國 立 交 通 大 學

電控工程研究所

碩 士 論 文

高擬真度之工件隨機堆疊取放演算法的模擬與驗證平台

A High Fidelity Simulator for Random Bin Picking Scenario Evaluation

研 究 生：莊　凱　傑

指導教授：胡　竹　生 教授

中 華 民 國 一〇三 年 七 月

高擬真度之工件隨機堆疊取放演算法的

模擬與驗證平台

A High Fidelity Simulator for

Random Bin Picking Scenario Evaluation

研 究 生：莊 凱 傑　　Student : Kai-Chieh Chuang

指導教授：胡 竹 生 教授　　Advisor : Dr. Jwu-Sheng Hu

國 立 交 通 大 學
電控工程研究所
碩 士 論 文

A Thesis
Submitted to Institute of Electrical Control Engineering
College of Electrical and Computer Engineering
National Chiao Tung University
in Partial Fulfillment of the Requirements
for the Degree of
Master
in
Electrical Control Engineering

July 2014

Hsinchu, Taiwan, Republic of China

中華民國 一〇三 年 七 月

高擬真度之工件隨機堆疊取放演算法的模擬與驗證平台

研究生：莊　凱　傑　　指導教授：胡 竹 生 教授

國立交通大學電控工程研究所

摘要

本論文提出了一個高擬真度模擬與驗證平台。此平台目前適用於工件隨機堆疊取放演算法（random bin picking）的模擬與驗證。此平台能自動產生大量的演算法實驗數據，並讓使用者能夠藉由這些數據，了解其演算法的特性並做出改善。因為此平台擁有逼真的模擬環境，使得模擬得到的結果能與真實環境得到的結果相近。

本篇論文專注於建立此模擬與驗證平台和建立高擬真度環境的方法。首先，此平台能讓使用者指定模型，並透過物理引擎模擬該模型的多物體隨機堆疊情形。接下來，藉由此平台的虛擬深度感測器擷取該場景，並將逼真的擷取資料傳送給姿態估測演算法。經過姿態估測演算法對收到的資料處理後，將演算法結果傳送回此平台。最後，此平台利用存在於模擬世界中的正確資料(ground truth)，驗證演算法結果的準確度。藉由不斷的模擬與驗證，此平台能提供大量的準確演算法統計特性。

本篇論文將提供讀者此平台的建立方法，以及如何改進原來既有的模擬平台來達大逼真的模擬結果。並在最後，利用此平台來驗證工業技術研究院的姿態估測演算法。並透過此平台未姿態估測演算法找到最佳的參數。

關鍵詞：深度感測器模擬、雜訊模擬、隨機堆疊取放演算法、姿態估測演算法、分割演算法、模擬驗證平台。

A High Fidelity Simulator for

Random Bin Picking Scenario Evaluation

Student : Kai-Chieh Chuang Advisor : Dr. Jwu-Sheng Hu

Institute of Electrical Control Engineering
National Chiao Tung University

ABSTRACT

We present a realistic evaluation platform for a random bin picking scenario on a simulator, targeted to evaluate the performance of an algorithm, for instance, pose estimation, segmentation, etc. It's a fully automated process. First it will simulate objects stacking randomly in a box. Secondly, a highly realistic virtual depth sensor will capture the scene and send the data to the targeted algorithm. Then, our evaluation platform waits for the algorithm result and evaluates the algorithm by comparing the result with ground truth. By running this process continuously, we can obtain a vast amount of valuable statistics data about the performance of the algorithm. In this thesis, we will mention the architecture of our evaluation platform, and focus on how we try to model an active stereo depth sensor by simulating the real sensor's attributes and noise, making it as realistic as possible.

Keywords: Virtual RGB depth sensor, Synthetic noise, Random bin picking, Pose estimation, Segmentation, Simulation, Evaluation.

誌 謝

碩士學業的完成，最感謝指導教授胡竹生老師的教導。雖然只有兩年的修業，透過老師的細心教導，在研究與學業都有所進步。在碩士期間，老師引薦我至工業技術研究院工讀，讓我學到了許多工業自動化的專業知識，對我的碩士學業有很大的幫助。在此獻上對老師的最高謝意。

感謝我的家人的支持，讓我能無後顧之憂的修習學業。也要感謝鄂雅芳陪伴我，給予我鼓勵與動力，使得我碩士期間，有很好的學習成果。

特別感謝工研院吳晉嘉博士，同時也是交大的學長。在工研院帶領我研究，並且教導我許多工業自動化的知識、程式開發等專業知識。

最後，感謝實驗室的學長、同學與學弟妹，碩士期間有了你們才能如此順利。

莊凱傑 謹誌

2014 年 07 月

Content

摘要…… i

Abstract ……..…… ii

誌謝……..…… iii

Content ……...……. iv

List of Tables ……...……...…..... vii

List of Figures ……..……...….. viii

Chapter 1. Introduction ……... 1

1.1 Motivation and Objective ……. 1

1.2 Contributions ……... 3

1.3 Thesis Architecture …….. 4

Chapter 2. The Evaluation Platform …….. 5

2.1 Introduction to Gazebo …… 5

2.2 Configuration for Pose Estimation …….. 6

2.3 The Evaluation Process ……..…….. 8

2.3.1 Interface to Gazebo – Plugins ……... 9

2.3.2 Simulation Description Format ……... 10

2.3.3 Architecture of the Evaluation Platform …….. 11

2.3.4 Geometric Symmetry in 3D Model ……. 14

2.3.5 Success Criteria for Pose Estimation …….. 16

2.4 Features …….. 18

Chapter 3. Pose Estimation Algorithm ……. 20

3.1 The ITRI Pose Estimation Algorithm …… 20

3.1.1 Off-line Stage …….. 20

3.1.2 On-line Stage ……... 21

3.1.3 Feature Extraction and Individual Feature Saliency …… 21

Chapter 4. Physics Simulation …….. 23

4.1 Dynamics Simulation ……..... 23

4.2 Collision Detection …… 24

4.3 Random Stacking ……... 26

Chapter 5. The Virtual RGB-D Sensor …… 28

5.1 Target RGB-D Sensor …… 28

5.2 Extract Depth Image ……... 30

5.3 Simulation Methods ……... 31

5.3.1 The Occlusion Effect …….. 31

5.3.2 Intensity Saturation of Projected Light ……..… 34

5.3.3 Low Confidence Effect …….. 37

5.3.4 Noise Synthesis …….. 39

5.3.5 Blurring Effect …… 41

5.3.6 The Simulation Result ……. 42

5.3.7 The Disturbance on Threshold …… 43

Chapter 6. Performance of the Evaluation Platform ……..…… 48

6.1 Time to the Steady State ……. 48

6.2 Virtual Depth Sensor Capturing Time ……..... 49

6.3 Computation Time ……. 49

Chapter 7. Experiments ……..… 50

7.1 Ideal Segmentation ……..... 50

7.2 Evaluating Pose Estimation …….……... 51

7.2.1 Single Model Pose Estimation …… 51

7.2.2 Multiple Models Pose Estimation …… 55

7.3 Optimizing Parameters Using the Evaluation Platform ……..... 56

Chapter 8. Conclusion ……..…….. 59

References ...…… 60

List of Tables

Table 5-1 specifications of the ITRI depth sensor …….. 28

Table 6-1 Specifications of the PC used in this thesis …… 48

Table 6-2 Time to steady state of the random stacking process ……. 48

Table 6-3 Time to generate a 1280 x 960 synthetic depth image ……... 49

Table 7-1 Success rates using different search radius of the SHOT descriptor …….. 57

Table 7-2 Success rates using different similarity thresholds ……. 57

List of Figures

Figure 1-1 the concept of the evaluation platform …… 4

Figure 2-1 Architecture of Gazebo …… 5

Figure 2-2 An example of the simulation environment …… 7

Figure 2-3 The flow diagram of the evaluation process ……... 8

Figure 2-4 A simplified world SDF file …….. 10

Figure 2-5 A detail flow diagram of the evaluation platform …….. 11

Figure 2-6 Illustration of 2D rotational symmetry ……..... 14

Figure 2-7 The symmetries in a cylinder ……..... 15

Figure 2-8 A visualization of a rotation expressed in axis-angle representation …… 16

Figure 2-9 An example of an evaluation result for multiple models pose estimation ……..... 19

Figure 3-1 The flow diagram of the ITRI pose estimation algorithm …… 20

Figure 4-1 CAD models used in this thesis …… 23

Figure 4-2 An example of calculating physics properties …….. 24

Figure 4-3 An undesired result of Wrench physics simulation …….. 25

Figure 4-4 Semi-solution for penetration issues …… 25

Figure 4-5 The stacking result of Cylinder, Brazo, and Wrench ……... 27

Figure 5-1 Illustration of an ITRI depth sensor ……. 28

Figure 5-2 Images taken from the ITRI depth sensor …… 29

Figure 5-3 Model of virtual depth sensor …….. 29

Figure 5-4 An example of a perfect depth image from computer graphics ……... 31

Figure 5-5 Illustration of an occlusion phenomenon ……. 32

Figure 5-6 Illustration of the simulation method for an occlusion phenomenon …….. 33

Figure 5-7 Difference between perfect data and simulated data with an occlusion effect ... 33

Figure 5-8 An illustration of intensity saturation effects …….. 34

Figure 5-9 The different types of reflection’s influence on the depth sensor are illustrated ... 35

Figure 5-10 The simulation result of a two material object …… 36

Figure 5-11 Extraction of the specular information …… 36

Figure 5-12 Simulated data with occlusion effects and without occlusion effects ……. 37

Figure 5-13 Illustration for explaining the low confidence effect …….. 38

Figure 5-14 Simulated data with only occlusion effects and additional confidence check … 38

Figure 5-15 Noise from the real depth sensor and synthetic noise ……...….. 40

Figure 5-16 Point cloud with synthesized noise added ……... 40

Figure 5-17 Shows the blurring effect captured by the real depth sensor ……... 41

Figure 5-18 Synthetic sensor data with blurring effect ……... 41

Figure 5-19 Simulation methods applied on perfect depth data ……. 42

Figure 5-20 Point cloud captured by ITRI depth sensor in the 3D view …… 43

Figure 5-21 An example of 2 dimension coherent noise generated by Perlin Noise …….. 43

Figure 5-22 the Differences between non-coherent noise and Perlin Noise ……... 44

Figure 5-23 Perlin Noises with different grid size and its combination ……..……... 44

Figure 5-24 The 4 grid points bounding (x, y) …… 45

Figure 5-25 Vectors from the grid points to (x, y) …….. 45

Figure 5-26 Influences from the grid points ……... 46

Figure 5-27 Visualization of the weighted function …… 46

Figure 5-28 Perlin Noises with different grid size …….. 47

Figure 5-29 The combination of Perlin Noises with different grid size …….. 47

Figure 7-1 Ideal segmentation of a scene …… 50

Figure 7-2 The evaluation result of Brazo …….. 51

Figure 7-3 The evaluation result of Wrench ……... 52

Figure 7-4 The evaluation result of 9 Cylinders ……. 52

Figure 7-5 The evaluation result of 18 Cylinders ……... 53

Figure 7-6 A difference success rate on a different pose ……. 54

Figure 7-7 Evaluating multiple models pose estimation ……...…….. 55

Figure 7-8 Socket model ……. 56

Figure 7-9 The trials needed to sufficiently determine the success rate ……. 56

Figure 7-10 The scatter plot of the similarity threshold experiment ……... 57

Chapter 1. Introduction

1.1 Motivation and the Objective

The need for industrial automation has been increased significantly in recent years due to the rising cost of human labor. With the advent of image sensors and reduction of computational cost, intelligent robots that can handle randomly stacked work pieces or tools is becoming more and more feasible. The technique is referred as Random Bin Picking (RBP), where automatically identifying the pose of an object in a randomly stacked pile is the most crucial issue.

Conventional RBP approaches involve a data acquisition step using laser scanners or 2-D stereo vision, a data preprocessing step using mesh construction or plane fitting, an optional feature extraction step and a pose estimation step using Iterative Closest Point or RaNdom SAmple Consensus based matching [6] and [7]. Though different in these standard steps, most works follow the similar framework handling real data and increase pose estimation accuracy by improving data capturing, preprocessing or feature extraction processes. Very few research efforts have devoted to the prior analysis of sensor physics and how such information can help online pose estimation. To the best of our knowledge, the authors of [8], [9], and [10] are the only groups of researchers adopting a simulated database to narrow down the search space of pose estimation, aiming to remove false matches and increase estimation accuracy.

In this thesis, we develop a simulation platform for a random bin picking scenario. The idea leverages the constantly decreasing cost of computing power and the tool that can help develop simulation environments becomes mature, which makes simulating an algorithm with efficiency become feasible. Essentially, we attempt to create a simulation environment that provides a high fidelity copy of the real world environment, including random stacking simulation, a sensor model and noise synthesis.

One of the advantages of using simulation is the availability of ground truth. The problem

of the absence of ground truth in real world is common in researches, [6] in which they use the error metric of comparing distances between points on the model and their closest point in the scene .They also point out the problem themselves and suggest that a more sophisticated error metric should probably be used to qualify the resulting matches. In [7], they use the metric of using repeatability by measuring the variation between the measured pose with the average of multiple measurements. Another error metric uses the manually label ground truth, which is of course not accurate. These error metrics cannot reflect the real performance of their algorithm and place an obstacle to compare different algorithms.

Furthermore, the cost and time for experiment in real world are enormous. Most experiment results shown in the researches used few samples, which leads to insufficient performance data. And not to mentioned using the experiment result to improve their algorithm. Although [6], [7], and [8] have used the power of simulation, they did not provide a detail comparison between real sensor data and simulated sensor data, and the simulation is in lack of realistic simulation and reasonable physics. So the result generate from the simulation is doubtful.

It is becoming increasingly difficult to ignore the power of simulation. Although, the similarity rate between simulation and real world directly influences the reliability of the simulation result. So far, however, there have been little discussion in realistic simulation, or by specific, the realistic of the vision sensor. In the research of the Mitsubishi Electric Research Laboratory, [9], they use the multi-flash camera (MFC), [19], to capture the depth edges in the scene. The depth edges are used for the shape-matching algorithm called fast directional chamfer matching (FDCM). In their simulation stage, they obtained the depth-edge image by simulating the MFC imaging in OpenGL. To show the robustness of the FDCM algorithm, they removed a small fraction of the ideal depth edges. Furthermore, the depth-images were corrupted with significant noise by adding uniformly sampled line segments. Although, these synthetic noises significantly degraded the quality of the ideal depth-images, it is nowhere similar to the real depth-images provided in the paper. This induces some doubts on the

simulation results.

In [2], they analyzed the noise present in range data measured by a Konica Minolta Vivid 910 scanner, in order to better characterize real scanner noise. Their research field is in 3D surface denoising, where most researches assumed the noise to be independently distributed Gaussian. They show via measurements of an accurately machined almost planar test surface that real scanner data does not have such properties. They performed a Pearson's chi-square test on the scanner data, which rejected the hypothesis that the estimated noise distribution is Gaussian. They also found that the scanner noises have significant short range correlation. They also provided a Fourier analysis on the noise, and synthesized the noise with Fourier Transform. At last, they conducted an experiments using the realistic synthetic noise on denoising algorithm. They concluded that many previous papers claiming good denoising results based on experiments with synthetic Gaussian noise are over-optimistic in their assessment of the ability of algorithms to remove real scanner noise.

1.2 Contributions

The contributions of this thesis is that a high fidelity simulator. A virtual RGB-D sensor that mimic the real sensor, using systematic methods to simulated the observation of the real sensor data. Adjustment to the Gazebo [3], which our simulator based on, is made for more realistic physics simulation. Although the simulation is not exact copy of the real world, important features of the flow are nonetheless captured.

An automatic evaluation platform is proposed to evaluate the performance of pose estimation. However, the simulator is not only limited to pose estimation. It is designed with flexibility. The concept of the evaluation platform is illustrated in Figure 1-1.

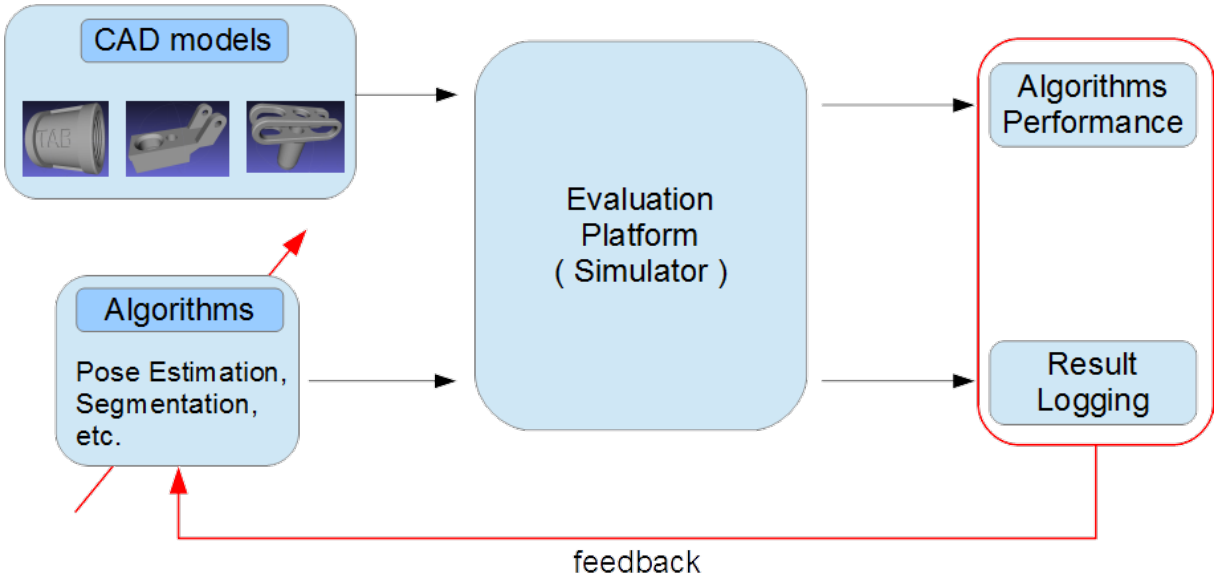

Figure 1-1. The concept of the evaluation platform. Input to the algorithm is the target algorithm and target models, then the algorithm result will be generated automatically. The evaluation results are useful, one application of which is to adapt the algorithm using the feedback from it.

Two experiments show the efficiency using our evaluation platform to evaluate the performance of the algorithm. The first experiment evaluated the performance of the pose estimation algorithm, [1], developed in Industrial Technology Research Institute of Taiwan (ITRI). From the vast amount of evaluation results, properties of the algorithm and improvement can be unveiled. Second experiment revealed the power of the power of simulator, massive amount of experiments is conducted, and the results help to improve the pose estimation by finding the optimal value of parameters.

1.3 Thesis Architecture

The evaluation platform is introduced in Chapter 2. A brief introduction of the pose estimation algorithm used to demonstrate the power of simulation is in Chapter 3. Adjustment and configuration made to the Gazebo for a better simulation result is in Chapter 4. Chapter 5 provides the simulation methods to mimic the real RGB-D sensor in virtual space, where defects and noise of the real sensor are specified and simulated. We show the efficiency of our evaluation platform in Chapter 6. Extensive experiments are conducted in Chapter 7. Finally, the conclusion is stated in Chapter 8.

Chapter 2. The Evaluation Platform

In this chapter, we will introduce the simulator for evaluating the random bin picking scenario. The architecture and features of the simulator will be explained in detail.

2.1 Introduction to Gazebo

Gazebo [3] is a 3D multi-robot simulator, which offers the ability to accurately and efficiently simulate populations of robots in complex indoor and outdoor environments. It is a robust physics engine, high-quality graphics, and convenient programmatic and graphical interfaces. The architecture of Gazebo is shown in Figure 2-1. Gazebo is divided into several libraries, which is physics library, rendering library, sensors library, transport library, and GUI library. These libraries are used by two main processes, server and client. The server process runs the physics loop and generates sensor data, while client process provides user interaction and visualization of a simulation.

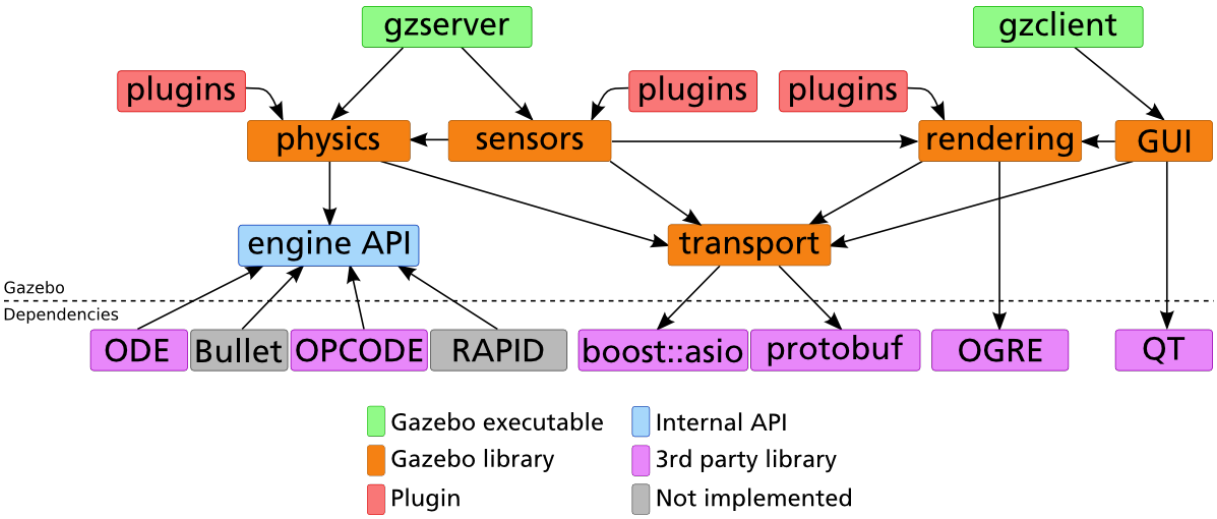

Figure 2-1. Architecture of Gazebo.

The physics library runs the physics update cycle, loads and maintains all the models and their plugins, and is capable of saving and load simulation state. Gazebo utilizes third party physics engines, such as ODE [4] or Bullet, to compute the proper dynamic and kinematic effects. The rendering library utilizes OGRE to visualize the simulation world. This library is

used by the graphical interface to allow the user to see and interact with simulation. It is also used by the sensor library to generate data for sensors like cameras. The sensors library has the responsibility of loading and updating individual sensors such as lasers, cameras, and IMUs. The transport library uses boost::asio [12] to create and maintain socket-based connections between Gazebo components. Google protobuf [13] provides the message serialization and deserialization infrastructure to pass data between components. The GUI library uses Qt [14] to allow users access to a running simulation. The GUI provides a mechanism to view, create, modify, and save simulation instances.

The user can interact with Gazebo through plugins, which provide direct access to Gazebo's API and direct control over all aspects of the simulation engine, including the physics engine, graphics libraries, and sensor generation. Another way to communicate with Gazebo is through TCP/IP, which is a socket-based message passing using Google Protobufs. It allow the separation of server and client, that is, we can run the simulator and the algorithm on separate computers, even on a different platform. The features mentioned above make Gazebo a powerful tool for research and development.

2.2 Configuration for Pose Estimation

Although the final goal for the simulator is evaluating the whole random bin picking scenario, the simulator is currently aimed for evaluating the most important part, the pose estimation algorithm. The pose estimation has the mission to estimate the pose of the objects randomly stacked in a box, given only the information captured by the RGB-D sensor. So there are three main characters in the simulation environment, the virtual RGB-D sensor, a container to hold the objects, and the target model for pose estimation to estimate. A peek to the simulation environment is shown in Figure 2-2.

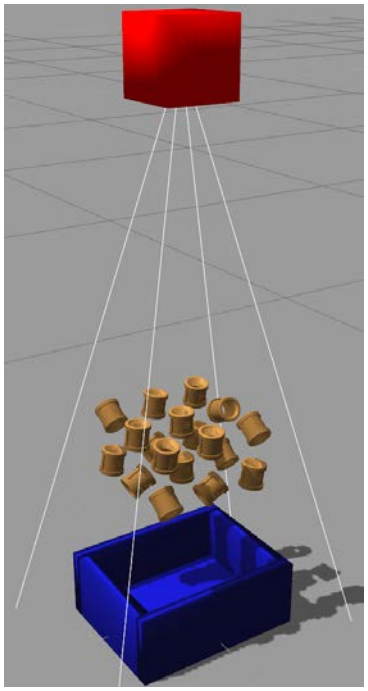

Figure 2-2. An example of the simulation environment.

The simulator can accept STL or Collada formatted mesh files, with Collada being the preferred format. As long as you have a popular format of the CAD model, it's easy to convert the format to STL or Collada. The objects are then randomly placed above the box, and the physics engine will handle the stacking simulation.

The virtual RGB-D sensor leverages the power of Ogre3D, which provides many of the advanced features seen in current commercial and open-source games. In particular, it provides support for using the vertex and pixel shader on the graphics card, which provides high-quality graphics while reducing the load on the CPU. The shader enables users to program the programmable GPU rendering pipeline. Our virtual RGB-D sensor uses the shader to extract depth data and simulates the effects of the real sensor. Besides the shader, Ogre3D also gives us easy access to special effects such as the particle system, transparency, realistic shadow effects, and realistic material properties. This is important to generate more realistic synthetic RGB images. The more realistic the sensor data are, the more likely it is that the same algorithm will work both in the simulator and in the real world. More details will be provided in Chapter

5.

A virtual RGB-D sensor as well as box and target objects, is treated as an individual model in Gazebo, so it's possible to have a multiple virtual RGB-D sensor. That is, the evaluation platform is capable of evaluating the random bin picking algorithm at the same time. The advantage for this is that we can develop a production line in the simulator first, then if the production line works well, then we start to construct the production line in the real world, but if the production line doesn't works well, then the user can find the problem in the simulator, and do the modification in it directly. This will indeed save the cost of the construction of production line.

2.3 The Evaluation Process

Figure 2-3, shows the flow diagram of the evaluation process.

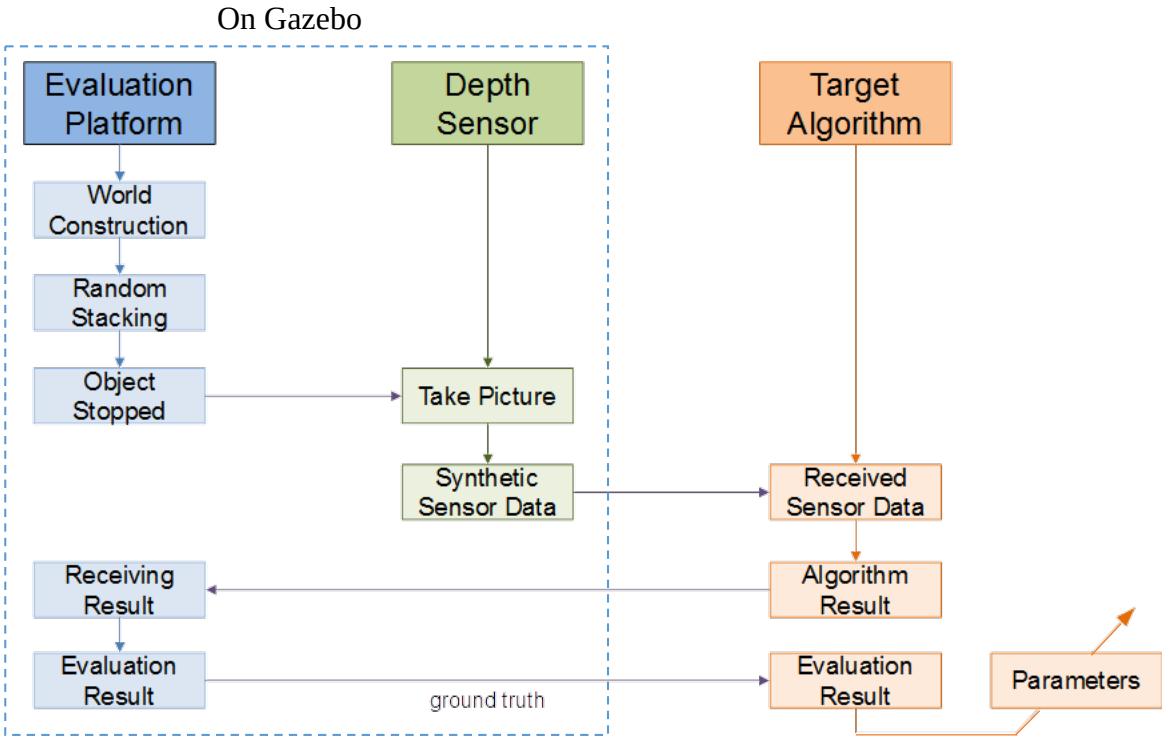

Figure 2-3. The flow diagram of the evaluation process

The evaluation process is mainly consist of the evaluation platform, the virtual RGB-D sensor, and the algorithm to be evaluate. The process begins with constructing the environment in Gazebo, including loading the target model, box, and virtual RGB-D sensor to their specified

pose, then Gazebo will simulate the random stacking of the models. After the models have converged into a steady state, the evaluation platform will ask the virtual RGB-D sensor to capture the scene, and generated a realistic synthetic sensor data that mimic the real sensor in the real world.

Next, the target algorithm will receive the synthetic sensor data through the socket-base message mechanism. Be aware that, the algorithm is not in the Gazebo nor the evaluation platform, it is an independent process either on the same computer or a separate computer. The only communication between the algorithm and the evaluation platform is through TCP/IP. After receiving the sensor data, the algorithm will start processing and a result will be sent to the evaluation platform. Then the algorithm result will be compared to the ground truth, and an evaluation result is saved locally and also given to the target algorithm. The evaluation result could be used to adjust the algorithm on the fly, or saved for later assessment by the user.

In this thesis, the target algorithm is a pose estimation algorithm, the algorithm result will be the pose of the estimated object. Evaluation platform will check it with the ground truth. If it is correct, the estimated object will be picked up from the box. The physics engine will then simulate the new stacking without the correctly estimated object. Then the evaluation process will ask the virtual sensor to capture the scene again, and so on.

2.3.1 Interface to Gazebo - Plugins

Gazebo offers several methods for interfacing with the user. One of the methods is to use plugins, which is an in-process interface method. Plugins are written in C++, compiled as shared objects, and loaded into the same address space as Gazebo. Plugins are useful because they let developers control almost any aspect of Gazebo. It is the main interface to programmatically alter the simulation. There are currently 5 types of plugins, which is World, Model, Sensor, System, and Visual. Each plugin type is managed by a different component of Gazebo. For example, a Model plugin is attached to and controls a specific model in Gazebo. Similarly, a

World plugin is attached to a world, and a Sensor plugin to a specific sensor. The System plugin is specified on the command line, and loads first during a Gazebo startup. This plugin gives the user control over the startup process.

2.3.2 Simulation Description Format

Gazebo uses XML to load and save information about a simulation world or model. They have defined their own format, called Simulation Description Format (SDF), which encapsulates all of the necessary information for a robotic simulator. These information include the description of the scene, physics, models, lights, and plugins. The SDF file for a simulation world will contain the settings of physics, the scene of the simulator, the model included, and its world plugin. The user can setup their world environment using the SDF file. The SDF file for a simulation model will contain a collection of links, collision shapes, joints, and sensors. The user can design their robot using this SDF file. A simplified SDF file describing a simulation world is given in Figure 2-4. The SDF file will be loaded into the Gazebo at start up, and specify the ODE library to be used in the simulation world. A directional light is added to the scene. A depth sensor model is included in this simulation world. Its pose, collision shape, and Gazebo sensor type are specified.

```
<sdf version="1.4">
    <world>
        <physics type="ode">
            <gravity> 0 0 -9.8 </gravity>

        <light type="directional">
            <direction> 0 0 -1 </direction>

        <model name="depth_sensor">
            <pose>
            <link>
                <collision>
                <sensor type="depth">
```

Figure 2-4. A simplified world SDF file.

2.3.3 Architecture of the Evaluation Platform

Figure 2-5, shows a detail flow diagram of the evaluation platform.

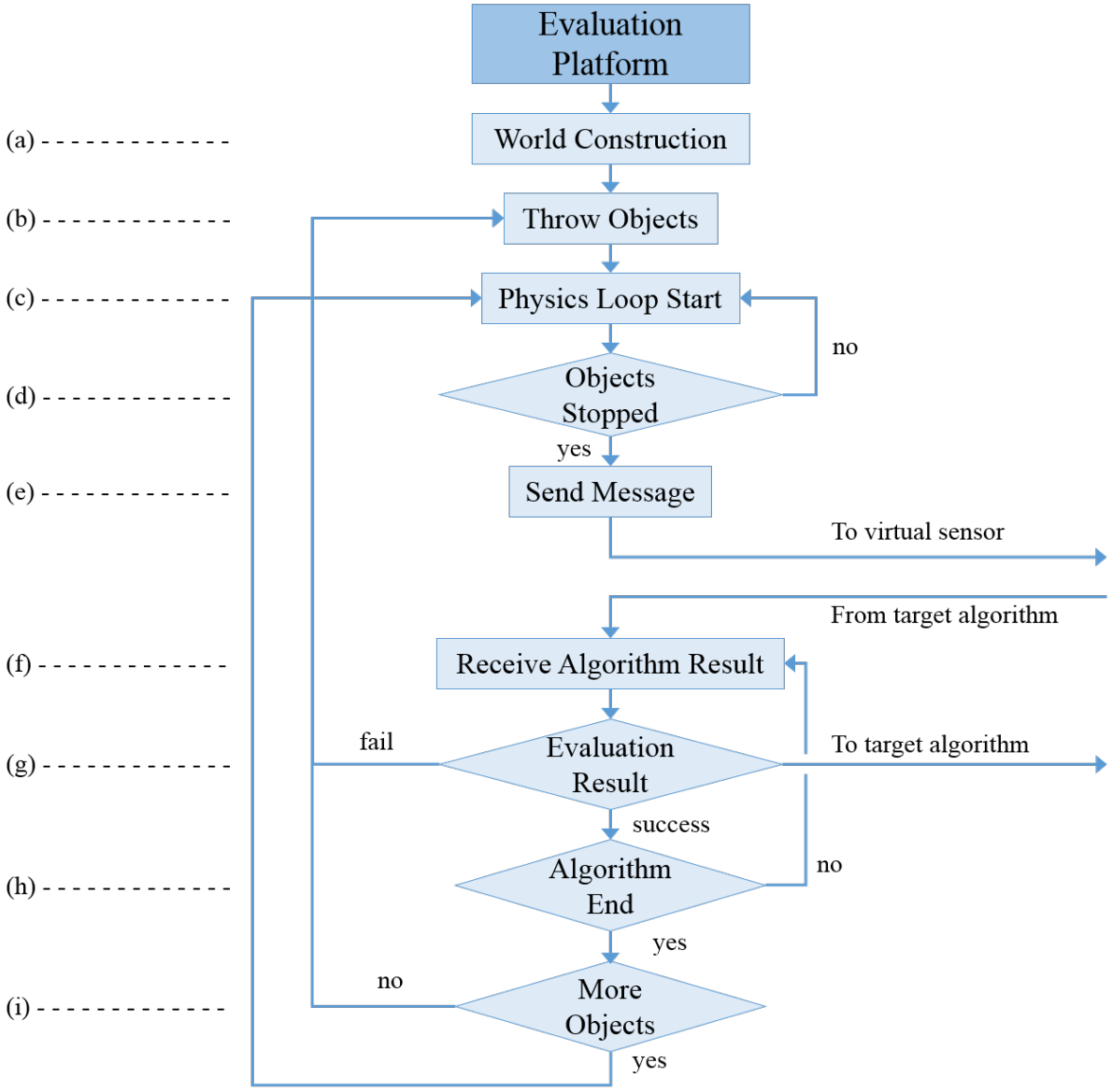

Figure 2-5. A detail flow diagram of the evaluation platform. The symbol, (a), (b)… (i), represent the corresponding blocks in the flow diagram.

The main responsibilities for the evaluation platform is illustrated in Figure 2-3, colored in blue. It is clear that the information needed covers every aspect of the Gazebo, so the evaluation platform is designed with the World plugin. This will gives the evaluation platform the full control of the whole simulator.

The first task for the evaluation platform is to construct the environment (Figure 2-5, (a)). We use SDF to describe the virtual RGB-D sensor, the box, and the model. Though, we

programmatically load the models into the simulation world using a friendly designed parameters file. This allow the user of the evaluation platform to modify the simulation world without the prerequisite of understanding SDF details. The settings of the parameters file include, the size of the box, the pose of the sensor, the specifications of the sensor, and the stacking method of the objects. The objects are placed above the box, where the positions of the objects are arranged in layers of array, and with their poses randomly generated (Figure 2-5, (b)). An instance can be seen in Figure 2-2. The user can set the width and length of the array, the number of layers, and the distance between each models.

After completing the construction of the simulation environment, the physics simulation loop will start (Figure 2-5, (c)). A callback function is connected to the events of the WorldUpdateBegin, that is, whenever the physics loop start, the callback function will be called. In the callback function, it will checks the movement of the objects every 100 millisecond (Figure 2-5, (d)). If the linear velocity of every object are within a threshold, the random stacking scene is in potential steady state. And if the random stacking scene is in potential steady state for 5 consecutive time interval, the random stacking scene is considered as in steady state. Then the objects in the box will be set to static object. The reason for designing the mechanism in this particular way is because the objects will have small jitter when colliding each other. Be aware that the values mentioned above are also adjustable using the parameters file.

A message will be sent to the virtual RGB-D sensor (Figure 2-5, (e)), and ask the sensor to capture the scene. The message is sent through a publish/subscribe mechanism provided by the Gazebo transport library. The evaluation platform will publish a topic, while the virtual sensor will subscribe to that same topic. When evaluation platform send a message using that topic, the virtual sensor can receive the message from the topic. This mechanism provided by the Gazebo helps decoupling the relationship of two instances, which is what the programmers want.

The virtual sensor will provide the synthetic data to the target algorithm. And the algorithm will sent the algorithm result back to the evaluation platform with the same mechanism mentioned above (Figure 2-5, (f)). The algorithm result for the pose estimation will be the estimated pose[1] in the camera coordinate and the model being identified. Since the ground truths are in world coordinate, a transformation must be applied to the estimated pose. The estimated pose in camera coordinate are described using 4x4 homogeneous transformation matrix, ${}^{Camera}\hat{P}$. The desired estimated pose in world coordinate, ${}^{World}\hat{P}$, can be obtained by post-multiplying the pose of the virtual sensor in world coordinate, ${}^{World}P_{Camera}$:

$$ {}^{World}\hat{P} = {}^{World}P_{Camera} \times {}^{Camera}\hat{P} \tag{1} $$

After obtaining the estimated pose in world coordinates, the ground truth will be the closest object in the simulator relative to the estimated pose. The pose of the ground truth is ${}^{World}P_{GroundTruth}$. Then the error of the estimation result ε can be obtained by:

$$ \varepsilon = {}^{GroundTruth}\hat{P} = {}^{GroundTruth}P_{World} \times {}^{World}\hat{P} = \left({}^{World}P_{GroundTruth}\right)^{-1} \times {}^{World}\hat{P} \tag{2} $$

The error of the estimation result is equivalent to the estimated pose in ground truth coordinate, ${}^{GroundTruth}\hat{P}$.

The evaluation platform will first check if the object is recognized correctly, then the errors between estimated pose and ground truth will be calculated (Figure 2-5, (g)). If the errors are also within the tolerance value, then the algorithm result is labeled as success. If not, the algorithm result will be labeled as fail. Finally, the evaluation result will be saved locally and sent to the target algorithm. An example of an evaluation result is shown in Figure 2-9.

[1] The pose is expressed using 4x4 homogeneous transformation matrix, which contain the information of orientation and translation.

If the algorithm result is incorrect, the evaluation platform will simulate a mew random stacking scene again (Figure 2-5, (b)). If the algorithm result is correct, the estimated object will be removed. Because the algorithm may generate multiple results with only one sensor data, the evaluation platform will keep receiving algorithm results (Figure 2-5, (h)). After the algorithm ends (Figure 2-5, (i)), the evaluation platform will start the physics loop to simulate the stacking scene without the estimated objects. Or if there is no more objects in the box, the evaluation platform will simulate a new random scene again (Figure 2-5, (b)).

The evaluation platform will automatically evaluate the algorithm with the flow diagram shown in Figure 2-5.

2.3.4 Geometric Symmetry in 3D Model

Because the geometric symmetry in 3D model, the success criteria may vary from different types of model. In this subsection, the focus is on the geometric symmetries, which will affect the success criteria. This includes the rotational symmetry, circular symmetry, and cylinder-like symmetry. The reflection symmetry is not considered in this subsection, since it doesn't affect the success criteria. As well as simple objects, like sphere and cube, it should be handled as special cases.

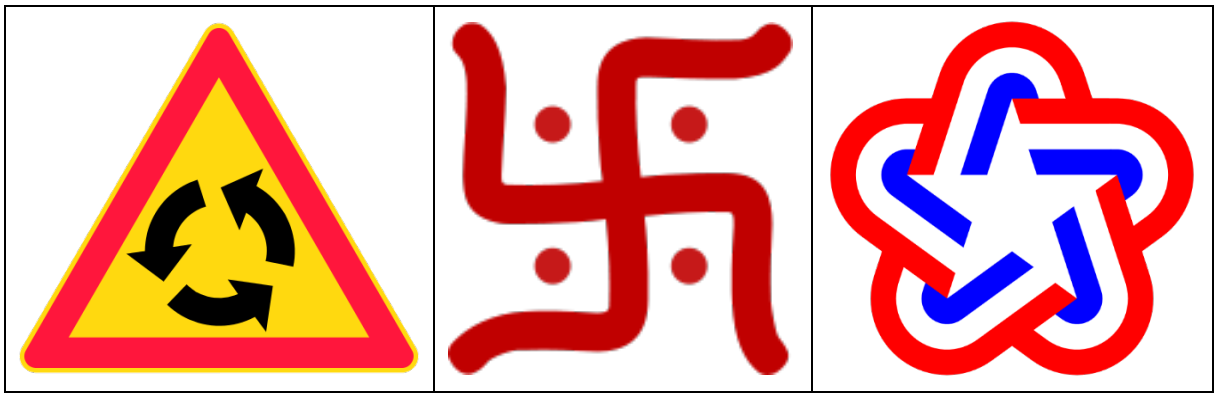

Figure 2-6. Illustration of 2D rotational symmetry. From left to right, the order is 3, 4, and 5.

Rotational symmetry of order n, means that by rotating an angel of $360^o/n$ (180^o, 120^o, 90^o, 72^o, 60^o, etc.) with respect to a particular point (in 2D) or axis (in 3D) does not change the

object. Figure 2-6, illustrates the 2D rotational symmetry with different order n. An example of 3D rotational symmetry with order 2 is the Wrench model (Figure 4-1, right).

Circular symmetry in mathematical physics applies to a 2-dimensional field which can be expressed as a function of distance from a central point only. This means that all points on each circle take the same value. In 3-dimension space, the appearance of the object does not change by rotating any angle with respect to a particular axis. This particular axis is called the axis of symmetry. Despite the small dissymmetry in Cylinder model (Figure 4-1, left) and Socket model (Figure 7-8), they have circular symmetry property. Because the size of those structure are not captured in the real sensor and the structure is not important for the assembly afterwards.

For Cylinder model (Figure 4-1, left), it has one circular symmetry, and infinite rotational symmetries of order 2. Since any axes that passes through the center point of the cylinder and perpendicular to the axis of cylinder is the axis of a rotational symmetry of order 2. A visualize explanation is given in Figure 2-7. We consider this type of model having cylinder-like symmetry.

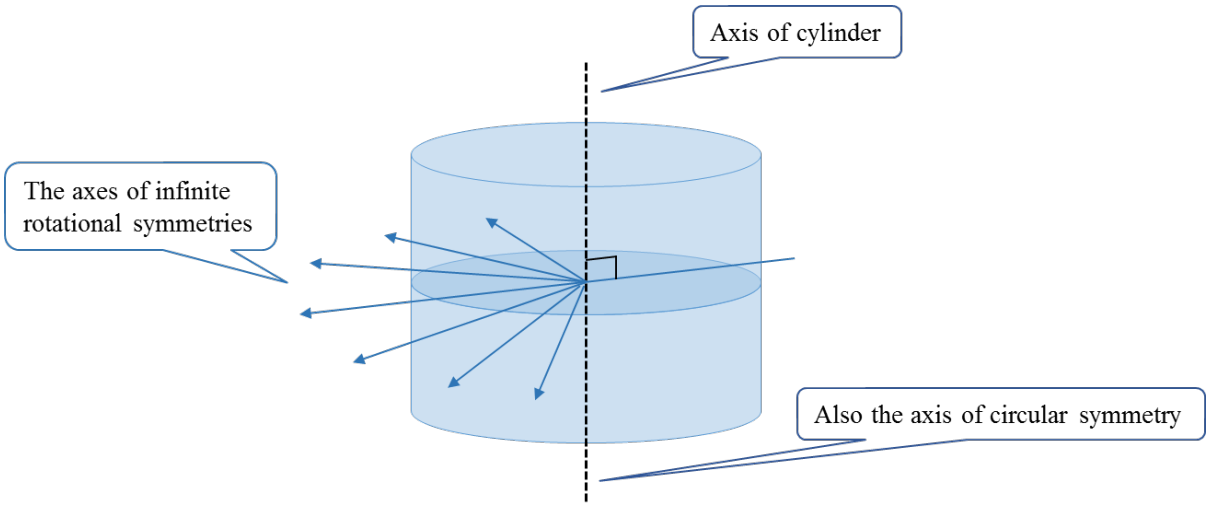

Figure 2-7. The symmetries in a cylinder.

The evaluation platform allow the user to specify symmetric properties for a model, then the evaluation platform will determine the success criteria according to the information given.

2.3.5 Success Criteria for Pose Estimation

The errors are expressed in translation error and rotation error. The translation error is the residual distance between estimated position and ground truth position. The rotation error are expressed in axis-angle representation (Figure 2-8), which can express a rotation in a three-dimensional Euclidean space by two values: a unit vector $\vec{u}$ indicating the direction of an axis of rotation, and an angle θ describing the magnitude of the rotation about the axis. Both errors can be extracted from the 4x4 error matrix, $\mathcal{E}$.

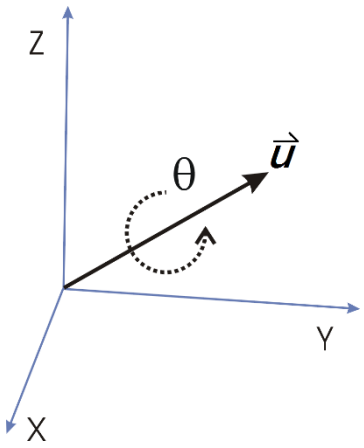

Figure 2-8. A visualization of a rotation expressed in axis-angle representation.

To determine if an estimated pose is success, the evaluation platform will first check whether the translation error is within tolerance value. Then the evaluation platform checks the magnitude of the rotation (θ_{error}) of the axis-angle representation. This two criteria are applicable to all type of models. Though, if a model have symmetry property, more conditions have to be considered.

For models with rotational symmetry. There are two additional criteria for this type of models, **deviation of the axis of symmetry** and **magnitude of the error rotation** (θ_{error}). Deviation of the axis of symmetry is the angle between axis of symmetry ($\vec{u}_{\text{axis of symmetry}}$) of the model and the axis of error rotation ($\hat{u}_{error}$):

$$\text{Deviation of the axis of symmetry} = \arccos(\vec{u}_{\text{axis of symmetry}} \cdot \hat{u}_{error}) \quad (3)$$

The deviation of the axis of symmetry makes sure the error rotation rotates about the axis of symmetry. The value of the magnitude of the error rotation depends on the order of the rotational symmetry. Take the Wrench model for example, it has a rotational symmetry of order 2, the magnitude of the error rotation (θ_{error}) should be close to 0^o or 180^o, the estimated pose is correct. If a model has a rotational symmetry of order 3, the magnitude of the error rotation (θ_{error}) should be close to 0^o or 120^o, 240^o. If both the criteria, deviation of the axis of symmetry and magnitude of the error rotation, are within tolerance rate, the estimated pose are correct.

For models with circular symmetry. The additional criterion is the **deviation of the axis of symmetry** (3). Since rotating any angle with respect to the axis of circular symmetry, does not change the appearance of the model. The magnitude of error rotation can be any value. If the deviation of the axis of symmetry is within tolerance value, then the estimated pose are correct.

For models with cylinder-like symmetry. The additional criteria are the **deviation of the axis of symmetry** and the **magnitude of error rotation**. This type of models can be evaluated using two cases. First case is the same with circular symmetry, only checks the deviation of the axis of circular symmetry. For the second case, the rotational symmetry in cylinder-like model, it will check if the magnitude of error rotation is 0^o or 180^o, and if the axis of error rotation are perpendicular to the axis of cylinder. If the criteria are with tolerance value, then the estimated pose are correct.

2.4 Features

Although, currently the evaluation platform is aimed for the random bin picking scenario, it's designed for a general purpose. Almost all the aspects of the evaluation platform are configurable, including the box size, stacking methods, and pose of virtual sensor. The virtual sensor could be located at a fixed position, or mounted on a robot arm. The synthetic sensor data are also configurable, including its resolution, FOV, working distance, intensity of effects, and synthetic noise. The virtual sensor can be adapted to simulate different real depth sensor with the same mechanism. It's important not to disrupt the original framework of the Gazebo, to simulate the virtual RGB-D sensor. We make use of the core of the Ogre3D, though the changes are only in the internal of the virtual depth sensor, and do not affect the original Gazebo. Therefore, others can take the virtual RGB-D sensor and put in to their simulator for other purposes.

The evaluation platform for pose estimation is capable of evaluating a multiple models pose estimation algorithm, see Figure 2-9. The box is filled with multiple models. When the algorithm result is given, the evaluation platform will first check if the recognized model is correct, and then the pose is estimated.

Another important feature is that the evaluation result will be logged, so that the user can know the performance of their algorithm. The information saved is enough to reconstruct the environment for assessment, where the user can find out the problem of their algorithm and develop a different approach to solve the problem. An example of the evaluation log (Figure2-9, top) and its visualization (Figure 2-9, bottom), are presented in Figure 2-9.

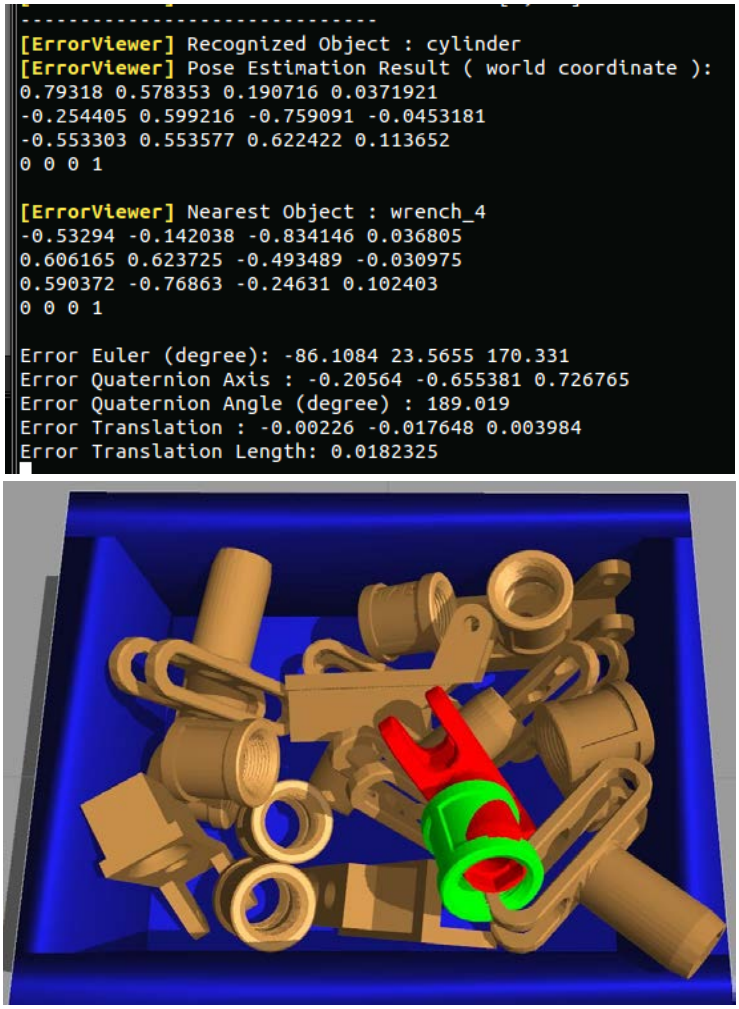

Figure 2-9. An example of an evaluation result for multiple models pose estimation. The top part shows the evaluation log of the result, including algorithm result, ground truth, and error information. And the bottom part shows the reconstructed environment of the evaluation result, the red object is the ground truth, and the green is the result from algorithm result.

Chapter 3. Pose Estimation Algorithm

In this chapter, we will introduce the pose estimation, [1], developed in ITRI. The pose estimation algorithm is paired with the evaluation platform to conduct several experiments in this thesis.

3.1 The ITRI Pose Estimation Algorithm

Figure 3-1, shows the flow diagram of the ITRI pose estimation algorithm. The algorithm has two stages, the off-line stage and the on-line stage. The off-line stage is executed in the virtual space, where the on-line stage can be executed either in the virtual space or in the real world.

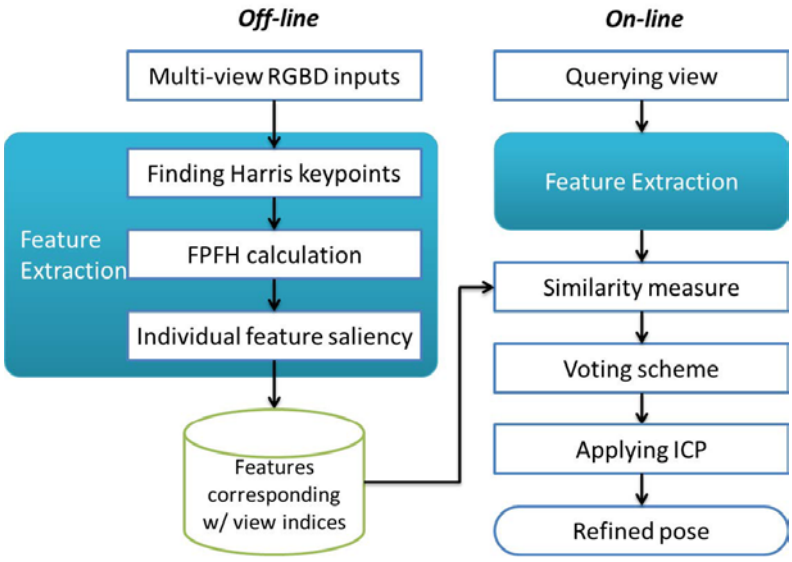

Figure 3-1. The flow diagram of the ITRI pose estimation algorithm.

3.1.1 Off-line Stage

In the off-line stage, the algorithm constructs the object database, which stores the features extracted from multi-viewpoint depth images. Here they use a virtual sensor to capture a depth image in each view. The depth image is converted to point cloud data using the sensor's parameters, followed by a feature extraction process. Since the feature plays a key role in matching an unknown view to the correct pose, a feature saliency analysis is applied, as

explained in subsection 3.1.3. After that, the multi-viewpoint features are stored into a database which is ready for pose estimation.

3.1.2 On-line Stage

In the on-line stage, the pose estimation process is performed on a query depth image. The depth image can be acquired by a real sensor or a virtual sensor. Features are extracted from the query depth image. Then the extracted features are compared to the features in the database. A weighted vote, according to the saliency of the features, is given to the views that contain the corresponding features. The views that have higher votes will be selected as candidate views. These coarse pose will then be refined using Iterative Closest Point (ICP) algorithm.

3.1.3 Feature Extraction and Individual Feature Saliency

To estimate the pose of an observed RGBD image set, one needs to extract its feature key points and match these key points to those previously stored in the database. Due to possible noise, lighting and shadow conditions, such key points will inevitably differ from those extracted from ideal models and may cause errors in pose estimation. Our objective, therefore, is to analyze each individual key point and determine which key points stay salient (thus more reliable) under noise. Salient key points are expected to be associated with higher importance (e.g. higher weights in pose estimation) while less salient ones should be given smaller weights.

They use 3-D Harris key point detection, since it is computationally efficient and relatively robust. For feature descriptors, they use the Fast Point Feature Histogram (FPFH), due to its recent success in many 3D recognition applications [16]. Though, other options such as 3D SIFT key point [17] and Shape Context feature descriptors [18] may also be included in the feature extraction process.

For both ideal and noisy 3-D models, they first extract key points and compute feature descriptors on these key points. To measure the descriptor saliency of the i-th noise-free key point, we collect all different key points on its noisy counterpart and calculate all pairwise descriptor distances to the noise-free key point descriptor,

$$d_{ij(j\neq i)} = dist(f_i, \hat{f}_j) \tag{4}$$

Where f_i denotes the descriptor of the i-th key point on the noise-free model and $\hat{f}_j$ is the descriptor of the j-th key point on the noisy model. For the distance between descriptors $dist(f_i, \hat{f}_j)$, they use symmetric Kullback-Leibler divergence (KLD). Although other distances such as histogram intersection are also available. They then warp the KLD into range [0,1] using the sigmoid function:

$$\theta_{ij} = \frac{2}{1+e^{-\lambda d_{ij}}} - 1 \tag{5}$$

The saliency of the i-th key point (θ_i) is then obtained through the average of all θ_{ij}'s.

$$\theta_i = 1 - \frac{1}{N-1}\sum_{j=0, j\neq i}^{N-1} \theta_{ij} \tag{6}$$

Where $\theta_i = 1$ indicates high saliency and $\theta_i = 0$ indicates low saliency.

Chapter 4. Physics Simulation

In this chapter, we will provide some information to improve the physics simulation in Gazebo. In order to achieve realistic simulation, we must handle every aspect with caution, which includes dynamics simulation and collision detection.

4.1 Dynamics Simulation

Dynamics simulation handles the motion of objects affected by forces and torques. An object's physics properties must be provided correctly so that the physics engine could simulate the motion of an object precisely. These properties includes mass, the center of gravity, inertia matrix, friction, etc. With the help of a CAD model, the acquisition of these information becomes easy. The model used in this thesis is shown in Figure 4-1, which is about 50 millimeters. With the CAD model, we can calculate these properties using Solidworks. If we have the real object we can measure the mass of the object directly and obtain the density, or we can use Solidworks to calculate the mass using the CAD model and its material's density. The center of gravity and inertia matrix can be determined using the density and the shape given in the CAD model. The friction can be determined by the material of the model. Figure 4-2, shows an example of calculating physics properties using Solidworks.

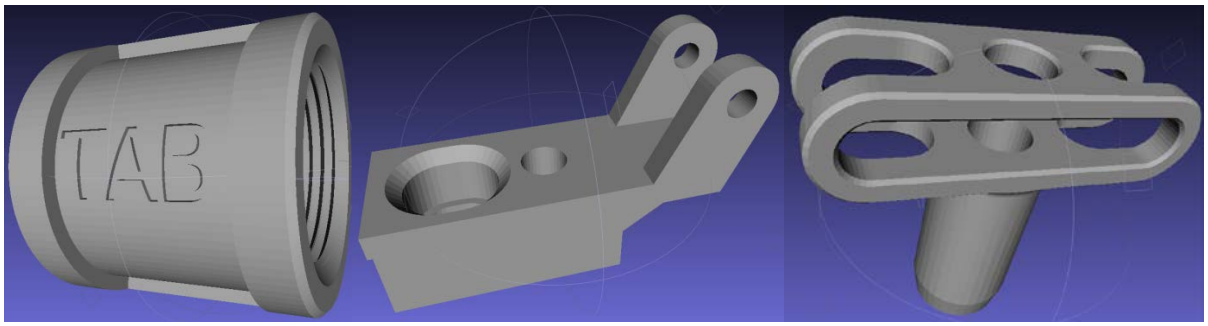

Figure 4-1. CAD models used in this thesis, from left to right, include Cylinder, Brazo, and Wrench. The size of the model is around 50 millimeters.

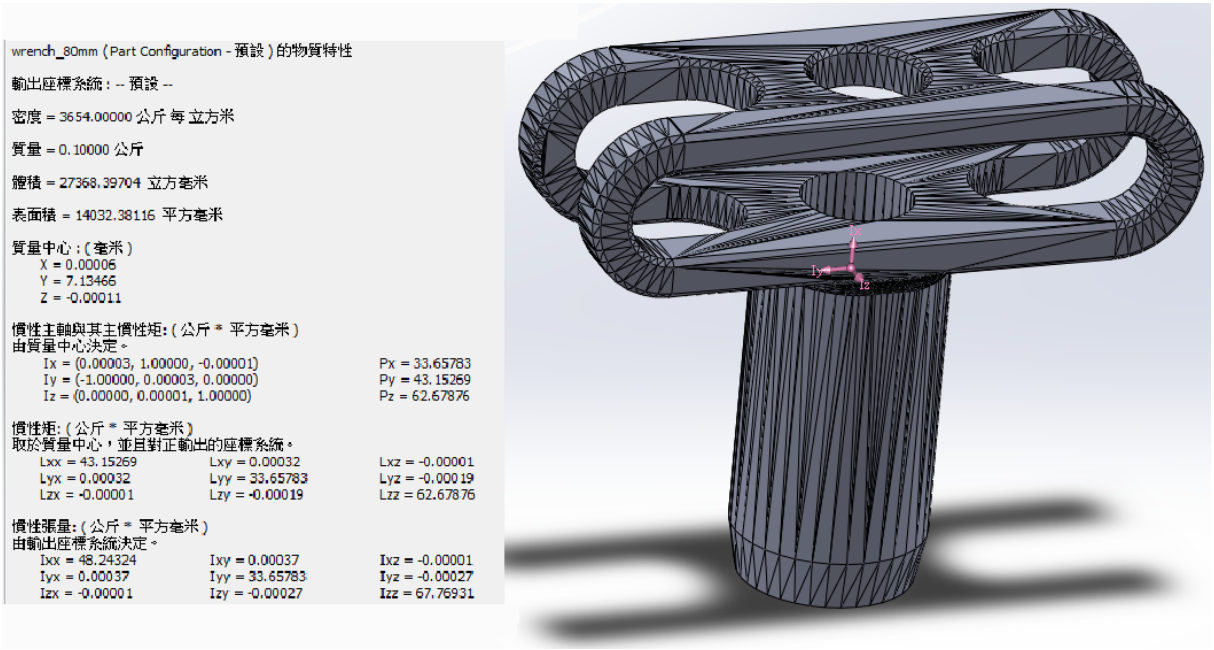

Figure 4-2. An example of calculating physics properties.

4.2 Collision Detection

In Gazebo, the physics engine will handle the collision between two objects. The collision shape can be generated directly from the CAD model automatically, though due to the mechanism of collision detection, an undesired result could happen. To simulate the collision of two objects, the two objects will penetrate slightly into each other, then recover from the penetration. What makes it worse is that if objects stack over one another, the penetration depth increases. This mechanism will have no problem on thick models, but when it comes to a thin model, an undesired result like Figure 4-3. may happen. Because of the shape of the Wrench model, the penetration went through the thin part of the model.

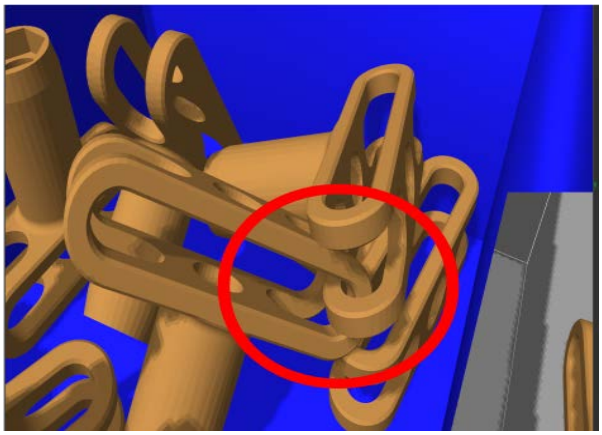

Figure 4-3. An undesired result of Wrench physics simulation, which shows that two models are locked

Unfortunately, there is no general solution for this kind of problem. Instead, a modification to the model could prevent this from happening. Although some stacking poses will not be simulated, but since the probability of that stacking poses is low, it should be a good trade-off for the more undesired situation. We use MeshLab to modify the Wrench model, we sealed the hole at the side of the model (Figure 4-5), so that the recovery of the penetration process will not end up in lock situations.

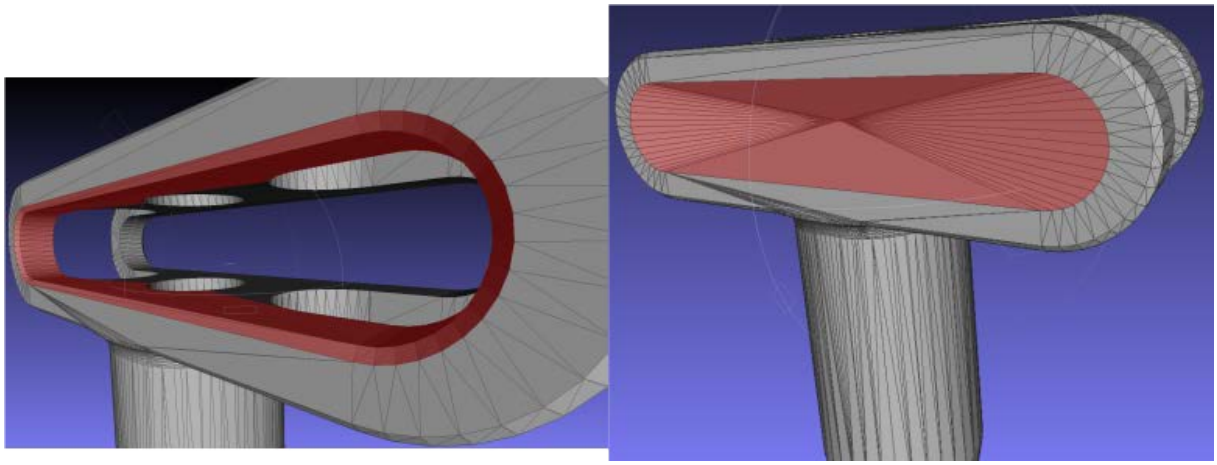

Figure 4-4. Semi-solution for penetration issues. (Left) Red indicates the faces being selected. (Right) The modified Wrench models.

4.3 Random Stacking

With the efforts made in this chapter, an example of stacking result of each model is shown in Figure 4-6. The size of box is 210 x 160 x 80 mm. It is filled with 18 models. Despite there will be jitters that may happen when the model have thin parts, like Wrench and Brazo, the random stacking result is good. Although it’s hard to judge the rationality of the pose from a 2D view, we have examined closely by orbit around the box and come to a conclusion that the random stacking is reasonable in the sense of physics.

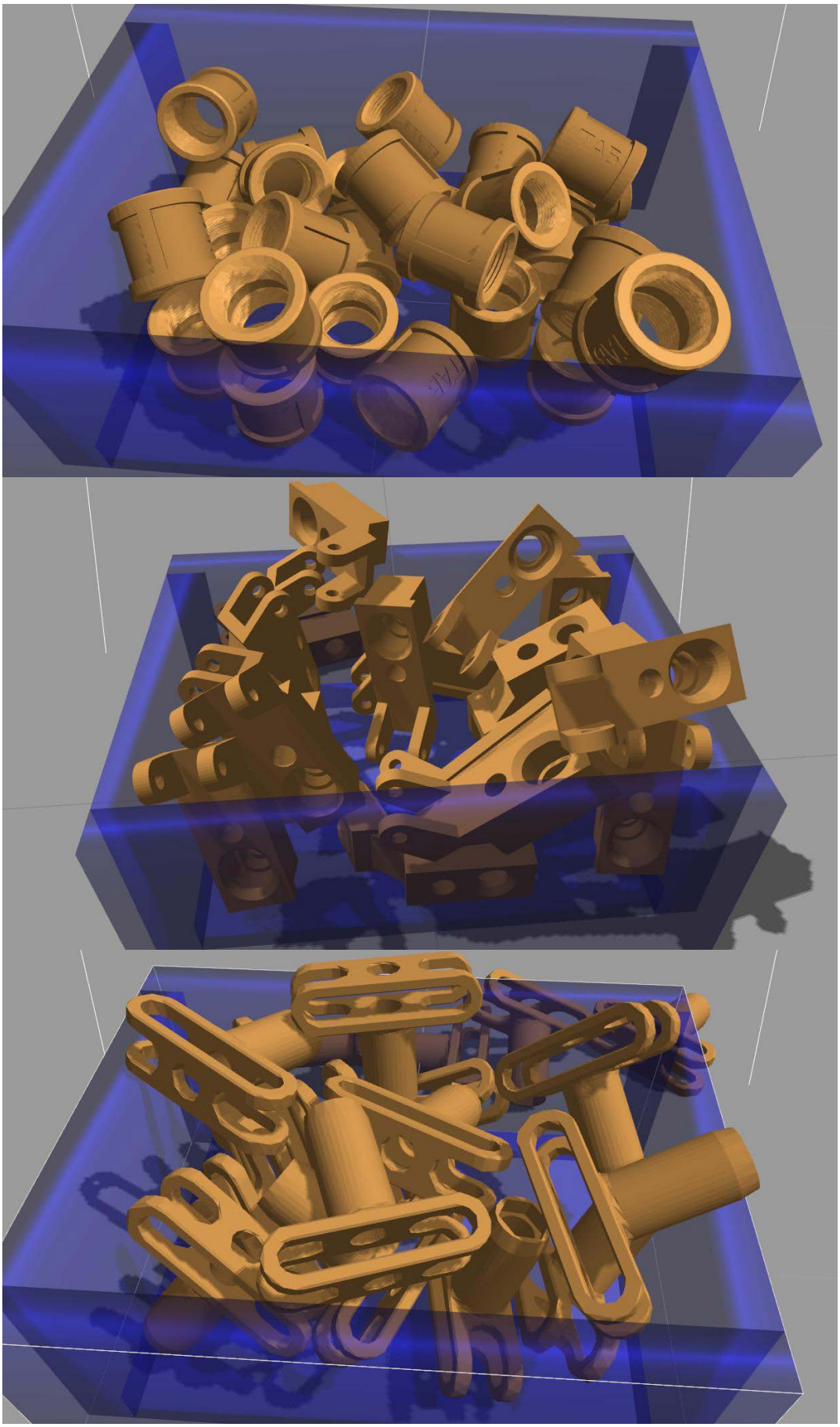

Figure 4-5. The stacking result of Cylinder (Top), Brazo (middle), and Wrench (Bottom).

Chapter 5. The Virtual RGB-D Sensor

In this chapter, we will present a series of systematic methods to simulate a real depth sensor. Although, Gazebo provides some sensors, ray sensor, depth sensor camera, it's far from resembling a real sensor. By taking advantage of modern computer graphics, we manage to create a realistic depth image which resembles an image taken from the real depth sensor.

5.1 Target RGB-D Sensor

In this thesis, we will focus on virtualizing a RGB depth sensor developed in the Industrial Technology Research Institute of Taiwan. The depth sensor has one RGB camera, two IR cameras and one IR projector, as shown in Figure 5-1. It is a kind of active stereo camera, it projects structural lights on the scene, and uses triangulation to generate the depth image. Some specifications are listed in Table 5-1.

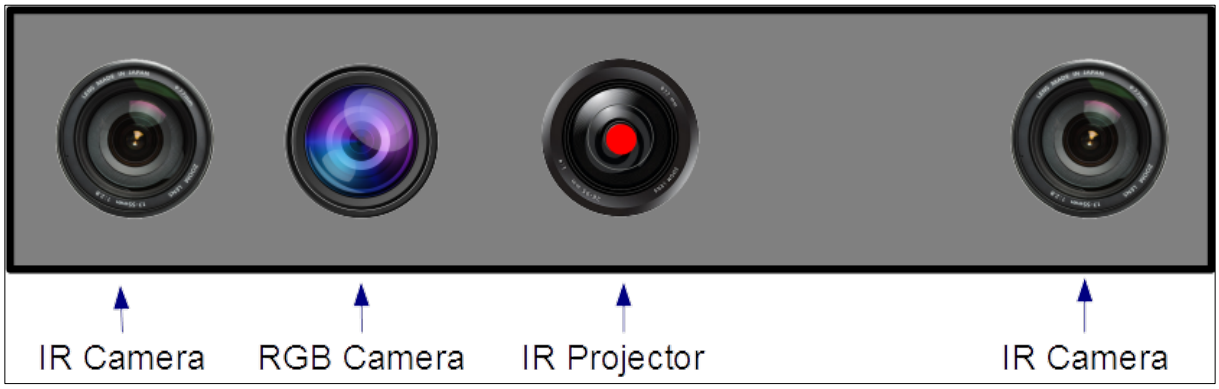

Figure 5-1. Illustration of an ITRI depth sensor.

Sensor	ITRI Depth Sensor
Resolution	1280 x 960 pixels
Field of View (Horizontal)	16.06 degrees
Working Distance	800 mm
Working Range	700 ~ 900 mm
Sampling Rate	3 Hz
Noise Average Magnitude	0.2 mm
Noise Standard Deviation	0.25 mm

Table 5-1 Specifications of the ITRI depth sensor.

Due to physics limitations and the algorithm itself, defects and noise can be seen in the depth image (Figure 5-2) taken from the ITRI depth sensor. From observations, we categorize these attributes of the depth sensor into five categories, the occlusion effect, intensity saturation, confidence effect, blurring effect and sensor noise. We will show how to simulate every attribute in the next section.

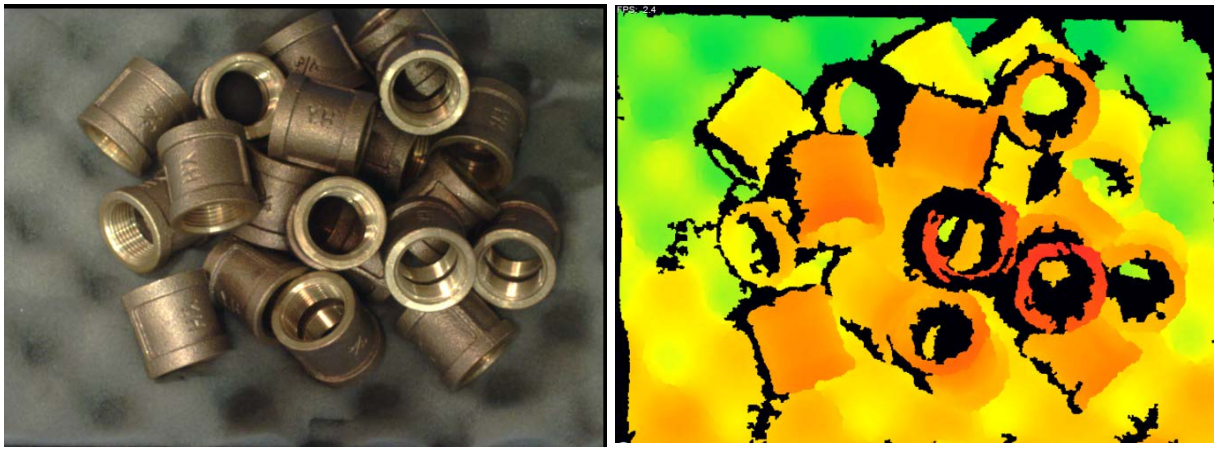

Figure 5-2. Images taken from the ITRI depth sensor, RGB image (left), depth image (right). In the depth image, black pixels indicate the undetermined depth value at that point.

Since the sensor data (Figure 5-2) are similar to the Kinect sensor , we model it as a Kinect-like depth sensor (Figure 5-3), a distance of 100mm between the IR projector and the IR camera, and an RGB camera located at the same position of IR camera. As for the specifications of the virtual depth sensor, it will be the same as the ITRI depth sensor, including resolution, field of view, working range, etc. For the rest of the section, this model will be used to generate a synthetic depth image.

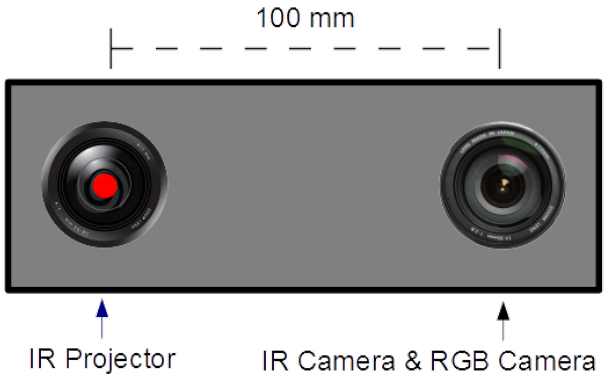

Figure 5-3. Model of virtual depth sensor.

5.2 Extract Depth Image

To extract depth image, we use a computer program called shader. Shaders replace a section of video hardware typically called the Fixed Function Pipeline (FFP), so-called because it performs lighting and texture mapping in a hard-coded manner. Shaders provide a programmable alternative to this hard-codded approach [11]. It's used to program the programmable GPU rendering pipeline, and to achieve customized shading in computer graphics. Normally, we will get RGB colors from the rendering pipeline output, but with shader, we can get any information that is available in the rendering pipeline, like the position of vertex, normal of vertex in different space, and light position, light direction, camera position, etc.

In the shader we can obtain the position of vertex in camera space, which coincides with the information obtained from a real depth sensor. Using shader to extract the depth information of the scene is very efficient, comparing to the ray tracing method. This is because the shader is written to apply transformations to a large set of elements at a time, for example, to each pixel in an area of the screen, or for every vertex of a model. This is well suited to parallel processing, and most modern GPUs have multiple shader pipelines to facilitate this, vastly improving computation throughput.

An example of depth image extracted from computer graphics is shown in Figure 5-4, apparently, it's nowhere similar to the depth image taken from the real depth sensor (Figure 5-2). So attributes of a depth sensor must be simulated and applied to the perfect depth image in order to achieve a realistic result.

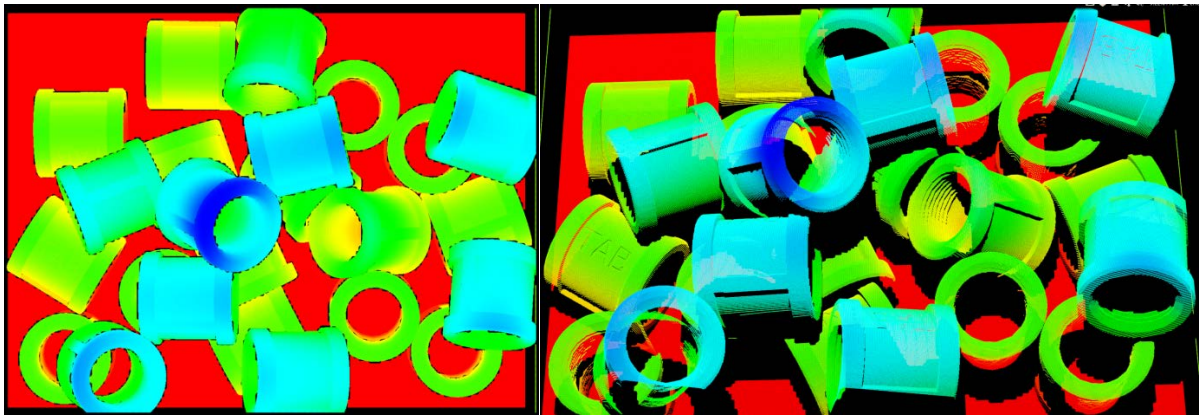

Figure 5-4. An example of a perfect depth image from computer graphics, 2D view (left), 3D view (right).

5.3 Simulation Methods

5.3.1 The Occlusion Effect

This phenomenon is inevitable in stereo vision. In other word, any measurement device that requires using two sensing equipment will have an occlusion phenomenon. In our case, one point must be visible to both the IR camera and the IR projector marked as green in Figure 5-5, in order to measure the depth value of that point. The red in Figure 5-5 indicates places that are occluded by the rising edge of the plane, only the IR camera can see those places but not the IR projector, so there will be no available depth value in those places. This occlusion phenomenon appears in Figure 5-2 (right) as black pixels. It should be noted that every pixel in the depth image is visible to the IR camera but not the IR projector.

In Figure 5-5, if we put a point light at the position of the IR projector, then the red area will be shadowed by the rising edge. So if we can determine places that are in shadows, then we can know where the occlusion phenomenon will appear.

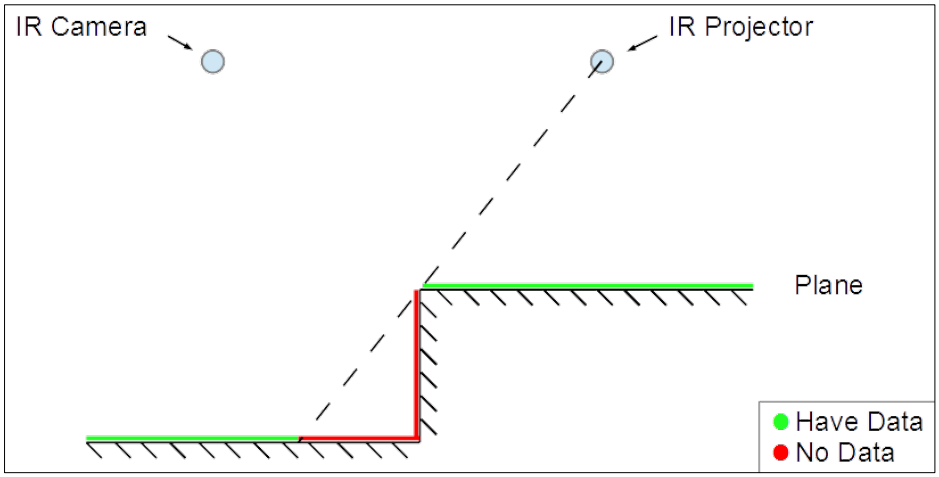

Figure 5-5. Illustration of an occlusion phenomenon.

To simulate this phenomenon, we make use of the Depth Shadow Mapping technique in Ogre3D, which is an image-based shadow determination algorithm. The simulation method is consist of two passes. First, it renders a 2D depth map from the IR projector's point-of-view. Second, it renders a scene from the IR camera's point-of-view. In the second pass, the shader will determine point seen by IR camera's xyz position relative to the IR projector, and compare the point's depth z to the depth value stored at position xy in the depth map. If the point's depth z is greater than the depth value stored in the depth map, the point is occluded as something closer to the IR projector that occlude it. An illustration of the simulation method is in Figure 5-6.

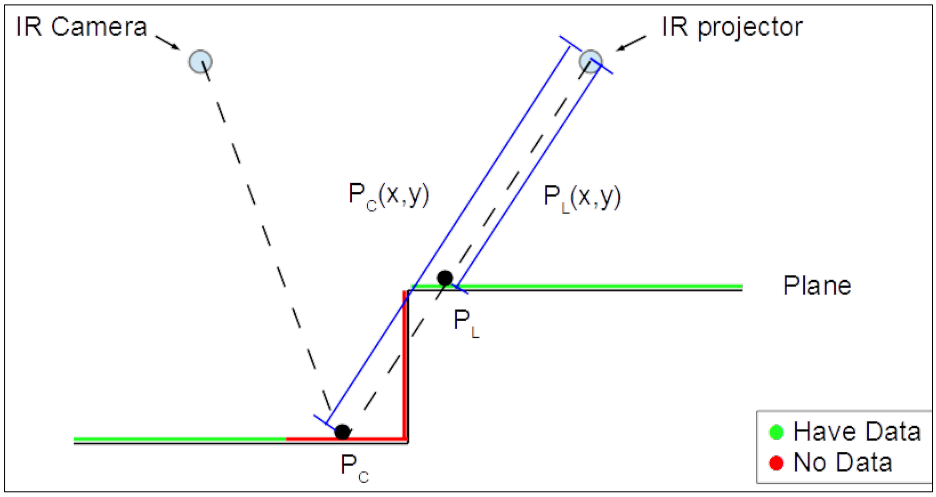

Figure 5-6. Illustration of the simulation method for an occlusion phenomenon. First, a 2D depth map XY is created from the IR projector's point-of-view. In the second pass, the IR camera sees a point P_c, then the point is transformed to IR projector's point-of-view, xyz, where $z = P_c(x,y)$. The depth value $P_c(x,y)$ is compared to the depth value $P_L(x,y)$ at position (x,y) stored in the depth map, if greater ($P_c(x,y) > P_L(x,y)$), the point P_c is occluded (by point P_L) and thus we can determine that no depth value is available at that point.

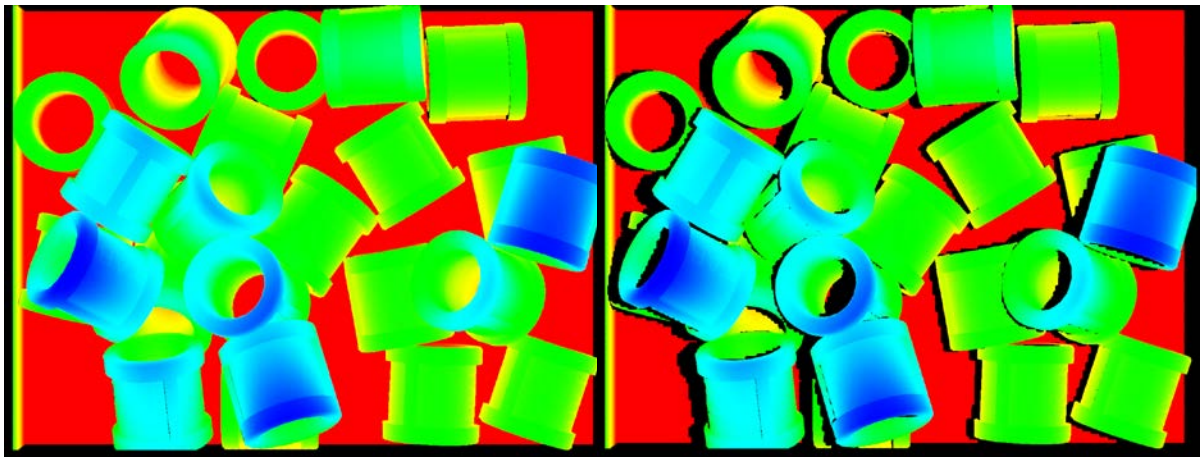

Figure 5-7. Difference between perfect data (left) and simulated data with an occlusion effect (right). The IR projector is on the right side of picture and the IR Camera is on the left side of the picture. Notice that the occlusion effects appear at the left side of objects, and the occlusion effect is more severe in the left side of the image than the occlusion effect in the right side of image.

The simulation result of an occlusion effect is shown in Figure 5-7. Unavailable depth value at a pixel is painted in black, where occlusion effects occurred.

5.3.2 Intensity Saturation of Projected Light

When the structural light projected on a smooth surface, the intensity of the reflected light will be too high, and therefore the IR camera will have difficulties in these situations, an example is shown in Figure 5-8. As you can see, some white strip appears in the inner side of the cylinder (Left of Figure 5-8), which results in failure of depth calculation.

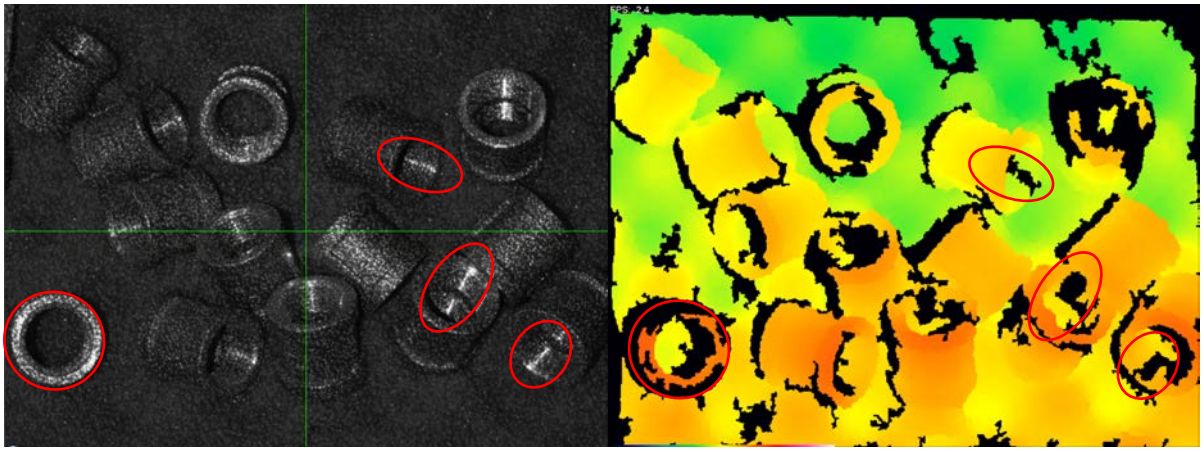

Figure 5-8. An illustration of intensity saturation effects. The left picture shows the random dots projected on the scene. The right picture illustrates the depth image from depth sensor. It should be noted how the high intensity dots affects the final depth image.

This phenomenon is the result of specular reflection, which is distinct from diffuse reflection, where incoming light is reflected in a broad range of directions. Specular reflection is the mirror-like reflection of light from a surface, in which light from a single incoming direction is reflected into a single outgoing direction. Such a behavior will cause intensity saturation, due to all the energy from the IR projector is reflected into the IR camera. Since the IR camera cannot identify the random dots, which become a white strip, the depth sensor is no longer capable of determining the depth value in this situation.

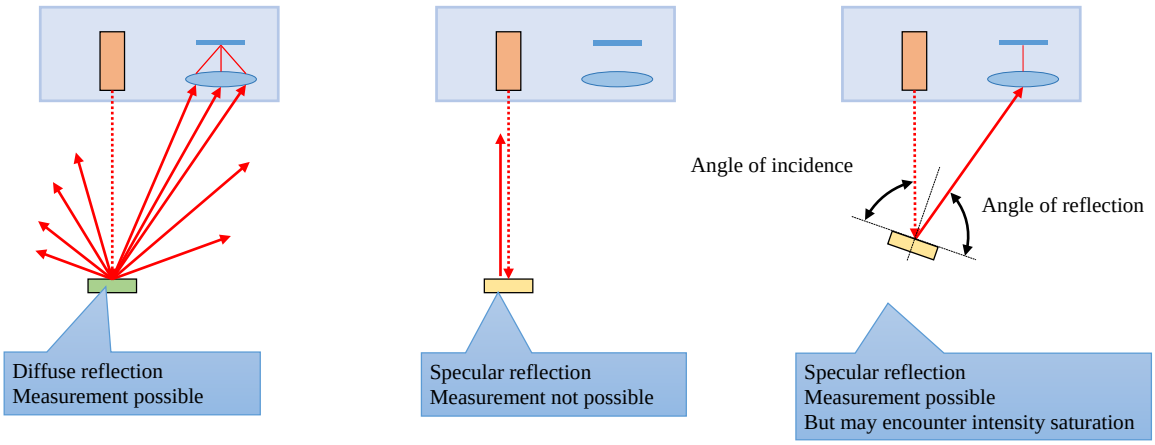

Figure 5-9. The different types of reflection's influence on the depth sensor are illustrated.

Another possible defect could also happen if the reflected light didn't reach the IR camera. In contrast to the diffuse reflection, specular reflection will only reflect the light in one direction, so if the IR camera is not in the path of the reflected light, the depth value is not available either. So when the depth sensor encounters a polished surface model, two problems mentioned above are likely to happen (see Figure 5-9).

To simulate this phenomenon, we use the very basic of the computer graphics. Normally, there are three settings that can affect the appearance of an object, which is the ambient color, diffuse color, and specular color. The ambient color determines how much ambient light (directionless global light) is reflected. The diffuse color determines how much diffuse light is reflected. The specular color determines how much diffuse light is reflected. First, we split the model into two parts, one for a smooth surface and the other for a rough surface. Second, we set the diffuse color and specular color to mirror the appearance of the real model (Figure 5-10). Third, we extract only the specular info of the scene (Figure 5-11), we can obtain the intensity of specular reflection. Finally, we determine a threshold based on the tolerance for the intensity of the specular reflection, those intensities that exceed this threshold will have no depth value (Figure 5-12).

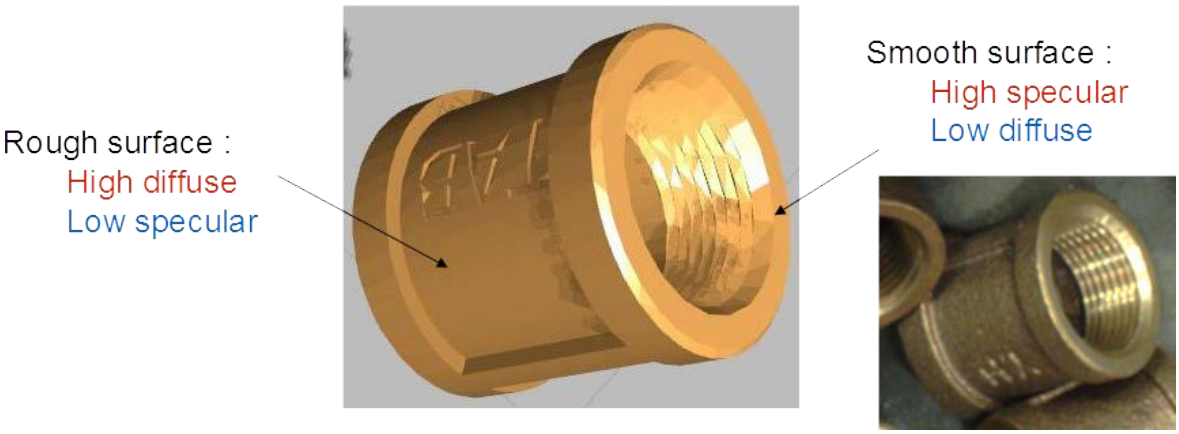

Figure 5-10. The simulation result of a two material object. The object at the lower-right is its appearance in the real world.

We only simulate intensity saturation, one of the phenomenon caused by specular reflection. Since most objects are the combination of diffuse and specular reflection material, it is not likely that there will be no light received by the IR camera. Nevertheless, it is easy to extend this method to support this phenomenon, by using diffuse and specular information extracted from the scene.

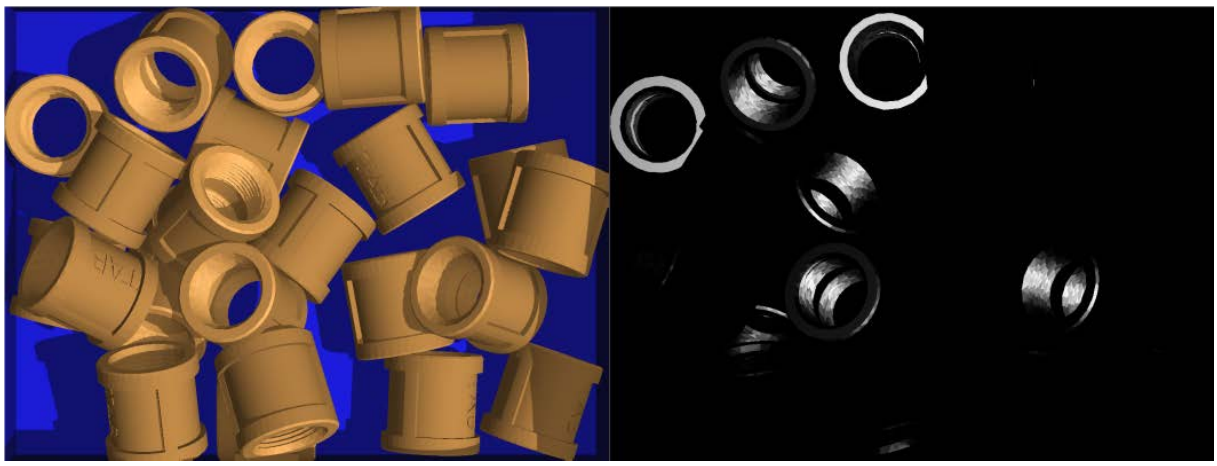

Figure 5-11. (Left) The RGB image of the scene. (Right) Extract only the specular information, the brighter of the color the higher of the intensity

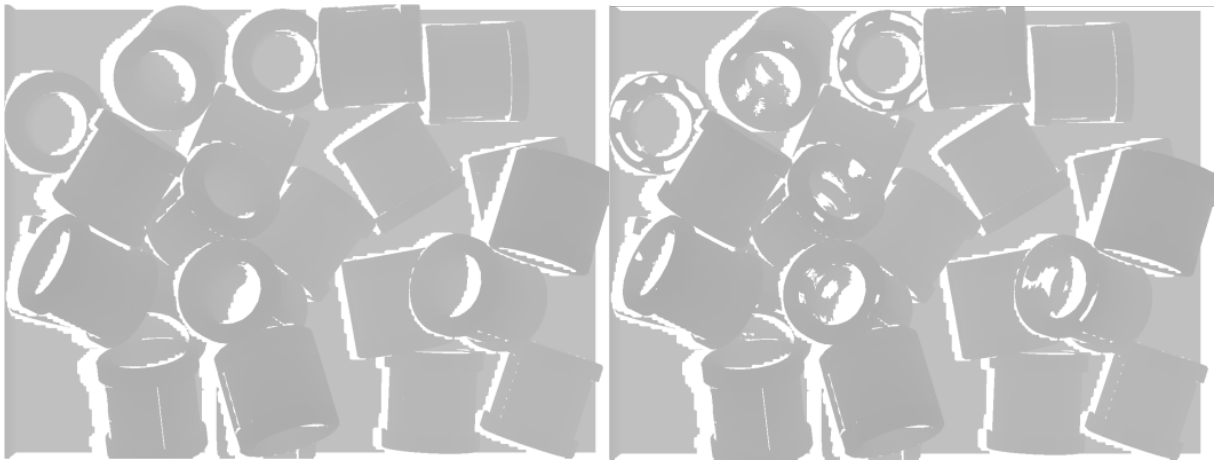

Figure 5-12. (Left) Simulated data with only occlusion effects. (Right) Simulated data with intensity saturation effects added. Be aware that some disturbance is added to the threshold, to create a more realistic synthetic data.

5.3.3 Low Confidence Effect

The cause of this effect occurs when the depth sensor tries to solve the depth of a point P, where the point normal $\vec{n}$ and the view ray $\vec{r}$ become approximately perpendicular (Figure 5-13(a) and (b)). In this case, we say that the depth sensor has low confidence on the depth value calculated at that point. The definition of confidence:

$$confidence = \cos\theta = \frac{\vec{n} \cdot \vec{r}}{\|\vec{n}\|\|\vec{r}\|} \tag{7}$$

$$Range(confidence) = [0,1] \tag{8}$$

In contrast to the high confidence situation (Figure 5-13(c)), low confidence has problems when determining the depth value. One of the problems is that the light is reflected away from the IR camera, causing a low intensity situation. The other problem is that when the light is projected on a steep surface, causing a deformation of the structural light. The combination of these two problems will affect the depth value calculated at the low confidence situation. It may cause higher noise level and even make the depth sensor not able to determine a depth value.

To simulate this effect, we use the power of shader. In the shader we can get information about the object in the scene, including the camera position, point position, and point normal. Using these pieces of information, we can calculate the confidence at every point. From

experiments, a value of 75 degree is set for the confidence threshold. The simulation result of the confidence is shown in Figure 5-14.

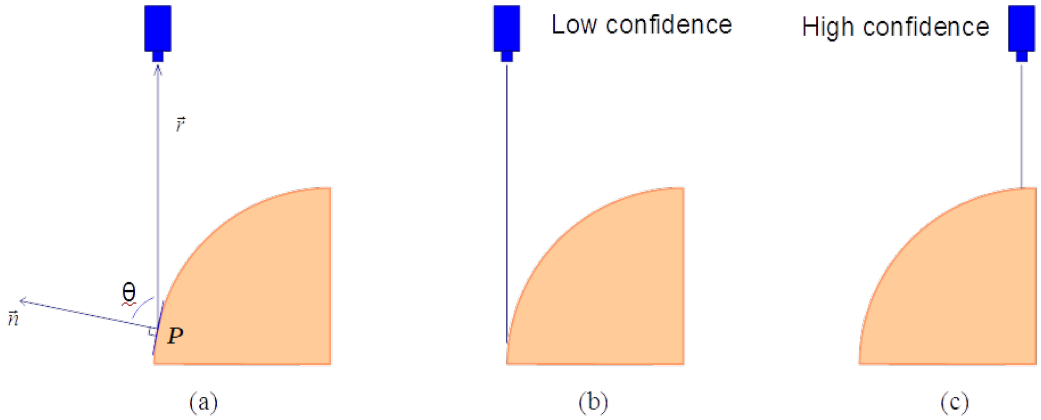

Figure 5-13. (a) Illustration for explaining the low confidence effect. (b) and (c) show the difference between high confidence and low confidence.

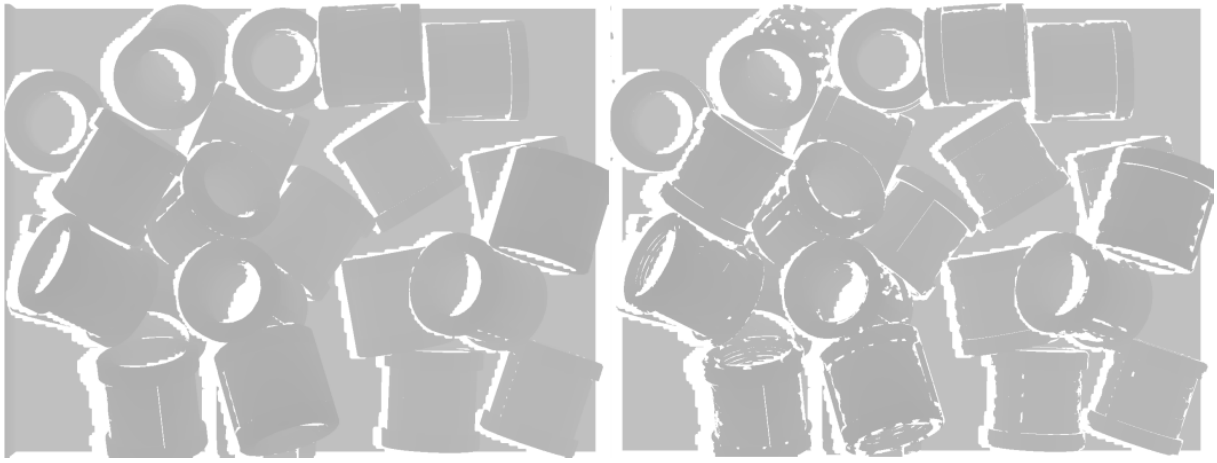

Figure 5-14. (Left) Simulated data with only occlusion effects. (Right) Simulated data with confidence check added. Be aware that some disturbance is added to the threshold, to create more realistic synthetic data.

5.3.4 Noise Synthesis

We use the method proposed in [2] for our noise synthesis. It uses the basic knowledge of the Fourier Transform, which can preserve the noise's properties without losing the randomness of the noise. In 1D Fourier Transform, a signal can be transformed into the magnitude and phase of the spectrum. The magnitude holds the information of the amount of periodic waves in different frequencies, while the phase holds the information of the position of the periodic waves. Similarly, in 2D Fourier Transform, the magnitude holds the information of the amount of different patterns, while the phase holds the position of different patterns. By preserving the magnitude of the spectrum of the noise, the composition of the noise is preserved. And by randomly generating the phase, we can achieve the randomness of the noise. So the noise synthesis method is to use inverse Fourier Transform of the original magnitude of the noise and random phase. We show the noise synthesis method in the following steps:

(1) Capture a depth image of a plane, perpendicular to the view ray.

(2) Extract sensor noise from the captured depth image.

(3) Apply 2D Discrete Fourier Transform on sensor noise, with equation:

$$F(u,v) = \frac{1}{MN}\sum_{x=0}^{M-1}\sum_{y=0}^{N-1} f(x,\mathrm{y})e^{-i2\pi\left(\frac{ux}{M}+\frac{vy}{N}\right)} \tag{9}$$

(4) Preserve the magnitude of the spectrum except for low frequencies, and generate the random phase of the spectrum.

(5) Generate noise using inverse 2D Discrete Fourier Transform of the original magnitude and random phase, with equation:

$$f(x,y) = \sum_{u=0}^{M-1}\sum_{v=0}^{N-1} \mathrm{F(u,v)}e^{i2\pi\left(\frac{ux}{M}+\frac{vy}{N}\right)} \tag{10}$$

Figure 5-15, Left, shows the synthesized noise by the method mentioned above. The synthetic noise in this case has a slightly different structure to the original one, which is to be

expected as there is some structure in the phase of the noise. Nevertheless, the structure of this synthetic noise is quite similar to that of the original noise at least with respect to the magnitudes, sizes, shapes, and density of the bumps. It is certainly closer to real depth sensor noise than white Gaussian noise. By using this method, the statistical analysis on noise can be avoid, while the properties of the noise are mostly captured. Figure 5-16, shows the synthetic noise applied on the point cloud generated from the simulator.

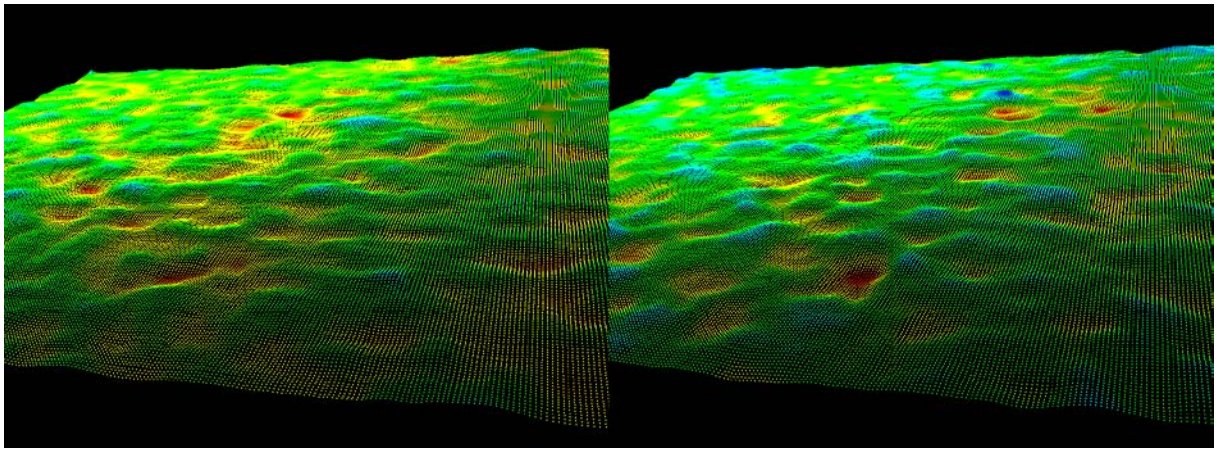

Figure 5-15. (Left) Noise from the real depth sensor. (Right) The Noise synthesis result.

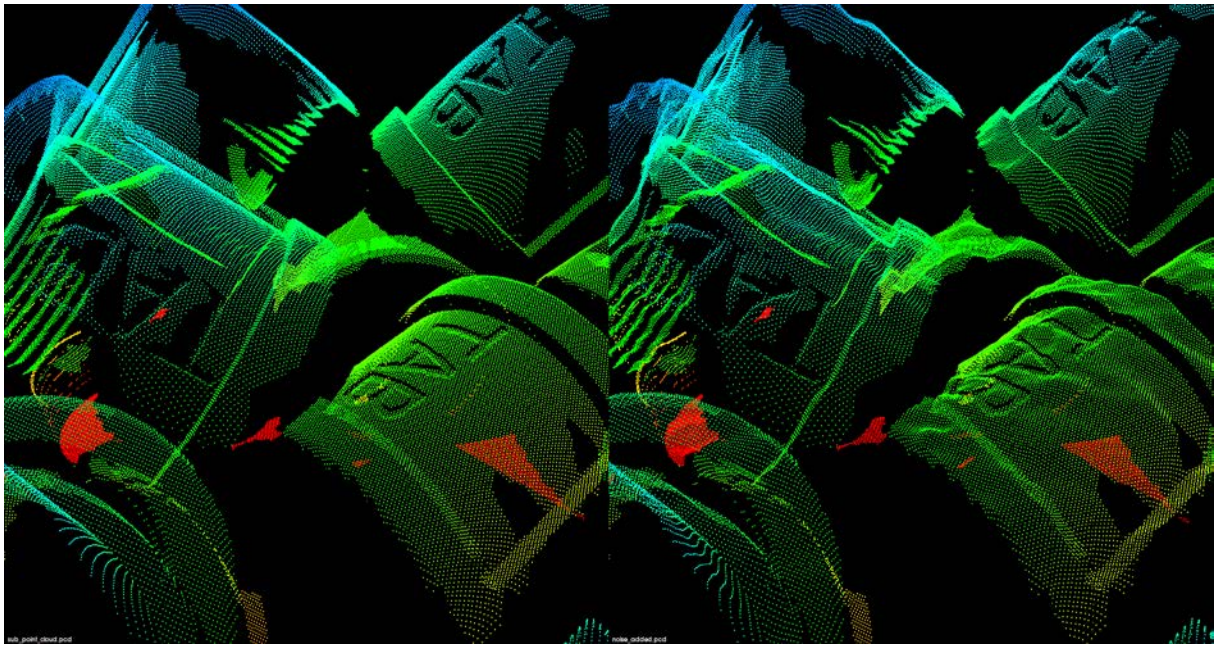

Figure 5-16. (Left) Point cloud without noise. (Right) Point cloud with synthesized noise added.

5.3.5 Blurring Effect

This effect is caused by the internal stage of the depth retrieving algorithm. As you can see in Figure 5-17, the edge is smooth, and the points are connected together at the junction of models. The simulation method is simply to apply a smoothing filter to the depth image, and a bilateral filter is used to prevent any undesired result at the edge of objects. A simulation result is shown in Figure 5-18.

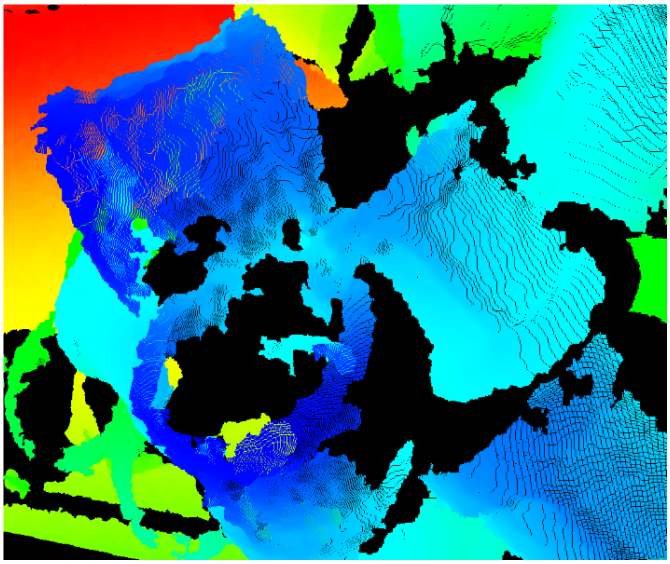

Figure 5-17. Shows the blurring effect captured by the real depth sensor.

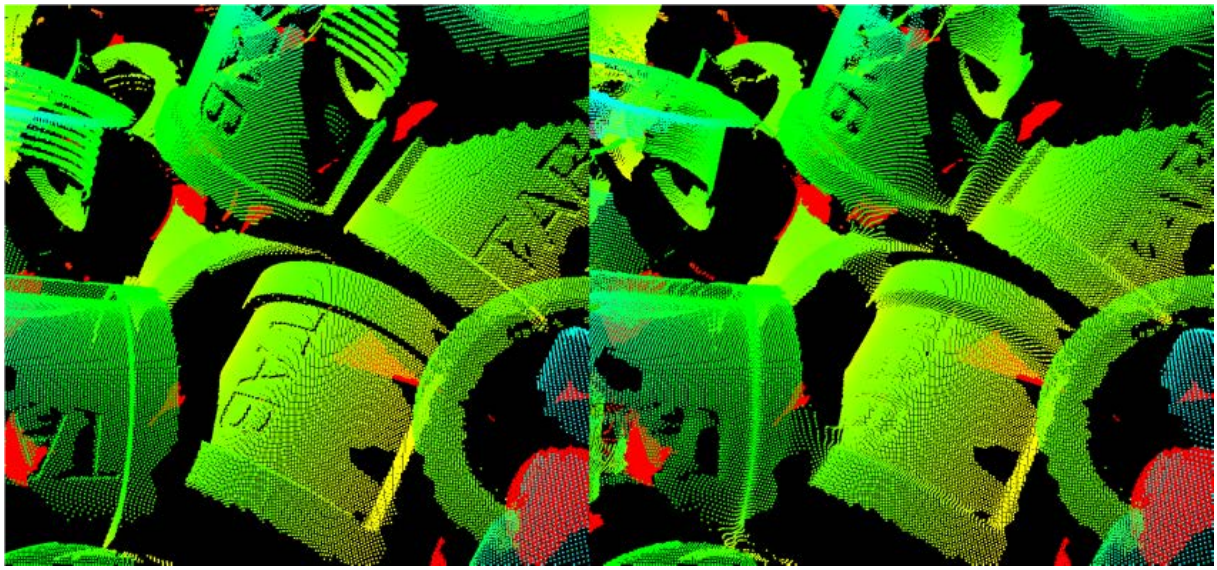

Figure 5-18. (Left) Without blurring effect. (Right) With blurring effect.

5.3.6 The Simulation Result

We provide a series of simulation methods to mimic the real depth sensor in the virtual space. These attributes mentioned are common in other depth sensors, including Kinect-like depth sensors. By adjusting the parameters in these simulation methods, it is possible to virtualize any depth sensor that has the same mechanism. An example of a simulation result is given in Figure 5-19. The parameters in simulation methods are adjusted to mimic the ITRI depth sensor. Compared with the data (Figure 5-2) captured from the ITRI depth sensor, the simulation result is much more similar to the real data than the perfect data (Figure 5-19, Left).

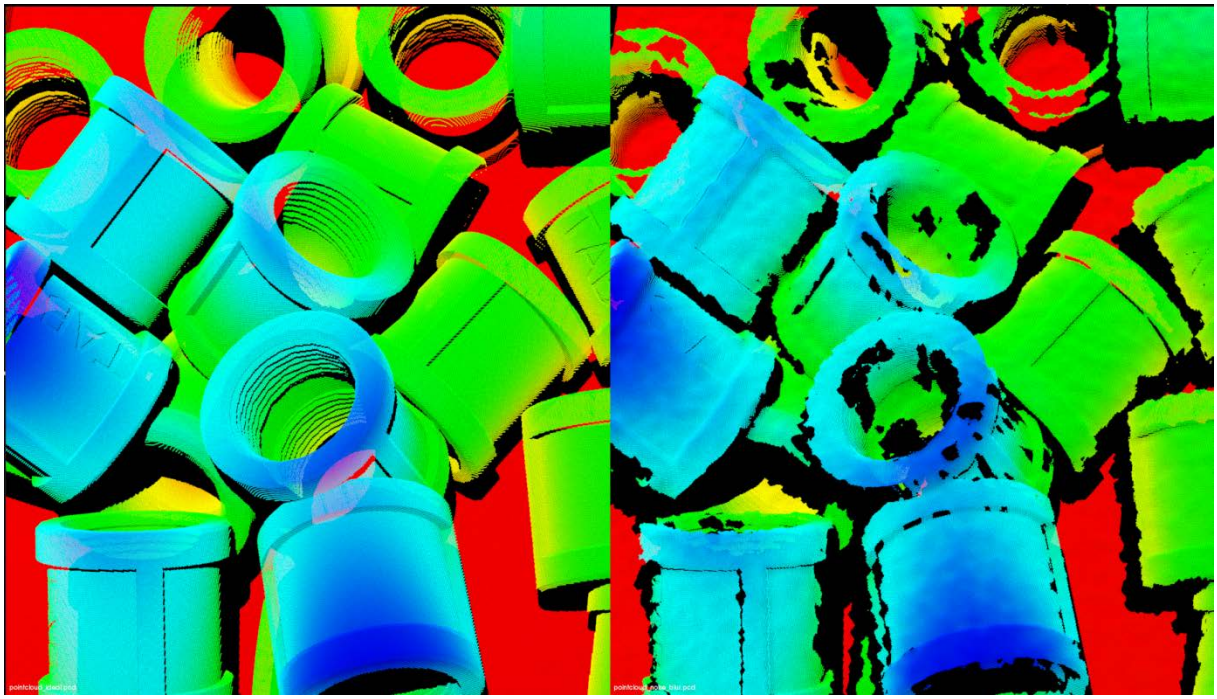

Figure 5-19. (Left) Perfect depth data in 3D view. (Right) Simulation methods applied on perfect depth data to mimic the ITRI depth sensor.

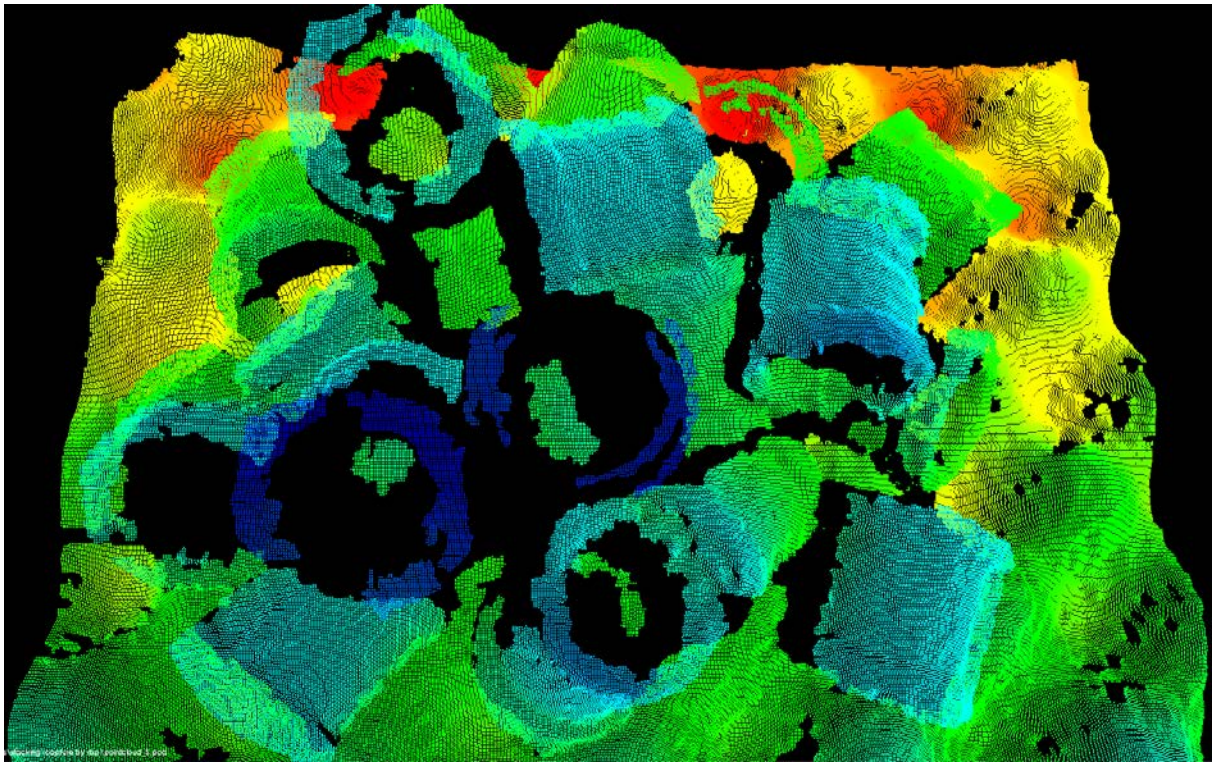

Figure 5-20. Point cloud captured by ITRI depth sensor in the 3D view.

5.3.7 The Disturbance on Threshold

In previous subsections, we apply disturbance on threshold to achieve natural result. The disturbance is a 2 dimension coherent noise, which means that for any two points in the space, the value of the noise function changes smoothly as you move from one point to the other. The noise is called Perlin Noise [15]. An example of the 2 dimension noise is in Figure 5-21. The thresholds are disturbed by the corresponding pixels in 2 dimension noise.

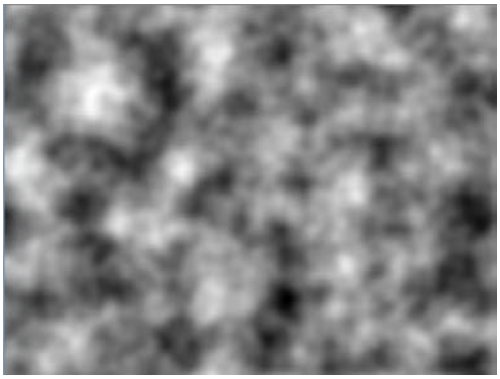

Figure 5-21. An example of 2 dimension coherent noise generated by Perlin Noise.

Many people have used random number generators in their programs to create unpredictability, make the motion and behavior of objects appear more natural, or generate textures. Random number generators certainly have their uses, but at times their output can be too harsh to appear natural. The differences between non-coherent noise and Perlin Noise can be seen in Figure 5-22.

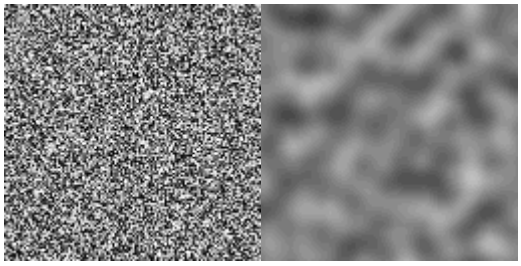

Figure 5-22. The differences between non-coherent noise (left) and Perlin Noise (right).

Though, same effects can obtained by applying a smooth filter on the non-coherent noise. But this can end up being computationally expensive, and are not as controllable as the Perlin Noise. The Perlin Noise is a procedural texture primitive, a type of gradient noise used by visual effects artists to increase the appearance of realism in computer graphics. The function has a pseudo-random appearance, yet all of its visual details are the same size (Figure 5-23, upper-left, lower-left, upper-right).

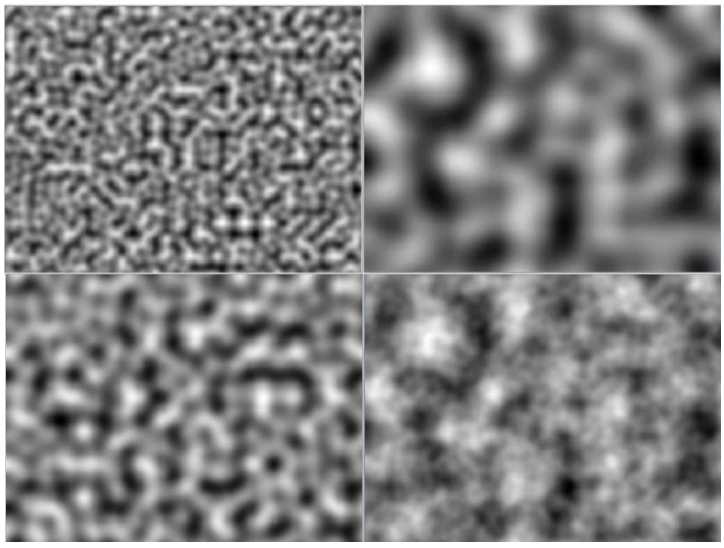

Figure 5-23. Pictures (640x480) of Perlin Noise with different grid size and its combination. Upper-left, grid size 20; Lower-left, grid size 40; Upper-right, grid size 80. Lower-right, the combination of the Perlin Noise generated with different grid size.

To generate a Perlin Noise in 2D picture, we split the picture into grids. The operation for every grid is the same. The noise value for (x, y) will depend on the grid points surrounding it, (x_0, y_0), (x_0, y_1), (x_1, y_0) and (x_1, y_1).

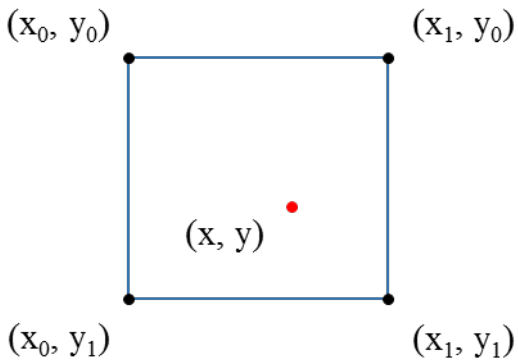

Figure 5-24. The 4 grid points bounding (x, y).

Now we need a function, (8), which takes each grid point and assigns it a pseudorandom gradient of length 1 in $\mathbf{R}^2$.

$$g(x_{grid}, y_{grid})=(g_x, g_y) \tag{11}$$

By pseudorandom, we mean that g has the appearance of randomness, but with the important consideration that it always returns the same gradient for the same grid point, every time it's calculated. It's also important that every direction has an equal chance of being picked.

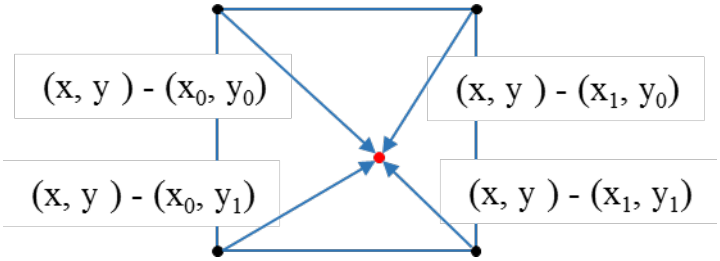

Figure 5-25. Vectors from the grid points to (x, y).

Then, the influence of each gradient can be calculated by performing a dot product of the gradient and the vector going from its associated grid point to (x, y).

$$s = g(x_0, y_0)\cdot((x, y)-(x_0, y_0)) \tag{12}$$

$$t = g(x_1, y_0)\cdot((x, y)-(x_1, y_0)) \tag{13}$$

$$u = g(x_0, y_1) \cdot ((x, y) - (x_0, y_1)) \tag{14}$$

$$v = g(x_1, y_1) \cdot ((x, y) - (x_1, y_1)) \tag{15}$$

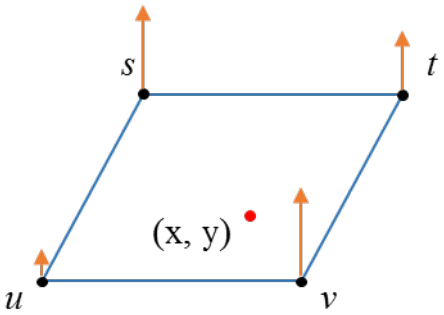

Figure 5-26. Influences from the grid points. It should be noted that these probably wouldn't be the actual values generated by the dot products of the vectors pictured above.

The value *s*, *t*, *u*, and *v* are the amount of influence on the (x, y) from the grid points, (x_0, y_0), (x_0, y_1), (x_1, y_0), (x_1, y_1), respectively.

Then a weighted average is apply to obtain the value of the noise at (x, y). That is, the value of the noise function should be influenced more by *t* than *s* when x is closer to x_1 than x_0 (as is the case in Figure 5-26). The weighted function is S_X and S_Y:

$$S_X(x, y) = 3(x - x_0)^2 - 2(x - x_0)^3 \tag{16}$$

$$S_Y(x, y) = 3(y - y_0)^2 - 2(y - y_0)^3 \tag{17}$$

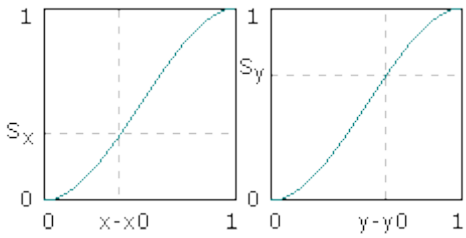

Figure 5-27. Visualization of the weighted function.

First, the averages are in x direction, which is the average between *s*, *t* and *u*, *v*.

$$\begin{aligned} a &= s + S_X(t - s) \\ b &= u + S_X(v - u) \end{aligned} \tag{18}$$

At last, the final noise value for (x, y):

$$\text{Noise value at (x,y)} = a + S_Y(b - a) \tag{19}$$

The noise value at every pixel can be generated in this manner. The Perlin Noise can be adjusted by the grid size, which will generate different results, Figure 5-28. The figure 5-29 shows a natural look of the combination of Perlin Noises.

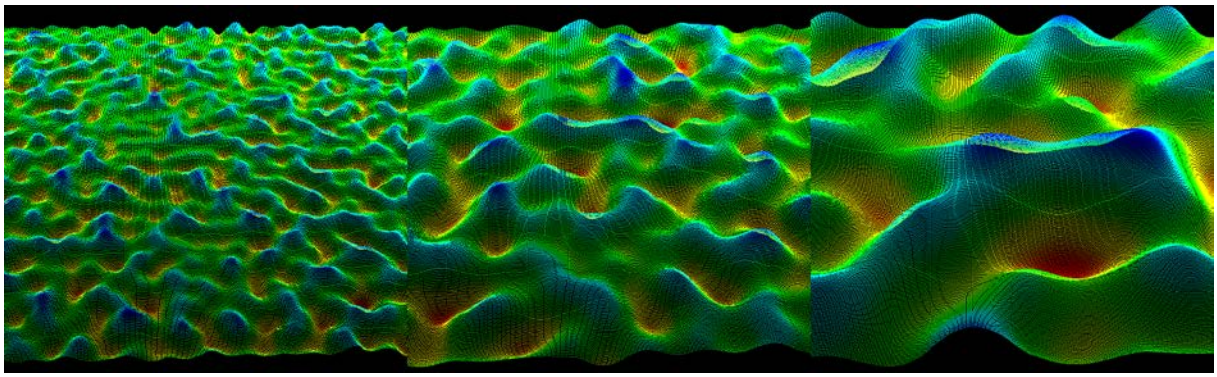

Figure 5-28. Perlin Noises with different grid size. From left to right, the grid size is 20, 40, and 80.

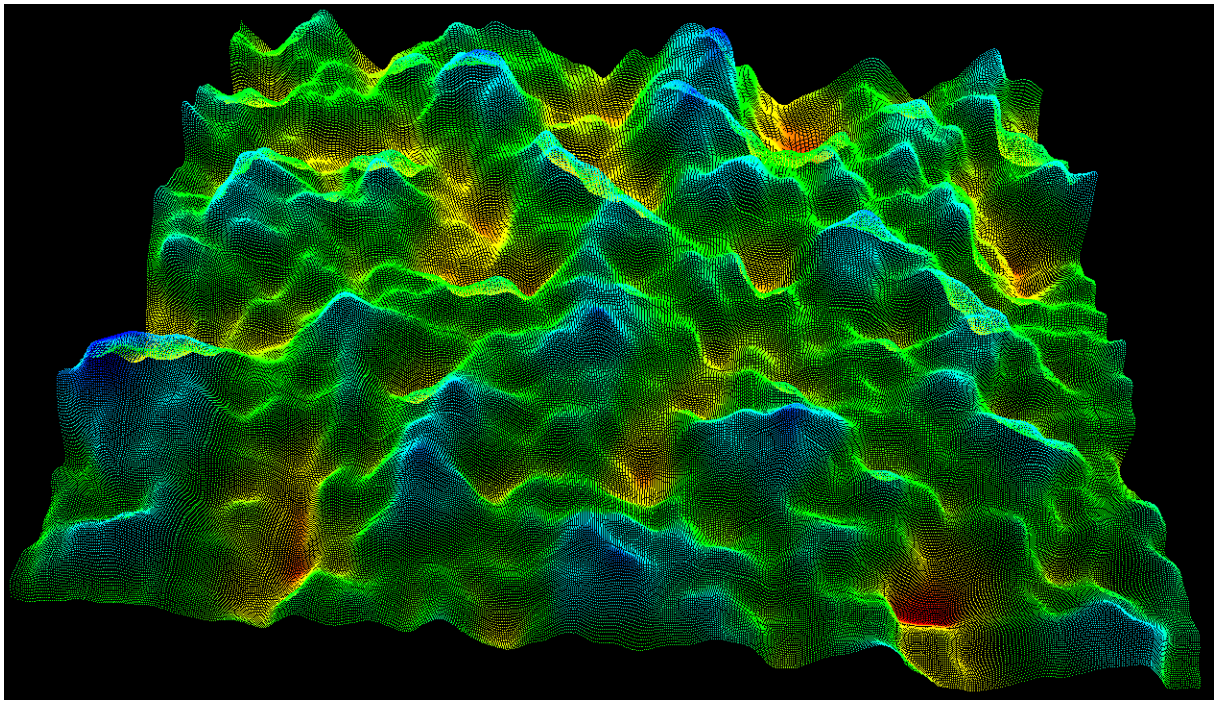

Figure 5-29. The combination of Perlin Noises with different grid size.

Chapter 6. Performance of the Evaluation Platform

In this chapter, we will show the performance of our evaluation platform. That is, how efficient it is to evaluate an algorithm with the simulator. The specifications of the PC used in this thesis are listed in the Table 6-1.

PC Specification	
CPU	Intel Core i7-2600 @ 3.8 GHz
GPU	GeForce GTX 650 Ti
Memory	8 GB

Table 6-1. Specifications of the PC used in this thesis.

6.1 Time to the Steady State

This section shows the time needed to simulate the random stacking of objects. The objects were first randomly placed above the box then the stacking simulations start. The time was counted from the beginning of the stacking simulations to the steady state of the objects. There were 500 trials conducted for each model (e.g. Cylinder, Brazo, and Wrench), and 9 or 18 of the objects were thrown in each test. The time for the random stacking is shown in Table 6-2.

Model	Number of Objects	Time to the Steady State (mean / standard deviation)
Cylinder	9	3.30 s / 1.54 s
	18	10.83 s / 5.04 s
Brazo	9	2.28 s / 0.93 s
	18	10.90 s / 5.77 s
Wrench	9	3.23 s / 1.61 s
	18	13.23 s / 5.60 s

Table 6-2. Time to the steady state of the random stacking process

6.2 Virtual Depth Sensor Capturing Time

The time needed to generate a 1280 x 960 synthetic depth image is shown in Table 6-3.

Stage	**Time**
Extract the depth image and occlusion effect	70.81 ms
Intensity saturation check	236.66 ms
Disturbing edge	196.24 ms
Confidence check	221.73 ms
Noise synthesis	6.52 ms
Blurring	188.19 ms
Transfer sensor data to algorithm	667.32 ms
Total	1596.66 ms

Table 6-3. Time to generate a 1280 x 960 synthetic depth image.

6.3 Computation Time

The total computation time for evaluating one stacking scene can be divided into 3 parts:

- Stacking time
- Sensor capturing time
- Pose estimation = Number of Estimated objects * time for estimating one object.

Then from Table 6-2 and Table 6-3, to evaluate one scene of 9 objects needs:

Computation time for one scene = 3.3 s + 1.6 s + 9 * 1 s = 13.9 s

Computation time for one object = 1.544 s

To evaluate one scene of 18 objects needs:

Computation time for one scene = 13.23 s + 1.6 s + 18 * 1 s = 32.83 s

Computation time for one object = 1.824 s

From the data above, the stacking time and sensor capturing time are longer than they take in the real world. But it is advised that other factors like human manipulation time haven't been taken into account. Since the evaluation progress is fully automatic, it has the advantage to run 24 hours a day without rest.

Chapter 7. Experiments

In this chapter, we will do experiments using our evaluation platform. First, we will evaluate the pose estimation algorithm [1]. Then we will use the evaluation platform to adjust the parameters of the algorithm to find the optimal parameter set.

7.1 Ideal Segmentation

A segmentation algorithm is needed to cluster the depth image generated from the depth sensor, so that the pose estimation algorithm can estimate the pose of each cluster. Since we only want to evaluate the pose estimation algorithm here, we prefer not to interrupt the result with a non-perfect segmentation algorithm. Therefore, we proposed an ideal segmentation that can cluster the depth image without any error (Figure 7-1). To obtain the ideal segmentation, we make use of Ogre3D to assign a unique color to every model in the scene. With this ideal segmentation, our evaluation platform is also capable of evaluating a segmentation algorithm.

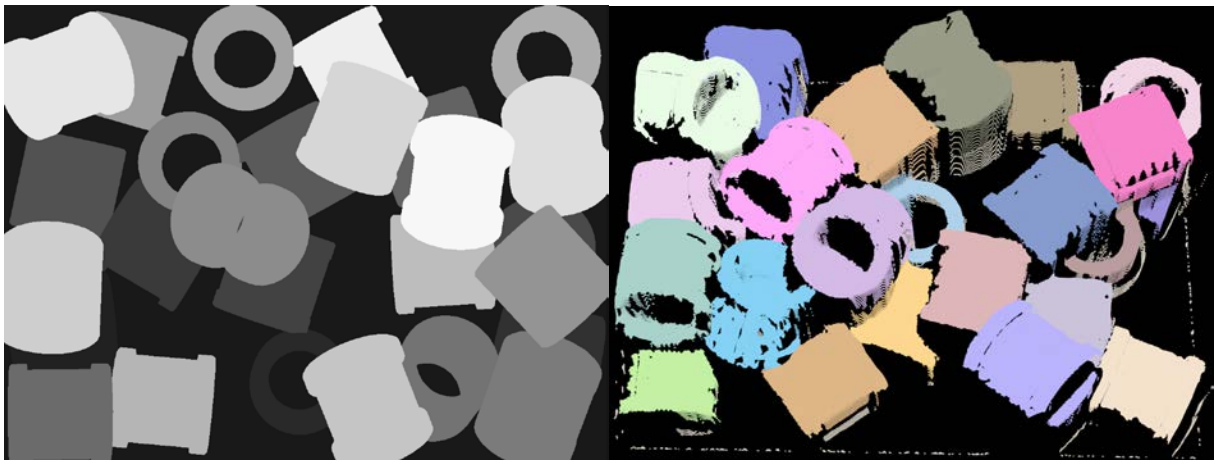

Figure 7-1. Ideal segmentation of a scene.

7.2 Evaluating Pose Estimation

In this section, we will use our evaluation platform to evaluate the pose estimation algorithm, not only to obtain the success rate of the algorithm, but also the properties of the algorithm.

7.2.1 Single Model Pose Estimation

The model used in this experiment is Cylinder, Brazo, and Wrench. A box is filled with 9 objects for each model, and an addition of 18 objects for Cylinder, since the cylinder is smaller. Then the virtual depth sensor generates the synthetic data and transfers the ideal cluster to the pose estimation algorithm, the pose estimation will then evaluate the clusters from top to down. And would restart the evaluating process once a failure estimation happened. The reason is that the consequence of a failure that happened in the real world is unknown. If the errors in translation and rotation were within 2.5 mm and 10 degree, we counted it as a success; otherwise, it was regarded as a failure. Over 1000 samples (stacks) were conducted for every experiment.

Figure 7-2, shows the evaluation result of Brazo. The success count before failure (SCBF) represents represent how many consecutive successes before a failure happened.

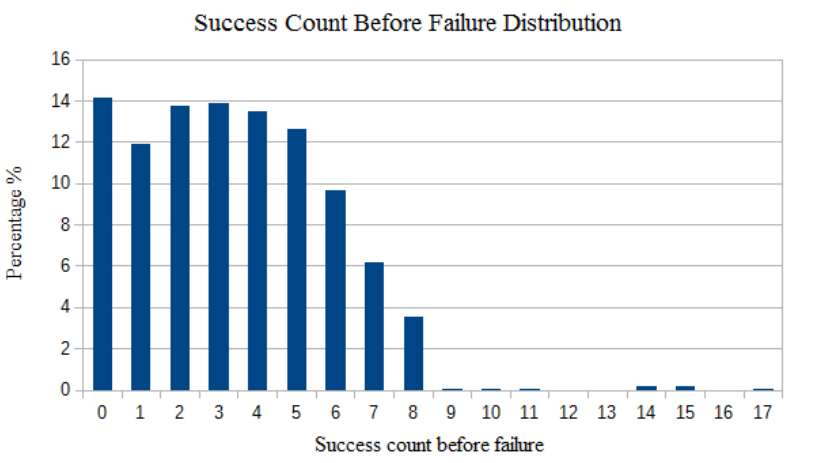

Figure 7-2. The evaluation result of Brazo.

Figure 7-3 shows the evaluation result of Wrench. In addition to the evaluation result, we found that the algorithm needs to go through several clusters before an output is given.

It should be noted that 18 objects of Brazo or Wrench have the similar result of 9 objects.

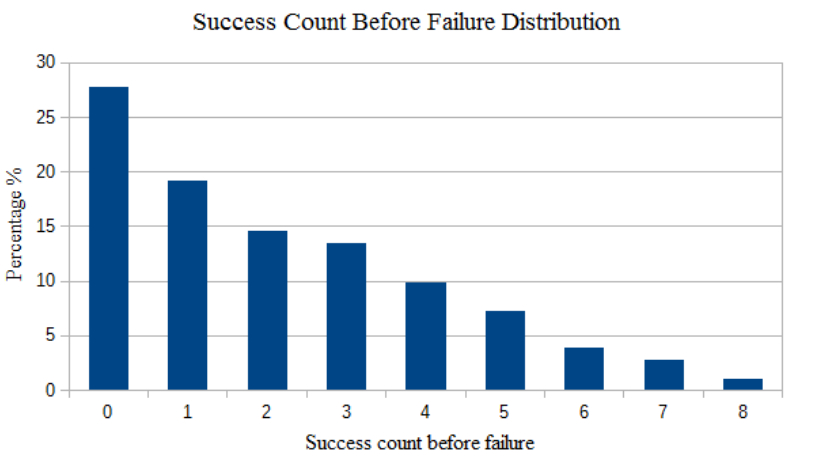

Figure 7-3. The evaluation result of Wrench

Figure 7-4, shows the evaluation result of 9 Cylinders.

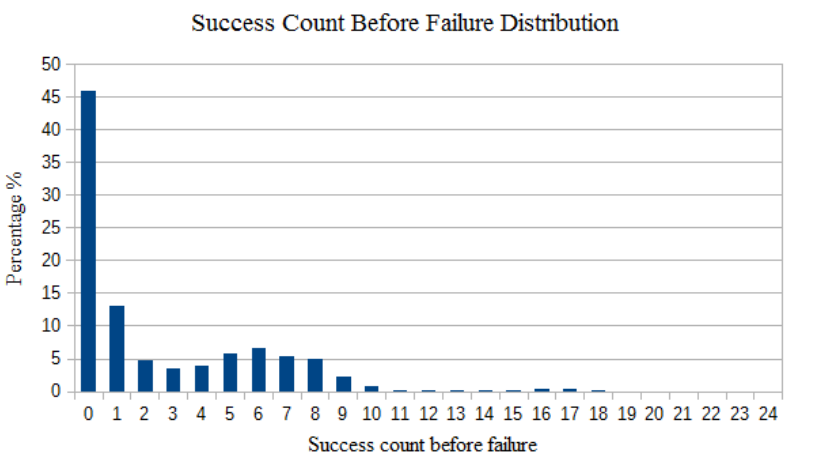

Figure 7-4. The evaluation result of 9 Cylinders

Figure 7-5, shows the evaluation result of 18 Cylinders.

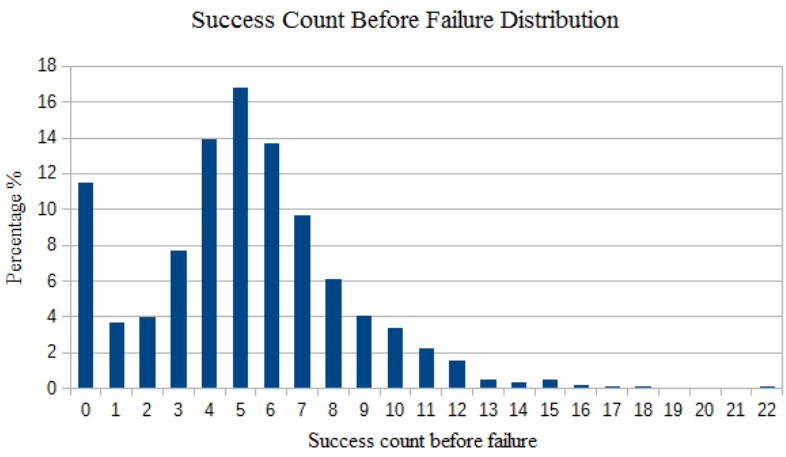

Figure 7-5. The evaluation result of 18 Cylinders

Interestingly, the SCBF distribution in Figure 7-5, shows a significant difference from the other results. The probability of n consecutive successes before a failure is:

$$P(n) = SuccessRate^{n} \times FailureRate^{1} \tag{20}$$

From the equation above, the SCBF distribution should be a monotonically decreasing function, like the distribution in Figure 7-3. However, the success rate may vary when the pose is different, which is observed in the evaluation log as indicated in Figure 7-6. The success rate is low when the axis of the cylinder is perpendicular or parallel to the view ray, which can also be seen in the result of 9 cylinders. The success rate of other poses is high and approximately 100 percent. According to Figure 7-6, there is no stacking in the experiment of 9 cylinders, while there is cylinder over another in the experiment of 18 cylinders. The lower layer of the cylinder have a low success rate, while the upper layer has approximately a 100 percent success rate, so that's why the distribution appears to have two peaks, the first peak shows the normal distribution of SCBF, while the second peak indicates the high success rate of the upper layer. This property is identical to the experiment result conducted in the real world. This finding, while preliminary, suggests that the evaluation platform is coincident with the real world.

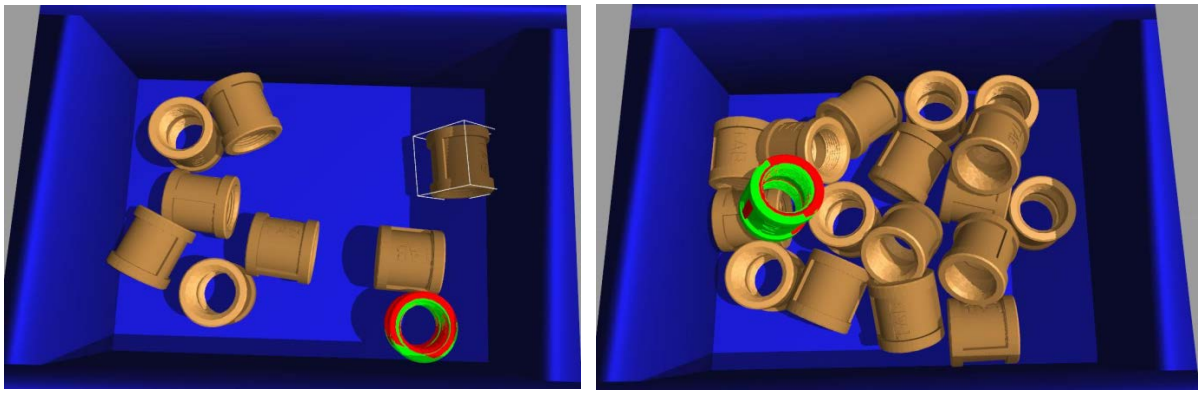

Figure 7-6. A difference success rate on a different pose, (Left) low success rate pose, (Right) poses other than the pose in the (Left) have high success rate. Green shows the estimated result from pose estimation, while red shows the ground truth.

7.2.2 Multiple Models Pose Estimation

In this experiment, we conducted a multiple models pose estimation with our evaluation platform (Figure 7-7). Two models, Brazo and Wrench, were used in this experiment, with 18 objects of them randomly stacked in the box. The pose estimation had a performance of 67.79% success rate in this experiment.

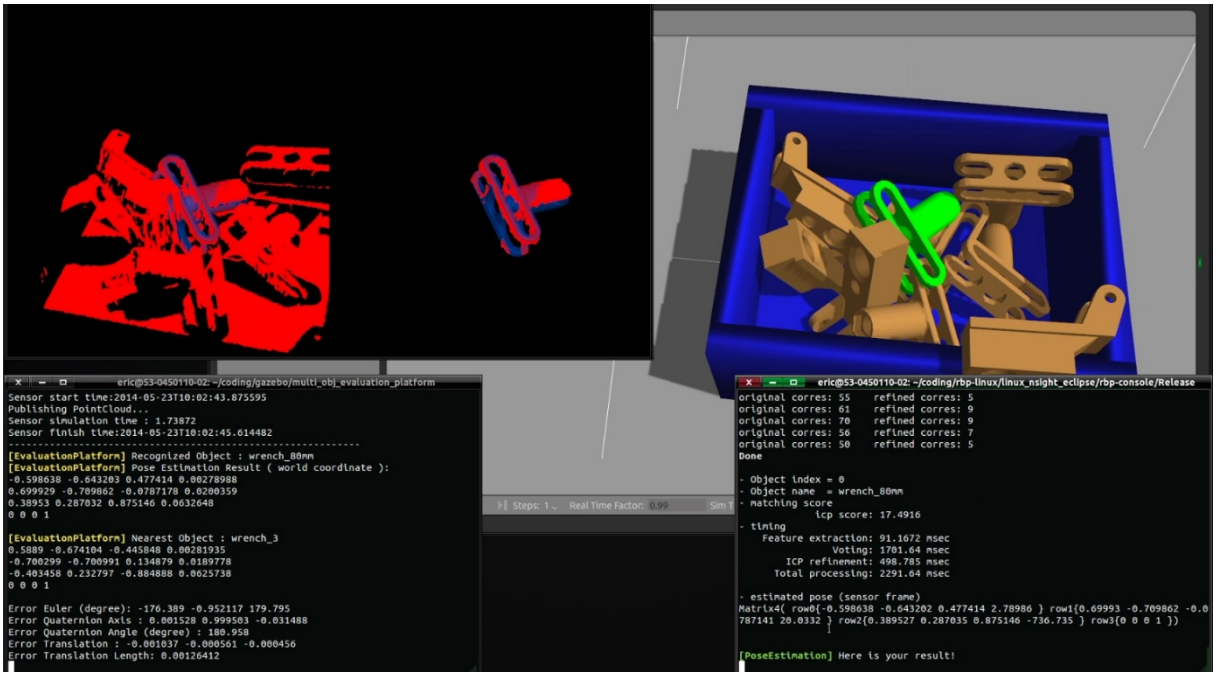

Figure 7-7. Evaluating multiple models pose estimation.

From this section, we show that the evaluation platform can generate a large amount of experiment data, and obtain the precise performance of the algorithm. Furthermore, the algorithm developer can learn the properties of their algorithm from the evaluation result and log, and improve the algorithm from it.

7.3 Optimizing Parameters Using the Evaluation Platform

In this section, we will show the feasibility to find the optimized parameters using the evaluation platform. The goal in this experiment is to find the best parameter of the pose estimation algorithm for the Socket model (Figure 7-8).

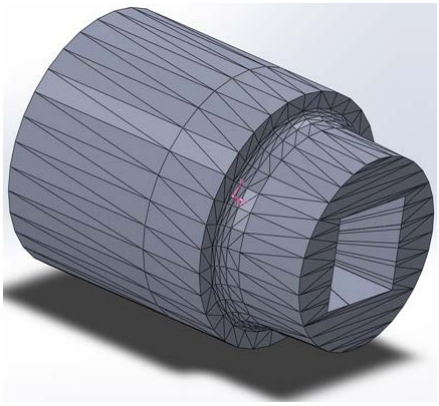

Figure 7-8. Socket model.

It's essential to find out how many trials (one trial means one stacking scene) are needed to sufficiently determine the success rate. We conducted an 18 Socket experiment 4 times with the parameter set used by the Cylinder, over 1400 trials were conducted for each experiment. From the experiment result, 560 trials were needed to determine the success rate within $\pm 0.4\%$ of the overall success rate of 1400 trials. Therefore, the experiments afterwards would have 560 trials for each.

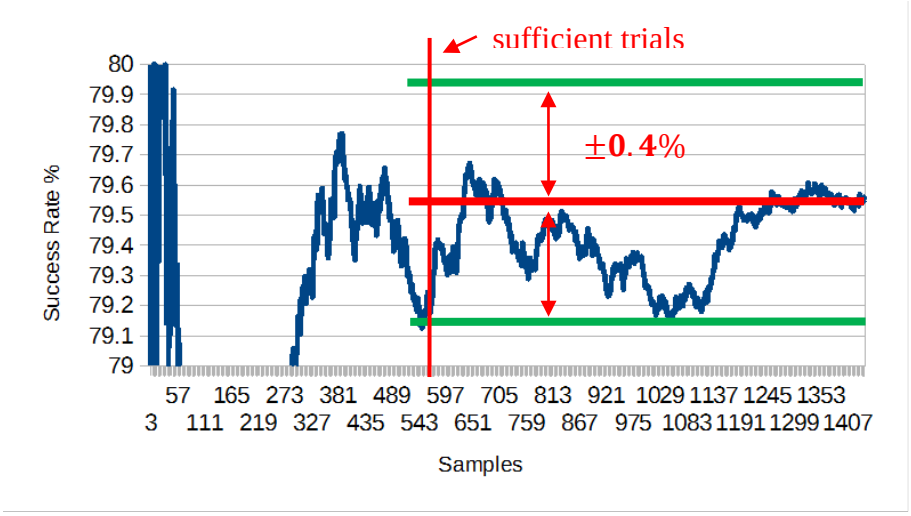

Figure 7-9. The trials needed to sufficiently determine the success rate.

We aim to adjust the parameters of the SHOT descriptor, search radius and similarity threshold. The experiment result of the search radius is listed in Table 7-1. Contrary to expectations, the result shows that two values have the best success rate. However, our key points are at least 5 mm apart, so the result of value 4 is not expected. Therefore, the optimal value for a search radius is 8, and the value will be used from now on.

Search radius (mm)	**4**	**6**	**8**	**10**	**12**	**15**
Success rate %	85.99	81.08	85.894	82.62	82.43	79.26

Table 7-1. Success rates using different search radiuses of the SHOT descriptor.

The similarity threshold is within the range of 0 to 1. We use a binary search method to find the optimal value for the similarity threshold. The result is shown in Table 7-2 and Figure 7-10.

Similarity Threshold	**0.1**	**0.3**	**0.4**	**0.5**	**0.55**	**0.575**	
Success rate %	84.71	85.894	85.83	85.9	85.08	85.99	
Similarity Threshold	**0.6**	**0.625**	**0.6375**	**0.65**	**0.675**	**0.7**	**0.9**
Success rate %	86.25	85.94	84.39	86.53	82.77	85.31	81.46

Table 7-2. Success rates using different similarity thresholds.

Figure 7-10. The scatter plot of the similarity threshold experiment.

Each experiment included 560 trials, and took up to 2.6 hours to complete.

The default parameters for the search radius and similarity threshold are 10 and 0.3, respectively. After this optimization experiment, we found the optimal value 8 and 0.65 for search radius and similarity threshold, which led to a 3.91% increase in the success rate (from 82.62% to 86.53%).

While the search radius is easier to adjust by humans, the similarity threshold have no prominent relationship with the success rate. It needs an extensive experiment to find the optimal value. Our evaluation platform is suitable with this kind of work. The optimization experiment demonstrated the advantages of automatical evaluation and extensive experiments. In this experiment, we conducted over 11,000 trials (75,000 poses estimated), and spent around 50 hours in three days. It is impossible to conduct such an experiment beforehand, because of the high cost of time and human power. The benefits from the simulator are tremendous, considering of conducting 50 hours experiments in three days while having free time to do other tasks simultaneously. On the other hand, spending 6 days, 8 hours in each, on the monotonous experiment. The benefits is magnify if some other advantages are also taken into account, like accelerating the simulation process using multiple PCs, time saved for setting up the environment, time saved for recording and post-processing of the experiment result, etc.

Chapter 8. Conclusion

In this thesis, we proposed a high fidelity simulator to evaluate the performance of the random bin picking algorithm. An architecture to automatically evaluate algorithms, with the environment in virtual space being configured to realistically simulate the environment in the real world. Certain modifications to the Gazebo were made, and a systematic simulation for mimicking the real sensor was documented. The virtual depth sensor is capable of simulating realistic sensor data, and is configurable to adapt to mimic other real sensors that have the same mechanism. At last, an extensive experiment was presented to show the efficiency of the simulator and the potential to help improve the algorithm considerably. However, the experiment conducted in the thesis is only one of the possible applications benefit from the evaluation platform. Other applications like offering standard benchmark, generating data set for experiment, generating training and testing data for machine learning, Big Data research, etc.

References

[1] Chin-Chia Wu, Yu-Feng Hsu, and Jwu-Sheng Hu, “Robust Pose Estimation of Randomly Stacked Rigid Objects through Virtual Space Modeling and Simulation,” Automation 2013, The 12th International Conference on Automation Technology, Nov. 2, 2013.

[2] Xianfang Sun , Paul L. Rosin , Ralph R. Martin , Frank C. Langbein, Noise analysis and synthesis for 3D laser depth scanners, Graphical Models, v.71 n.2, p.34-48, March, 2009.

[3] N. Koenig and A. Howard. Design and use paradigms for gazebo, an open-source multi-robot simulator. In In IEEE/RSJ International Conference on Intelligent Robots and Systems, pages 2149–2154, 2004.

[4] Open Dynamics Engine, http://www.ode.org/.

[5] Ogre3D: Object-Oriented Graphics Rendering Engine, http://www.ogre3d.org.

[6] O. Skotheim, J. T. Thielemann, A. r. Berge, and A. Sommerfelt, “Robust 3D object localization and pose estimation for random bin picking with the 3DMaMa algorithm,” Instrumentation, p. 75260E, 2010.

[7] J. Oh, C. Lee, and S. Lee, “Development of a structured-light sensor based bin-picking system using ICP algorithm,” in Control Automation and Systems (ICCAS), pp. 1673-1677, 2010.

[8] C. Choi, Y. Taguchi, O. Tuzel, M. Liu, and S. Ramalingam, “Voting-based pose estimation for robotic assembly using a 3D sensor,” in IEEE International Conference on Robotics and Automation, 2012, pp. 1724-1731.

[9] M.-Y. Liu, O. Tuzel, A. Veeraraghavan, Y. Taguchi, T. K. Marks, and R. Chellappa, “Fast object localization and pose estimation in heavy clutter for robotic bin picking,” The International Journal of Robotics Research, vol. 31, pp. 951-973, May 2012.

[10] K. Boehnke, “Object localization in range data for robotic bin picking,” in IEEE International Conference on Automation Science and Engineering (CASE), pp. 572-577, IEEE, 2007.

[11] Upstill, Steve. The RenderMan Companion: A Programmer's Guide to Realistic Computer Graphics. Addison-Wesley. ISBN 0-201-50868-0.

[12] Boost, http://www.boost.org/.

[13] Google Protocol Buffers, https://developers.google.com/protocol-buffers/.

[14] Qt, http://qt-project.org/.

[15] K. Perlin, “An Image Synthesizer,” in Computer Graphics (Proceedings of ACM SIGGRAPH 85), 24. 3.

[16] R. B. Rusu, N. Blodow, and M. Beetz, “Fast point feature histograms (FPFH) for 3D registration,” in IEEE international Conference on Robotics and Automation, 2009.

[17] P. Scovanner, S. Ali, and M. Shah, “A 3-dimensional SIFT descriptor and its application to action recognition,” in Proceedings of the 15th ACM International Conference on Multimedia (ACM MM), (New York, NY, USA), pp. 357-360, 2007.

[18] A. Frome, D. Huber, R. Kolluri, T. Bulow, and J. Malik, “Recognizing objects in range data using regional point descriptors,” in European Conference on Computer Vision (ECCV), pp. 224-237, 2004.

[19] Ramesh Raskar, Kar-Han Tan, Rogerio Feris, Jingyi Yu, and Matthew Turk, “Non-photorealistic camera: depth edge detection and stylized rendering using multi-flash imageing,” in ACM Transactions on Graphics, Volume 23 Issue 3, pp. 679-688, 2004.