds from *point clouds with many incorrect and inconsistent points*, e.g., the union of multiple coarse point clouds which are converted from various 2D depth images of an object that are possibly

inaccurate and inconsistent between themselves in terms of orientation and shape structure details. For example, a generated 2D depth image from radar data [GMJ, SHZ21] can possibly have a car's left and right sides switched, or it can be shaped like a similar but different car with different shape details. Note that the existing works (e.g., [YKH18]) on this subject usually do not have such strong inaccuracy assumption on their inputs except missing points. Nevertheless we compare R2P against five related baseline methods. They are popular point cloud-based generative models, mainly designed for the purposes of classification, segmentation, and point clouds completion, assuming input data is either sparse or there are missing points. To ensure fair comparison, we train all six models (including baselines and R2P) based on the same training and testing datasets. These baseline methods include:

- **PointNet**. It is introduced in [QSM16]. We choose 1024 as the dimension of global feature of PointNet. The loss function is a combination of CD and EMD.
- **PointNet++**. We use the same encoder architecture of PointNet++ [QYS17] for classification with three Set Abstraction (SA) modules to get a 1024-dimension global feature, followed by a decoder consists of 3 fully-connected layers. The loss function is also a combination of CD and EMD.
- **PCN_CD**. We use the same architecture of PCN[YKH18], and set the grid size as 2. The number of points in the coarse and detailed output point clouds are 1024 and 4096 respectively. Note that in this method, both d_1 and d_2 are CD.
- **PCN_EMD**. In this method, the architecture of the network is same as PCN_CD, but d_1 is EMD.
- **PNG**. This is the only architecture designed for point clouds reconstruction from inaccurate input point clouds among the 5 baselines. To ensure fair comparison, we use the combination of CD and EMD as its loss function.

We compare the performance of the two variants of our proposed R2P, labeled as R2P_CD (both d_1 and d_2 use CD) and R2P_EMD (both d_1 and d_2 use EMD), with the

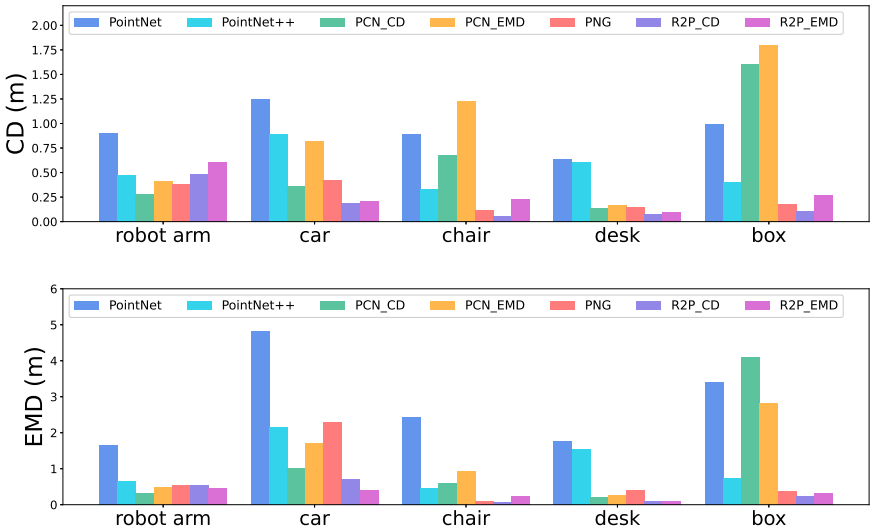


Figure 3.9: Quantitative comparison on datasets of 5 different objects using baseline methods and our method.

five baseline methods mentioned above in terms of CD and EMD. The results are shown in Figure 3.9. We can see that except the case of robot arm, R2Ps always have the smallest CD or EMD loss among all the methods. Even for the robot arm, the performance of R2Ps are similar to the other five methods.

We further compare the results of all six methods by visually examining their output points clouds, shown in Figure 3.10. Since the output point clouds of PointNet, PCN_CD, R2P_CD are very similar to those of PointNet++, PCN_EMD, R2P_EMD, respectively. they are not shown here due to space limitations. R2P_EMD can give the best shape reconstruction of objects among all the methods. Especially for some small objects with fine shape details like chairs, only our method R2P_EMD can reconstruct an object with accurate shape.

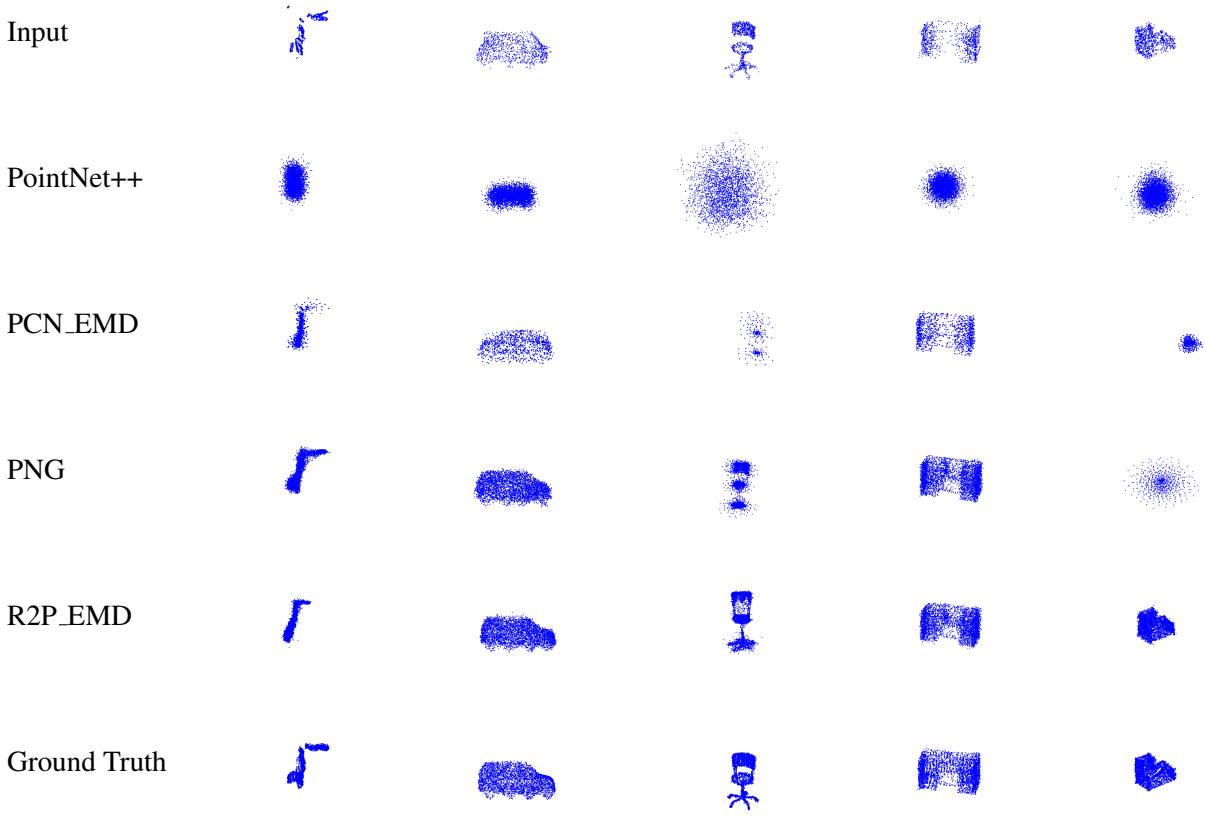


Figure 3.10: Comparison of the generated point clouds of different objects using different methods.

3.5 Discussion

3.5.1 Performance of Different Loss Functions in Stage 2

Loss function plays a quite important role in model training. A well-designed loss function can not only speed up the training process, but also affect the performance of a deep learning model. On the other hand, a bad loss function may not converge even with a well-designed network model. Hence, we need to carefully design our loss function used when training our model. However, existing popular evaluation metrics to compare point clouds are CD and EMD, which can only evaluate the overall distance between a pair of point clouds but cannot effectively capture the similarity of their shapes.

To search for a good loss function for R2P, we conduct a series of experiments with different combinations of these two metrics. Except the loss functions used in training are different, all other settings are all the same for these experiments. The quantitative results are shown in Figure 3.11. We explore 5 different combinations of d_1 and d_2. L1 means both d_1 and d_2 are CD, L2 means both d_1 and d_2 are EMD, L3 means both d_1 and d_2 are CD+EMD, L4 means d_1 is CD and d_2 is EMD, and L5 means d_1 is EMD and d_2 is CD. We can see that if we only use EMD in the loss function, which is L2 in the figure, the CD between output and ground truth point clouds will be large; and except in the box experiment, if we only use CD in the loss function, which is L1 in the figure, the EMD between output and ground truth point clouds will also be large. Hence, it makes sense to use the combination of CD and EMD, which is L3 in the figure, which gives both small CD and EMD. However, since calculating EMD is very expensive and time-consuming, using CD to evaluate intermediate output P_m and EMD to evaluate final output P_o, which is L4 in the figure, is also a good choice for the sake of efficiency.

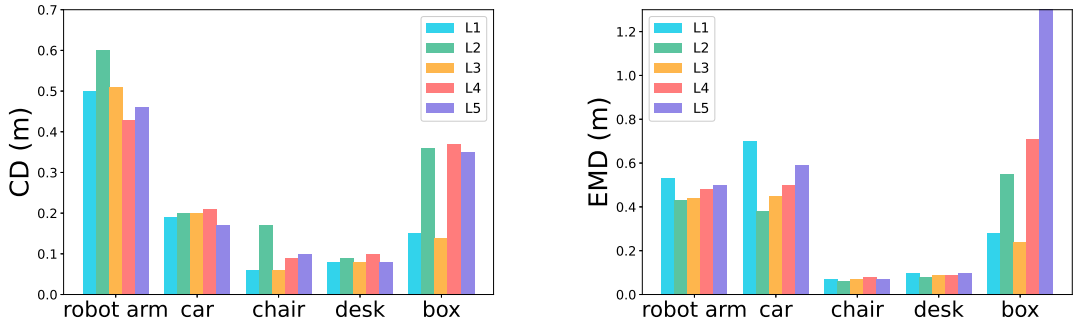


Figure 3.11: Comparison of different loss functions.

3.5.2 Limitations of CD and EMD

CD and EMD are important metrics to evaluate the difference between two point clouds. Generally speaking, small CD/EMD value means better reconstruction performance, and vice versa. However, due to the irregular format and lack-of-order information of point clouds, these two metrics are quite limited in terms of accurately indicating the shape dif-

ference between two point clouds, and hence cannot accurately describe the performance of a point cloud reconstruction method. Sometimes, a reconstructed point cloud may have larger CD or EMD though its shape is more similar to the ground truth point cloud. For example, as we can see in Figure 3.9, the EMD of chair using 3DRIMR is smaller than R2P_EMD, but from Figure 3.10, we can see that the output of R2P_EMD has more accurate shape of a chair. Hence, we should not focus only on the values of CD or EMD when evaluating a method's reconstruction performance.

We have also conducted a series of experiments to explore different designs of our network architecture, e.g., using different pooling methods to extract global features, applying discriminators during the training process, and deeper network architecture with more layers. Nevertheless, we find that the architecture shown in Figure 3.3 performs the best and is efficient.

3.6 Conclusion

In this chapter, we have discussed our second subsystem 3DRIMR/R2P, a deep learning architecture that reconstructs 3D object shapes in point cloud format based on raw mmWave radar signals. 3DRIMR/R2P is a conditional GAN based architecture. This architecture takes advantage of 3D convolutional operation and point cloud's efficiency of representing 3D shapes. Our experiments have shown its effectiveness in reconstructing 3D objects based on two snapshots of a commodity mmWave sensor. 3DRIMR/R2P can generate 3D objects in the form of smooth, dense, and highly accurate point clouds with fine geometry details, the generator of Stage 2 of 3DRIMR/R2P are directly converted from the 2D depth images that are generated from raw mmWave radar sensor data, and thus characterized by mutual inconsistency or errors in terms of orientation and shape. We have demonstrated with extensive experiments that R2P significantly outperforms existing methods such as PointNet/PointNet++ and PCN. In addition, we have shown the importance of loss function design in the training of models for reconstructing point clouds.

CHAPTER 4

APPLICATION OF 3DRIMR/R2P TO RECONSTRUCT MULTIPLE OBJECTS

Frequency Modulated Continuous Wave (FMCW) Millimeter Wave (mmWave) radar sensing recently has been shown as an effective sensing tool in low visibility environment, thus making it a promising sensing technique in autonomous vehicles [GMJ] and search/rescue scenarios [LRZ]. The capability of 3D object reconstruction is important in a search and rescue scenario, e.g., firefighting scenes, where heavy smoke makes optical sensing not practical. However, it is quite challenging to reconstruct 3D object shapes based on mmWave radar data because the data is usually of low resolution, sparsity, specularity, and large noise due to multi-path effects. As we introduce in the last chapter, recent work on 3D reconstruction has made some progress in this direction [GMJ, LRZ, SHZ21, SZH], focusing on single object reconstruction.

In this chapter, we go one step further to explore the feasibility of reconstructing 3D shapes of *multiple objects* in a space (which is more challenging than single object reconstruction), based on mmWave radar data collected from a sensor mounted on a UAV. We let the UAV fly in the 3D space and hover at various locations to scan/collect radar signals. Then the UAV can obtain a collection of heatmaps or energy intensity maps of the space after Fast Fourier transform (FFT) processing of the received raw radar signals. We take the 3D heatmaps as input to a deep learning model to generate the smooth and dense point clouds of the multiple objects in the space.

We investigate two different deep learning models for point clouds generation. (1) Model 1 is our recently proposed 3DRIMR/R2P model [SZH]. It consists of two stages. In Stage 1, it generates the 2D depth images of the space based radar signals. In Stage 2, it generates 3D point clouds of those objects based on their depth images obtained from Stage 1. Model 1 is used to reconstruct single objects in our previous work [SZH]. (2) Model 2 is formed by adding an image segmentation stage between Stage 1 and Stage 2 of Model 1.

The segmentation stage separates the objects in the depth image representation of a scene (result of Stage 1), so that they can be reconstructed separately in the next stage. The reason of introducing a segmentation stage is that working on single objects separately is easier than on multiple objects together when generating final point clouds.

4.1 Related Work

Recently the application of mmWave radar sensing and imaging has been investigated in various areas [LRZ, GMJ, VKJ18, YLC, FN, XJM21, SHZ21]. Different from person identification [VKJ18, YLC] and 3D human mesh estimation [XJM21], our work aims to reconstruct detailed 3D shapes of various objects in a space. Instead of focusing on a specific object (e.g., human body [YLC, XJM21]), we would like to develop a generic system that can reconstruct the 3D shape of any object. In addition, we choose to work on the raw radar energy heatmaps of a space instead of the point clouds generated by commodity radar sensors as we find them highly sparse and missing lots of information.

Our work is also closely related to a large body of literature in 3D reconstruction in computer vision [YWW17, DRN, SGF16, QSM16, QYS17, YKH18]. In particular the design of neural networks used in our architecture is inspired by PointNet [QSM16] and PCN [YKH18]. Our work in this chapter is developed based on our recent work on 3D reconstruction of a single object from mmWave radar signals [SHZ21, SZH] as we introduced in the last chapter. In addition, this work also deals with the input radar energy heatmaps of highly noisy SAR data due to the hovering instability of a small quadcopter UAV.

4.2 Methodology

4.2.1 Challenges

According to our work described in the last chapter, Chapter 3, reconstructing an object's 3D shape in the format of point clouds with mmWave radar data is challenging, because the data is usually of low resolution, sparsity, specularity, and large noise due to multi-

path effects. Hence, reconstructing multiple objects simultaneously in a space is even more challenging.

In addition, the sensing/reconstruction system we consider only utilizes low-cost commodity mmWave radar sensors (e.g., [Texc]) which have low resolution. Therefore, in order to obtain high resolution radar scan of a space, we mount a light-weight slider mechanism with radar sensor on a UAV so that the UAV can conduct Synthetic Aperture Radar (SAR) operation by sliding the sensor horizontally and vertically while hovering in the air. Due to the hovering instability of a small UAV, the data collected is highly noisy and the heatmaps of FFT processing is quite different from those of a stable precise SAR operation. Thus we are interested in whether such a vibrating UAV SAR operation can result in any meaningful reconstruction in practice.

4.2.2 Architecture Overview

We investigate two different models to address the problem of 3D reconstruction of multiple objects based on mmWave radar data. Model 1 is the model to reconstruct the 3D shape of the single object as we introduced in the last chapter, and Model 2 is formed by adding an imaging segmentation stage into Model 1. The input to each model is a set of k mmWave radar intensity maps or heatmaps collected by an UAV's SAR operation while hovering at k different locations surrounding a scene of interest. We will discuss the design details of these two models in the following.

4.2.3 Model 1

Model 1 is 3DRIMR/R2P introduced in Chapter 3 with R2P [SZH] as its Stage 2's generator. For completeness, we show the model in Figure 4.1. It consists of two stages. In Stage 1, generator network G_{r2i} generates depth images based on the radar scans of a scene from multiple view points. Those depth images can be directly converted to a rough point cloud that containing m objects. In Stage 2, the rough point cloud is processed by another generator network G_{p2p} to produce final dense and smooth point clouds of the those m objects in the scene.

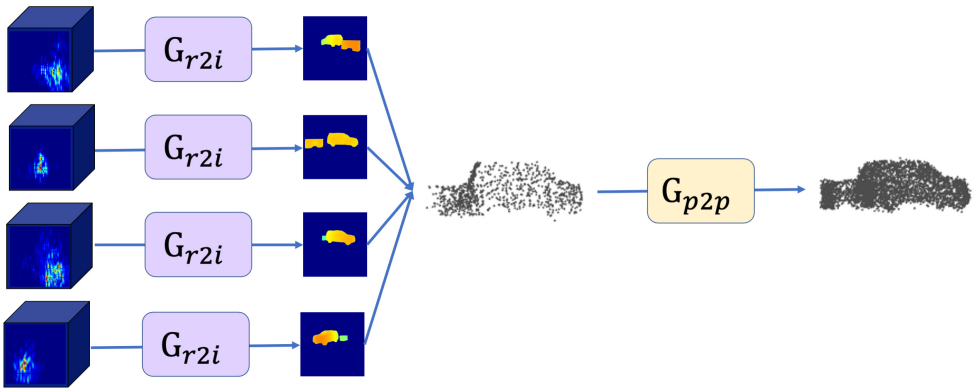


Figure 4.1: Processing pipeline of Model 1.

4.2.4 Model 2

We add an imaging segmentation stage into Model 1 to form Model 2. The idea is to separate the objects in the depth images (obtained from Stage 1), and then feed each individual object's depth image into the next stage to produce their corresponding final 3D point cloud, and finally combine them together. This model's processing pipeline is illustrated in Figure 4.2. Conditional GAN architecture is also used to train the model.

1. Stage 1: Each one of k mmWave radar intensity map captured from k view points is fed into generator G_{r2i}, which can output the corresponding depth image from each viewpoint.

2. Stage 2: An image segmentation network (e.g., Pix2Pix [IZZ17]) takes each generated depth image as input, and then does semantic segmentation of each pixel on the image. Based on the output annotation of each pixel, it can separate a depth image with multiple objects into m depth images, each containing only one object.

3. Stage 3: For each object, the model projects its corresponding k depth images from k viewpoints into a 3D space to form a coarse point cloud of the object. Then for each object, a G_{p2p} network takes its coarse point cloud as input, and outputs an accurate, complete and smooth point cloud of this object. Finally, the model combines all

point clouds of those m objects in the scene to get the final reconstructed point cloud containing multiple objects.

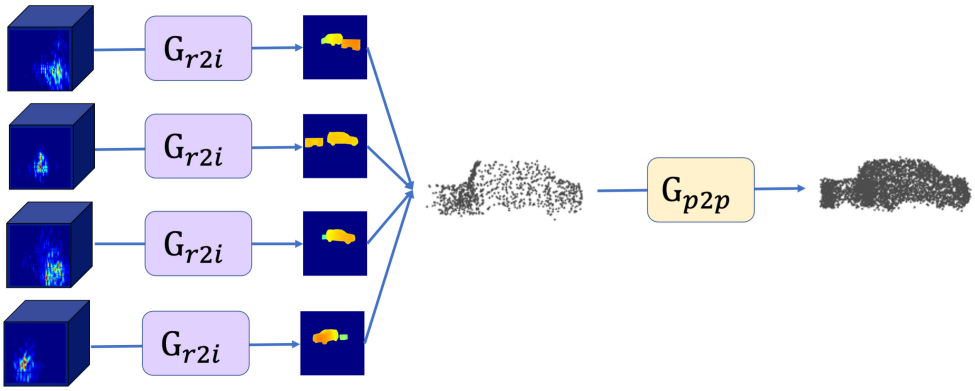


Figure 4.2: Processing pipeline of Model 2.

4.2.5 Image Segmentation Network of Model 2

The segmentation stage of Model 2 is to label every pixel of a depth image as either a target object or background. We use the popular Pix2Pix [IZZ17] network, which is a image-to-image translation conditional generative adversarial network (cGAN), in the segmentation stage. The generator is a 2D encoder-decoder convolutional neural network (CNN), which can map a grayscale depth image into its annotated image. The discriminator is a simple 2D CNN, which can output a score to indicate whether the generator's output is good or not.

Remarks. Since the model we introduce in the last chapter can reconstruct a single object well, we would like to explore its effectiveness of reconstructing multiple objects, thus we investigate Model 1. On the other hand, since image segmentation may be able to separate multiple objects from each other in a scene and hopefully reconstructing single objects might be easier, we come up with the design of Model 2. However, introducing the segmentation stage will inevitably add more uncertain errors in the formation of those coarse point clouds, and then those additional errors may make G_{p2p} not be able to extract

useful features and fail to reconstruct the 3D shapes of objects. For example, if some pixels of a small-sized object are annotated as background pixels during the segmentation stage, then the generated coarse point cloud may totally lose the shape characteristics of that object.

4.3 Implementation and Experiments

4.3.1 Datasets

Our experiments are mainly based on synthetic datasets, as collecting data via our current UAV platform is very time consuming. We first use OptiTrack [Opt] system to measure the deviation distances of a hovering UAV along x-, y-, and z-axis from its intended stable hovering position for SAR operation (scanning a space). Then based on the collected statistics we add noise (caused by hovering vibration) into the synthetic data generating process [GMJ, SHZ21]. We next show via an example that the radar data collected by a vibrating UAV's SAR operation visually looks quite different from the data of a normal stable SAR operation. Figure 4.3 shows a scene where a car and a desk are placed, and Figure 4.4 shows the analysis of the radar data of the scene. As shown in Figure 4.4, a normal SAR operation gives cleaner and more distinctly clustered radar energy maps and FFT heatmaps (Figure 4.4 (a) and (b)) when compared with the maps obtained from a vibrating UAV's SAR operation. Our experiments reported in this chapter are all based on the vibrating UAV's SAR data.

In addition, we follow a procedure that is similar to our previous work as described in Chapter 3 to generate ground truth depth images and point clouds. To generate the ground truth annotated images that are used to train the segmentation network, we manually set the background pixels to *black* color, and the pixels of the same object to the same RGB color (except black color).

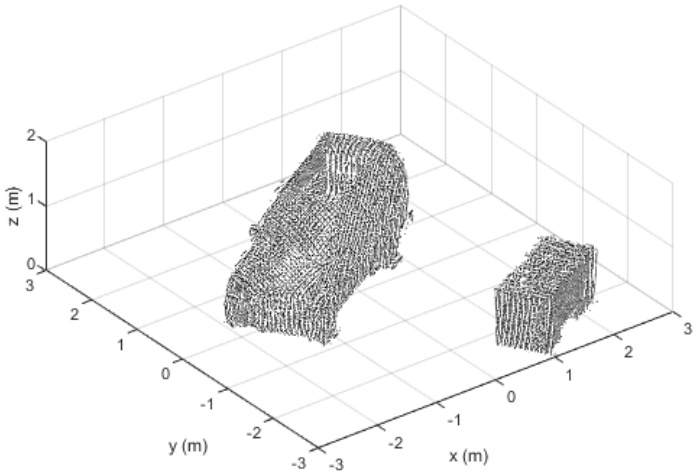


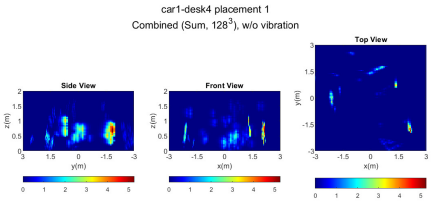
Figure 4.3: Example scene of two objects.

4.3.2 Model Training and Testing

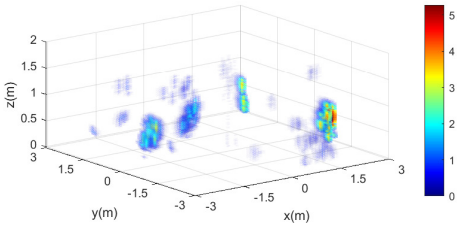
The training and testing processes of G_{r2i} and G_{p2p} in Model 1 are the same as those decribed in Chapter 3 except that in this chapter, both the training and testing data are point clouds of multiple objects rather than individual ones. Similarly, the training processes of G_{r2i} and G_{p2p} in Model 2 are the same as those in [SZH]. As for the segmentation stage of Model 2, we use 2400×4 views, totally 9600 pairs of images to train G_{seg} for 200 epochs, and the learning rate is 2×10^{-4} for the first 100 epochs and linearly reduced to 0 for the rest 100 epochs.

4.3.3 Evaluation Results

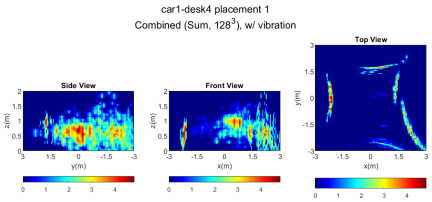
In our experiments, G_{p2p} takes an input point cloud with 1024 points and generates an output point cloud with 4096 points. If we combine m objects' generated point clouds of Model 2, the final output point clouds will contain $4096 \times m$ points, which is m times denser than the output point clouds of Model 1. For the sake of fairness, when comparing the Chamfer Distance (CD) and Earth Mover's Distance (EMD) of both models, we randomly sample 4096 points from the output point clouds of Model 2 so that the output point clouds



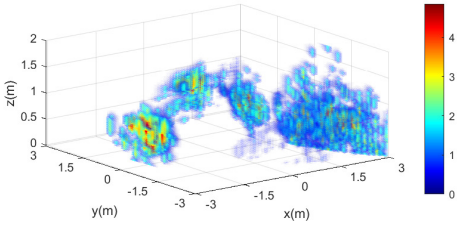
(a) Radar energy from normal SAR operation.



(b) FFT heatmap from normal SAR.



(c) Radar energy from vibrating UAV's SAR.



(d) FFT heatmap from vibrating UAV's SAR.

Figure 4.4: Comparison between normal SAR and the SAR of vibrating UAV.

using these two methods have the same number of points. Then we calculate the two models' CD/EMD with the ground truth point clouds containing 4096 points respectively.

Figure 3.10 shows the visual comparison of the two models in reconstructing two objects. We see that both models can give reasonably good reconstruction performance visually. However, there might be extra points between two objects in the point clouds generated by Model 1. This is due to the fact that Model 1 attempts to reconstruct multiple objects together. We also see that the point clouds generated by Model 2 are smoother and denser. We have similar observations of the experiments with three objects. Figure 3.10 also shows that both models are fairly robust to the highly distorted/noisy SAR data (Figure 4.4) caused by unstable UAV hovering.

Table 4.1 shows the test results of both Model 1 and Model 2 on two different scenes: a scene containing 2 objects (a car and a desk) and a scene containing 3 objects (a car, a desk and a robot arm). The loss type CD or EMD in the parenthesis of method names indicates the distance functions used in training. For example, Model 1 (CD) means the Model 1 is used and CD is used as the distance functions during training. We can see that the two models' test results of CDs are almost the same for the two scenes. However, their

Scene	Method	CD		EMD	
		avg.	std.	avg.	std.
2 objects	Model 1 (CD)	0.2	0.06	2.74	0.8
	Model 2 (CD)	0.21	0.06	4.74	1.04
	Model 1 (EMD)	0.22	0.05	0.57	0.32
	Model 2 (EMD)	0.25	0.07	3.79	0.87
3 objects	Model 1 (CD)	0.36	0.12	4.58	0.63
	Model 2 (CD)	0.3	0.08	5.81	1.28
	Model 1 (EMD)	0.31	0.08	0.74	0.28
	Model 2 (EMD)	0.33	0.1	4.46	1.15

Table 4.1: Quantitative results under different settings.

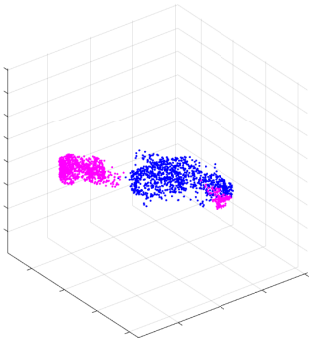
performances are different in terms of EMD. For the 2-object scene, if the loss functions used in training are CDs, then the average test result EMD between generated point clouds and ground truth point clouds of Model 2 is around 1.7 times of Model 1. If the loss functions in training are EMDs, then the average test result EMD of Model 2 is more than 5.6 times larger than Model 1. The scene with 3 objects also shows the similar results.

The superior performance of Model 1 over Model 2 in terms of CD/EMD can be explained as follows. During the training of G_{p2p} in Model 1, the goal is to reduce the losses (CD/EMD) of all m objects as a whole between the generated point clouds and the ground truth point clouds. On the other hand, the G_{p2p} of each object in Model 2 is trained separately, so the network is only trained to reduce the losses (CD/EMD) of a single object. Therefore, Model 1 will have much lower overall losses than Model 2 during testing or inference. Furthermore, we notice that the segmentation network in Model 2 can introduce additional reconstruction errors. One example is shown in Figure 4.5. Due to poor performance of G_{seg}, the projected point cloud of the m objects will contain lots of errors (Figure 4.5(a)), which further causes G_{p2p}'s failure. As shown in Figure 4.5(c)-(d), the object with too many wrong points (desk in this example) is totally missing in the output point clouds, and the other object is wrongly reconstructed (e.g., the orientation of the reconstructed car is inversed).

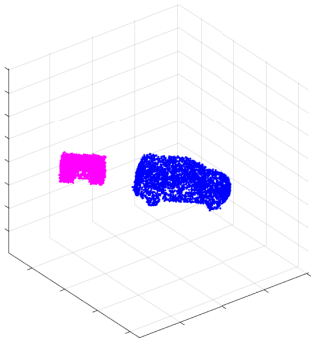
However, Model 2 can generate denser point clouds than Model 1. Besides, since Model 2 reconstructs each object separately, there will be no extra points between objects in the final point cloud, a problem associated with Model 1. For example, as we can see in Figure 3.10, the output point clouds of Model 1 (EMD) have some extra points between two objects.

4.4 Conclusion

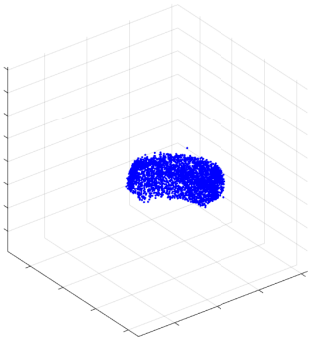
In this chapter, we have shown via an exploratory study that it is feasible to utilize the mmWave radar data collected by a vibrating small UAV's unstable SAR operation to reconstruct 3D shapes of multiple objects in a space. We have studied two deep neural network



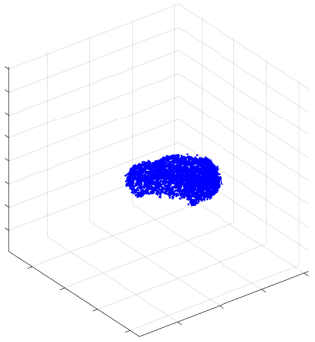
(a) Input/Coarse Point Cloud.



(b) Ground Truth Point Cloud.



(c) Output Point Cloud of Model 2 (CF).



(d) Output Point Cloud of Model 2 (EMD).

Figure 4.5: A failure example of Model 2.

models, and our experiments results have shown that they achieve promising results and they are robust to the vibrating UAV's SAR operation.

CHAPTER 5

CONCLUSION AND FUTURE WORK

5.1 Overall Conclusion

In this dissertation, we have presented our design framework of UAV-based Environment Sensing, Localization and Imaging System, which includes LIDAUS, an IoT sensing and localization subsystem, and 3DRIMR/R2P, a 3D reconstruction subsystem based on mmWave radar/Radar to Point Cloud.

With LIDAUS, a UAV navigates and explores in an unknown, challenging, and dynamic changing environment where GPS is not available or a space where it is inaccessible for humans, and thus can do self-localization and also localize wireless IoT devices through dynamic self-deployed anchors and novel algorithms (such as a weighted entropy-based clusteringof RSSI observation locations, 3D U-SLAM and its selective replay with dynamic deployed anchors, and path planning based on edge covering Eulerian cycles and Steiner tree route for cost minimization). LIDAUS is based on RSSI data only and can be easily deployed without any customized signal processing hardware and without requiring any fingerprinting or pre-trained model.

Then we have discussed 3DRIMR/R2P, a 3D reconstruction and imaging subsystem based on a conditional GAN deep learning architecture, which can reconstruct single 3D objects in the format of smooth and dense 3D point clouds from sparse and highly noisy mmWave radar signals of a commodity mmWave sensor. This architecture takes advantage of 3D convolutional operation and point cloud's efficiency of representing 3D shapes. We have demonstrated with extensive experiments that our method outperforms existing methods such as HawkEye, PointNet/PointNet++ and PCN. We also show the limitations of Chamfer Distance (CD) and Earth Mover's Distance (EMD), the two state of art point clouds evaluation metrics, in the evaluation of the shape similarity of two point clouds.

Furthermore, we have explored two different models to reconstruct 3D shapes of multiple objects in a space, and show that it is feasible to utilize the mmWave radar data collected by a vibrating small UAV's unstable SAR operation to reconstruct 3D shapes of multiple objects in a space. Our experiments results show that both two models achieve promising results and they are robust to the vibrating UAV's SAR operation.

5.2 Future Work

In the future, I will continue my research to further improve the two subsystems.

1. *Sensing and Localization subsystem.* We will further improve the software and hardware design and conduct large scale experiments in consideration of the limited battery capacity of UAVs to achieve high energy efficiency. We will also extend the system to include a swarm of UAVs with support from edge computing.

2. *3D Reconstruction and Imaging subsystem.* We will further develop novel modules into the architectures of these deep neural networks (both generators and discriminators), and also conduct large scale experiments on more practical real world environments and with more object categories to further improve our design.

REFERENCE LIST

[AAW] Roshan Ayyalasomayajula, Aditya Arun, Chenfeng Wu, Sanatan Sharma, Abhishek Rajkumar Sethi, Deepak Vasisht, and Dinesh Bharadia. "Deep learning based wireless localization for indoor navigation." In *ACM MobiCom 2020*.

[ABF] Marco Altini, Davide Brunelli, Elisabetta Farella, and Luca Benini. "Bluetooth indoor localization with multiple neural networks." In *IEEE 5th International Symposium on Wireless Pervasive Computing 2010*.

[ALI] ALIEXPRESS. `www.aliexpress.com`.

[AQ20] Zhenlin An and Lei Yang Qiongzheng Lin, Ping Li. "General-purpose deep tracking platform across protocols for the internet of things." In *ACM MobiSys*, 2020.

[AVB18] Roshan Ayyalasomayajula, Deepak Vasisht, and Dinesh Bharadia. "BLoc: CSI-based accurate localization for BLE tags." In *ACM CoNEXT*, 2018.

[Bit20] Bitcraze. *Crazyflie*, 01/05/2020.

[BKC17] Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla. "SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation." *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(12):2481–2495, 2017.

[BVH16] Wouter Bulten, Anne C Van Rossum, and Willem FG Haselager. "Human SLAM, indoor localisation of devices and users." In *2016 IEEE First International Conference on Internet-of-Things Design and Implementation (IoTDI)*, pp. 211–222. IEEE, 2016.

[DRN] Angela Dai, Charles Ruizhongtai Qi, and Matthias Nießner. "Shape completion using 3d-encoder-predictor cnns and shape synthesis." In *IEEE CVPR 2017*.

[EH05] Brian S Everitt and David C Howell. "Least Squares Estimation." *Encyclopedia of Statistics in Behavioral Science*, 2:1041–1045, 2005.

[FDU12] Sanja Fidler, Sven Dickinson, and Raquel Urtasun. "3d object detection and viewpoint estimation with a deformable 3d cuboid model." *Advances in neural information processing systems*, 2012.

[FN] Shiwei Fang and Shahriar Nirjon. "SuperRF: Enhanced 3D RF Representation Using Stationary Low-Cost mmWave Radar." In *Proc. of 2020 Intl Conf on Embedded Wireless Systems and Networks*.

[FSG] Haoqiang Fan, Hao Su, and Leonidas J Guibas. "A point set generation network for 3d object reconstruction from a single image." In *IEEE CVPR 2017*.

[GHD16] Mohammad Tayeb Ghasr, Matthew J Horst, Matthew R Dvorsky, and Reza Zoughi. "Wideband microwave camera for real-time 3-D imaging." *IEEE Transactions on Antennas and Propagation*, 65(1):258–268, 2016.

[GMJ] Junfeng Guan, Sohrab Madani, Suraj Jog, Saurabh Gupta, and Haitham Hassanieh. "Through Fog High-Resolution Imaging Using Millimeter Wave Radar." In *IEEE CVPR 2020*.

[Gra17] Inc. Grand View Research. *Bluetooth Beacon Market Worth 58.7 Billion by 2025*, 2017.

[HR92] Frank K Hwang and Dana S Richards. "Steiner tree problems." *Networks*, 22(1):55–89, 1992.

[IZZ17] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A Efros. "Image-to-image translation with conditional adversarial networks." In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1125–1134, 2017.

[JGZ17] Mengqi Ji, Juergen Gall, Haitian Zheng, Yebin Liu, and Lu Fang. "Surfacenet: An end-to-end 3d neural network for multiview stereopsis." In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 2307–2315, 2017.

[KHM17] Abhishek Kar, Christian Häne, and Jitendra Malik. "Learning a multi-view stereo machine." In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pp. 364–375, 2017.

[KJB15] Manikanta Kotaru, Kiran Joshi, Dinesh Bharadia, and Sachin Katti. "Spotfi: Decimeter level localization using wifi." In *ACM Sigcomm*, 2015.

[KLL] Chen Kong, Chen-Hsuan Lin, and Simon Lucey. "Using locally corresponding cad models for dense 3d reconstructions from a single image." In *IEEE CVPR 2017*.

[Kol09] Vladimir Kolmogorov. "Blossom V: a new implementation of a minimum cost perfect matching algorithm." *Mathematical Programming Computation*, 1(1):43–67, 2009.

[KPS14] C Praneeth Kumar, Ravi Poovaiah, Ajanta Sen, and Priya Ganadas. "Single access point based indoor localization technique for augmented reality gaming for children." In *Proceedings of the 2014 IEEE Students' Technology Symposium*, pp. 229–232. IEEE, 2014.

[LRZ] Chris Xiaoxuan Lu, Stefano Rosa, Peijun Zhao, Bing Wang, Changhao Chen, John A Stankovic, Niki Trigoni, and Andrew Markham. "See through smoke: robust indoor mapping with low-cost mmWave radar." In *ACM MobiSys 2020*.

[LVN18] Duc V Le, Wouter AP Van Kleunen, Thuong Nguyen, Nirvana Meratnia, and Paul JM Havinga. "Sombe: Self-organizing map for unstructured and non-coordinated ibeacon constellations." In *IEEE PerCom*, 2018.

[LZP11] Steven Lanzisera, David Zats, and Kristofer SJ Pister. "Radio frequency time-of-flight distance measurement for low-cost wireless sensor localization." *IEEE Sensors Journal*, 11(3):837–845, 2011.

[MBT17] Naser Hossein Motlagh, Miloud Bagaa, and Tarik Taleb. "UAV-based IoT platform: A crowd surveillance use case." *IEEE Communications Magazine*, 55(2):128–134, 2017.

[MMA14] Babak Mamandipoor, Greg Malysa, Amin Arbabian, Upamanyu Madhow, and Karam Noujeim. "60 ghz synthetic aperture radar for short-range imaging: Theory and experiments." In *2014 48th Asilomar Conference on Signals, Systems and Computers*, pp. 553–558. IEEE, 2014.

[Mon03] Michael Montemerlo. *FastSLAM: A Factored Solution to the Simultaneous Localization and Mapping Problem with Unknown Data Association*. "Carnegie Mellon University, PhD Thesis", 2003.

[Opt] OptiTrack. "OptiTrack Motion Capture Systems." `https://optitrack.com`.

[PBS] Sixu Piao, Zhongjie Ba, Lu Su, Dimitrios Koutsonikolas, Shi Li, and Kui Ren. "Automating csi measurement with uavs: from problem formulation to energy-optimal solution." In *IEEE INFOCOM 2019*.

[PUS] Despoina Paschalidou, Osman Ulusoy, Carolin Schmitt, Luc Van Gool, and Andreas Geiger. "Raynet: Learning volumetric 3d reconstruction with ray potentials." In *IEEE CVPR 2018*.

[QSM16] Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. "PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation." *arXiv preprint arXiv:1612.00593*, 2016.

[QYS17] Charles R Qi, Li Yi, Hao Su, and Leonidas J Guibas. "Pointnet++: Deep hierarchical feature learning on point sets in a metric space." *arXiv preprint arXiv:1706.02413*, 2017.

[SGF16] Abhishek Sharma, Oliver Grau, and Mario Fritz. "Vconv-dae: Deep volumetric shape learning without object labels." In *European Conference on Computer Vision*, pp. 236–250. Springer, 2016.

[SHZ15] Yuanchao Shu, Yinghua Huang, Jiaqi Zhang, Philippe Coué, Peng Cheng, Jiming Chen, and Kang G Shin. "Gradient-based fingerprinting for indoor localization and tracking." *IEEE Trans. on Industrial Electronics*, 63(4), 2015.

[SHZ21] Yue Sun, Zhuoming Huang, Honggang Zhang, Zhi Cao, and Deqiang Xu. "3DRIMR: 3D Reconstruction and Imaging via mmWave Radar based on Deep Learning." In *IEEE IPCCC*, 2021.

[SKW] Elahe Soltanaghaei, Avinash Kalyanaraman, and Kamin Whitehouse. "Multipath triangulation: Decimeter-level wifi localization and orientation with a single unaided receiver." In *ACM MobiSys 2018*.

[SM17] Edward J Smith and David Meger. "Improved adversarial systems for 3d object generation and reconstruction." In *Conference on Robot Learning*, pp. 87–96. PMLR, 2017.

[SM18] Engineering National Academies of Sciences, Medicine, et al. *Airport Passenger Screening Using Millimeter Wave Machines: Compliance with Guidelines*. National Academies Press, 2018.

[SMH07] David M Sheen, Douglas L McMakin, and Thomas E Hall. "Near field imaging at microwave and millimeter wave frequencies." In *2007 IEEE/MTT-S International Microwave Symposium*, pp. 1693–1696. IEEE, 2007.

[SR92] Scott Y Seidel and Theodore S Rappaport. "914 MHz path loss prediction models for indoor wireless communications in multifloored buildings." *IEEE Trans. on Antennas and Propagation*, 40(2), 1992.

[SRC12] Souvik Sen, Božidar Radunovic, Romit Roy Choudhury, and Tom Minka. "You are facing the Mona Lisa: Spot localization using PHY layer information." In *ACM MobiSys*, 2012.

[STE] STEREOLABS. "ZED mini Datasheet 2019 rev1." `https://cdn.stereolabs.com/assets/datasheets/zed-mini-camera-datasheet.pdf`.

[SXH] Yue Sun, Deqiang Xu, Zhuoming Huang, Honggang Zhang, and Xiaohui Liang. "LIDAUS: Localization of IoT Device via Anchor UAV SLAM." In *IEEE IPCCC 2020*.

[SXZ] Yue Sun, Zhuoming Xu, Honggang Zhang, and Xiaohui Liang. "3D Reconstruction of Multiple Objects by mmWave Radar on UAV." In *IEEE MASS 2022*.

[SZ15] Karen Simonyan and Andrew Zisserman. "Very Deep Convolutional Networks for Large-Scale Image Recognition." 2015.

[SZH] Yue Sun, Honggang Zhang, Zhuoming Huang, and Benyuan Liu. "R2P: A Deep Learning Model from mmWave Radar to Point Cloud." In *The 31st International Conference on Artificial Neural Networks (ICANN 2022), Bristol (UK), 6-9 September 2022*.

[Texa] Texas-Instruments. "DCA1000EVM." `https://www.ti.com/tool/DCA1000EVM`.

[Texb] Texas-Instruments. "Introduction to mmWave radar sensing: FMCW radars." `https://training.ti.com/node/1139153`.

[Texc] Texas-Instruments. "IWR6843ISK, 2021." `https://www.ti.com/tool/IWR6843ISK`.

[Tex20] Texas-Instruments. *mmWave sensors*, 2020.

[VKJ18] Baptist Vandersmissen, Nicolas Knudde, Azarakhsh Jalalvand, Ivo Couckuyt, Andre Bourdoux, Wesley De Neve, and Tom Dhaene. "Indoor person identification using a low-power FMCW radar." *IEEE Transactions on Geoscience and Remote Sensing*, 56(7):3941–3952, 2018.

[VKK16] Deepak Vasisht, Swarun Kumar, and Dina Katabi. "Decimeter-level localization with a single WiFi access point." In *13th USENIX NSDI*, 2016.

[VW07] Vijayanth Vivekanandan and Vincent Wong. "Concentric Anchor Beacon Localization Algorithm for Wireless Sensor Networks." *IEEE TRANSACTIONS ON VEHICULAR TECHNOLOGY*, 56, SEPTEMBER 2007.

[WL98] Jay Werb and Colin Lanzl. "Designing a positioning system for finding things and people indoors." *IEEE spectrum*, 35(9):71–78, 1998.

[WXY12] Kaishun Wu, Jiang Xiao, Youwen Yi, Min Gao, and Lionel M Ni. "FILA: Fine-grained indoor localization." In *IEEE INFOCOM*, 2012.

[WZL18] Nanyang Wang, Yinda Zhang, Zhuwen Li, Yanwei Fu, Wei Liu, and Yu-Gang Jiang. "Pixel2mesh: Generating 3d mesh models from single rgb images." In *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 52–67, 2018.

[WZX16] Jiajun Wu, Chengkai Zhang, Tianfan Xue, William T Freeman, and Joshua B Tenenbaum. "Learning a probabilistic latent space of object shapes via 3d generative-adversarial modeling." *arXiv preprint arXiv:1610.07584*, 2016.

[XJM21] Hongfei Xue, Yan Ju, Chenglin Miao, Yijiang Wang, Shiyang Wang, Aidong Zhang, and Lu Su. "mmMesh: Towards 3D real-time dynamic human mesh construction using millimeter-wave." In *ACM MobiSys*, 2021.

[XJS14] Jie Xiong, Kyle Jamieson, and Karthikeyan Sundaresan. "Synchronicity: pushing the envelope of fine-grained localization with distributed mimo." In *Proceedings of the 1st ACM workshop on Hot topics in wireless*, 2014.

[YKH18] Wentao Yuan, Tejas Khot, David Held, Christoph Mertz, and Martial Hebert. "PCN: Point completion network." In *2018 International Conference on 3D Vision (3DV)*, pp. 728–737. IEEE, 2018.

[YLC] Xin Yang, Jian Liu, Yingying Chen, Xiaonan Guo, and Yucheng Xie. "MU-ID: Multi-user Identification Through Gaits Using Millimeter Wave Radios." In *IEEE INFOCOM 2020*.

[YWW17] Bo Yang, Hongkai Wen, Sen Wang, Ronald Clark, Andrew Markham, and Niki Trigoni. "3d object reconstruction from a single depth view with adversarial learning." In *IEEE International Conference on Computer Vision Workshops*, 2017.

[YYR06] Moustafa Youssef, Adel Youssef, Chuck Rieger, Udaya Shankar, and Ashok Agrawala. “Pinpoint: An asynchronous time-based location determination system.” In *ACM MobiSys*, 2006.

[YZL13] Zheng Yang, Zimu Zhou, and Yunhao Liu. “From RSSI to CSI: Indoor localization via channel response.” *ACM Computing Surveys*, 46(2), 2013.