Generating 3D Point Cloud Data using 2D LiDAR Sensor with ROS

Archit Jain, Mihir Trivedi, Deepak Jaiswal

Department of Research and Development, Kalyani Strategic Management Services, Pune

***Abstract* — This paper describes an algorithm that performs 3D scanning with the help of 2D Light Detection And Ranging (LiDAR) sensor using Robot Operating System (ROS). Using a 2D 360 degree LiDAR we get a LaserScan data in 2D plane. To convert this 2D LiDAR into 3D scanned point cloud data, an additional physical third axis (Z axis) is used for collecting laserscan data in third dimension. With the help of ROS interface, an algorithm is written to convert these laserscan data into single point cloud data without any human intervention. The LaserScan data and point cloud data can be visualized in ROS supported applications such as Rviz (3D Visualization Tool) and Gazebo (3D Robotic Simulator). The generated point cloud data can be stored as a RosBag file as well as in point cloud data format extension (.pcd) which can further be utilized in meshing softwares such as MeshLab for generating 3D CAD models and exporting STL files of scanned objects.**

Keywords *— 3D scanning, LiDAR, Mapping, Point-Cloud, ROS*

I. INTRODUCTION

LiDAR technology has been used in numerous applications recently. The LiDAR system design has significantly advanced over time, leading to a design with very low SWaP (size, weight, and power) requirements. Due to LiDAR's light weight and energy efficiency, its use in aerial and mobile platforms has grown, making the once difficult tasks of mapping and obstacle avoidance possible. LiDAR technology has been growing rapidly with the hardware miniaturization. LiDAR has a wide range of possible uses, including space management in buildings, remote sensing, urban planning, and autonomous vehicles. LiDAR, which stands for Light Detection and Ranging. In order to measure the distance between a target object and the light path or between an object and its velocity, LiDAR technology uses sensors that highlight light. Accurate information on object velocity and distance can be obtained from the amount of time it takes for the illuminated light to return to its source after reflecting off the target object. The accuracy is made possible by light traveling at a constant speed through air. The number of laser beams that can be focussed on a surface and the technique of surface scanning are used to categorize LiDAR technology systems. LiDAR systems are available in 1D, 2D, and 3D. In order to gather accurate data on the X and Y axes, 2D LiDAR is made to produce a single beam of light in the direction of the target object on a horizontal plane. Typically, the LiDAR sensor spins in order to gather enough data on both the X and Y coordinates. When compared to their 3D equivalents, the size of a 2D LiDAR sensor makes it easier to port. The most effective 2D LiDAR sensors are for range and detection tasks. But how does 3D LiDAR work; unlike its 2D equivalent, it makes use of special types of sensors that rotate at 360 degrees while sending out several beams of light in the direction of a target's vertical plane to collect the target's X, Y, and Z coordinates. For mapping and scanning of landscapes, 3D technology is appropriate. A 3D LiDAR system is made to emit many light beams in order to gather more comprehensive object data than a 2D LiDAR system, which spins as it directs only one beam of light towards a surface. Because 2D LiDAR intends to collect object data only on its X and Y axes, the beam of light is only shot along the object's horizontal plane. Conversely, 3D LiDAR focuses its beam along its target's vertical plane to capture 3-dimensional data on the X, Y, and Z axes. This mode of scanning provides more detailed object information. Due to the extensive horizontal data it can collect, 2D LiDAR is well suited for tasks that call for a lot of ranging and detection. On the other hand, because of the amount of data that this model can collect along an object's Y-axis, 3D LiDAR systems are ideally suited for mapping and scanning landscapes. Despite the fact that both LiDAR technologies have a variety of industrial uses, 3D LiDAR is more expensive than 2D LiDAR. That is in part due to its level of accuracy and its capacity to gather precise object information. In this paper we report on the use of 2D LiDAR. It is a scanning LiDAR that provides 360 degree sensing capabilities in all three reference frames. There will be an end effector of a robotic arm that has a LiDAR sensor installed on it, which spins and detects the distance to objects from it continually from various angles. The data gathered from this process will be used to create a map of the environment's 2D plane. A single point cloud will also be created using the same laser scan data (assembled). With the help of a6 degrees of freedom (DoF) robotic arm, the LiDAR can cover the entire field of view of an object. The third Z axis transformations will be made on robotic arm feedback with help of the Robot Operating System (ROS) framework to transform each laser scan z axis component channel in assembling point cloud data. Using an open source 3D robotic simulator 'Gazebo ROS' we can simulate the robotic arm along with the LiDAR scanning in a virtual environment created by ourselves. To visualize and assemble laser scans into point cloud data, we will use 'Rviz' which is a 3D visualization tool for ROS. Here we can simulate and visualize the object being scanned into 3D single point cloud data (PCD) live.

II. Literature Review

Over the past ten years, range sensors like lidars have grown in popularity as essential parts of a significant portion of autonomous systems. The current decade has seen the most research activity and interest in the field of easily accessible 3D depth sensing sensors. [10] presents a 3D lidar that is both lightweight and modest in size. Although its application is restricted by a small vertical scan window (40 degrees). It is reported in [11] that a constantly spinning lidar placed on a micro aerial vehicle (MAV) is capable of measuring range in three dimensions. A rotating SICK lidar is employed in other applications [12]. [13] describes obtaining 3D perception for computing vertical plant profile and tree structure. There have been discussions about using MEMS-based low resolution 3D lidar for agricultural robots. According to ROAMS in [15] a 2D lidar is installed on top of a pan, tilt, and rolling setup is used to show a 3D range measurement system. A bottleneck in getting a 3D scan of nearly 360° FoV in [15] with a hardware setup. One of the key pieces of research on creating 3D mapping from 2D lidar was given by Raymond et al. in [16]. Rolling the sensor for 3D vision is demonstrated to be more advantageous, particularly for the Redback robot that could climb stairs, in addition to describing the possible rotation about elementary coordinate axes. A low-cost, commercially available 3D range sensor that performs pitching was presented by Morales et al. in [17]. Although the sensor perception results in an office environment are shown, the quality of the reconstructed depth map is not as good as it could be since different-shaped and-sized objects are not present in the environment. [18] generates a 3D depth image of the surroundings for the application to detect two-way traffic and crossings using a rolling SICK LMS 2D lidar.

III. Methodology/Experimental

A. Components/Flowchart

Workflow to scan any 3D object in robots workspace:

1. The LiDAR sensor is mounted on the end-effector of the robot arm for scanning.
2. The robot arm will guide the sensor around the object to scan any 3D object in its workspace.
3. The LiDAR sensor will generate 2D Laser Scan data points on each scan interval.
4. Further using ROS and algorithm built; point cloud data will be assembled using a set of these multiple laserscan data points on each transform generated by the robot arm.
5. And the assembled single point cloud data stored/saved can be viewed in any meshing open software (ex: Meshlab) and convert it to standard model format STL.

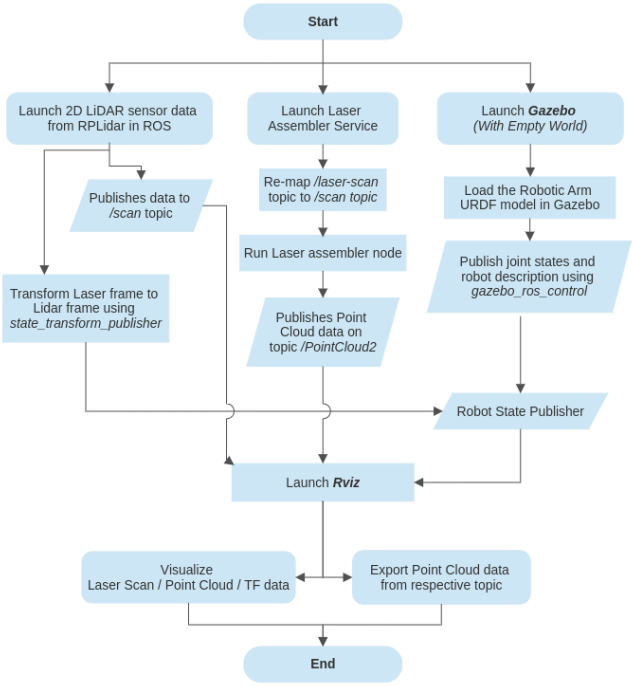

Figure 1.1: Flowchart of 3D Object Scanning using ROS tools (Gazebo, Rviz, Laser Assembler)

For computing hardware we need some edge devices and equipment to operate our robot using ROS. So, here's a list of equipment needed to perform the scanning operation.

1. Raspberry Pi 4 Model B
2. RPLidar A1M8
3. A computer with an internet connection and the ability to flash your microSD card. Here we'll be using a laptop.
4. High-performance microSD card: 32GB minimum
5. MicroSD to SD adapter
6. We will need a monitor, keyboard, and mouse (at least for the initial setup)
7. ROS applications use a lot of compute resources and the heat sink may not be enough for the heat generated. Consider adding a cooling fan to your Raspberry Pi 4.

We have used a 2D LiDAR (Light Detection And Ranging) sensor for scanning objects. RPLIDAR A1M8 is a renowned 2D LiDAR sensor available in the market and is based on laser triangulation ranging principle and uses high-speed vision acquisition and processing hardware developed by Slamtec. The system measures distance data more than 8000 times per second. It is a low cost 360 degree 2D laser scanner (LIDAR) solution. The system can perform 360 degree scan within 12-meter range (6-meter range of A1M8-R4 and the bellowing models). The produced 2D point cloud data can be used in mapping, localization and object/environment modeling. It can be configured up to 10 hz maximum. As it is based on a laser triangulation measurement system, it can work excellently in all kinds of

indoor and outdoor environments without direct sunlight exposure.

Figure 1.2: RPLidar A1M8 with connectors to integrate with Linux systems

Table 1.1: Technical specifications of 2D Laser scanner

Parameter	***RPLidar A1-M8***
Maximum measurement range(m)	6
Measurement error (mm)	±50
Scanning angle (deg)	360
Angular resolution (deg)	<=1
Scanning time (ms/cycle)	180
Measurement resolution (mm)	<0.5
Data interface and transfer rate	USB 2.0
Ethernet portSupply voltage (VDC)	5±5%
Current consumption (mA)	350
Weight (kg)	0.2
External dimensions (WxLxH in mm)	90 x 70 x 60

As the sensor has open source libraries to synchronize its laser scan data with ROS, we have integrated the 2D scanned data in Rviz to visualize the detected objects.
In our experiment, “Rviz” is adopted as the 3D visualization tool. With Rviz, one can see what is going on (e.g. robot position) or set goal pose for the robot arm from a remote place. We used “Gazebo” for simulation purposes, this software can communicate with ROS. So, we have tested our system in simulation to observe work-ability and safety.

B. Synthesis/Algorithm/Design/Method

The setup contains a robotic arm having 6 joints along with an 2D LiDAR mounted on the end effector of this robot arm. The design of the robot arm can be done in any Computer Aided Design (CAD) software such as CATIA, SolidWorks, Fusion 360, etc. After a basic CAD model is made of a 6 joint robotic arm and LiDAR sensor, make/write the Unified Robotics Description Format (URDF) files to use in ROS. Assemble individual links with revolute joints and add transmissions to each joint for controlling the position of the robotic arm. Set joint state publishers for each joint so that to use those joint variables for position control of respective joints in Gazebo (3D Robotic Simulator) and Rviz (3D Visualization Tool). Attach a LiDAR sensor to the end effector of the robot arm and add a Gazebo sensor plugin to use laser rays in simulation softwares.

B 1.1. Robot arm design:

The robot arm consists of 6 joints having 1 DoF each in all considered as 6 DoF robot arm. The robot arm was designed for the purpose of scanning. So, it has capabilities of scanning in a greater workspace as compared to load capacity. Also, this robot arm model depicts the motion of LiDAR sensors for a robust scanning process. One can use its own robot arm designed as per requirements.
There’s a base revolute joint of the robot arm holding the power to support the entire weight of other arm links and end-effector sensor. The base of the robot must be strong enough to hold the position of other linkages, so there’s a revolute joint in the Z axis providing 360 viewing capability to the robot's field of view.
The second link of the robot arm provides vertical height adjustment features in the Z axis. It has a revolute joint in the X axis. This arm has a greater length as compared to other linkages so as to reach greater height accordingly.
There’s another similar joint in the X axis but in the opposite direction of the second link. It is basically to compensate for the robot's height adjustments to keep the end effector head position at 90 degrees from the base whenever required. This joint has a short link (third link) so that another link ahead of this may have another DoF to produce more workspace distribution.
A fourth link is present for rotation in the Z axis but with greater link length similar to the second link. So, as to compensate for the height adjustments w.r.t (with respect to) the second link arrangement. But as it has a rotation along the Z axis, the robot has capabilities to bend forward and have a rotational trajectory parallel to the ground plane which increases the robots capacity to scan in each layer around the object.
An X axis direction joint is present on top of the fourth link to provide the end effector to scan objects in angular motion keeping the base joint at a fixed position. It basically refers to a tilt scan joint scanning from bottom to top from the same position where it started. This fifth link is again a shorter length link so as to avoid center of mass issue at

greater height from ground plane and keep the robot dynamics stable.
Finally, a sixth link is present on the topmost position which is also called the end-effector holder. This holder is basically used to mount any sensor on it to make the robot use for a specific purpose (here the sensor is LiDAR for scanning purpose). It has a Z axis revolute joint for additional movement.
In this way, the robot arm is designed having 6 joints with 6 DoF features for scanning any 3D object in its workspace.

B 1.2. ROS installation and LiDAR setup:
The most recent and final distribution of ROS 1 is called ROS Noetic. Considering that more than half of all robots utilize ROS, choosing to adopt it is a wise decision. Noetic will be sponsored through May 2025, so we have about five years left.

The much more recent Raspberry Pi model, the Raspberry Pi 4 with 8 GB RAM , provides the best performance yet in comparison. It functions virtually as a desktop PC when two 4K monitors are connected. If we still have older Raspberry Pis, we can use this guide for them even if Raspbian has recently changed its name to Raspberry Pi OS. When searching online, we can substitute "Debian" or "Debian Buster" for "Raspberry Pi OS" and the majority of the material will still be relevant to your situation even though the partial Debian name (Raspbian) has been removed as a result of the rebranding. Obviously other edge computing devices are present in the market to deal with such heavy computation, but considering a low cost and feasible compact control systems, Raspberry Pi 4 (Rpi) was used for this project scope.

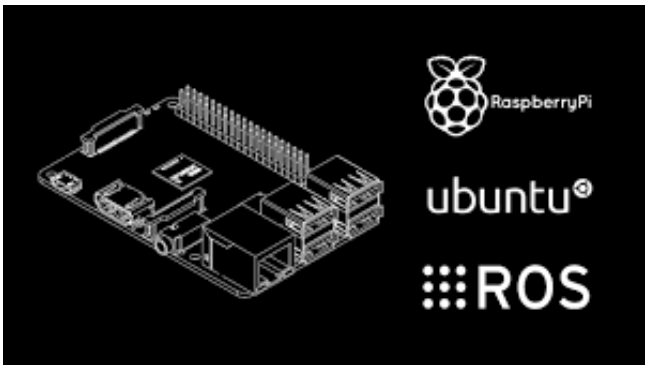

Figure 2.1: Raspberry Pi 4 Model B+ 8GB RAM supported Ubuntu/ROS

As you already know, ROS Noetic is mainly developed for Ubuntu 20.04, so Ubuntu is the recommended Linux OS for installation. You should also first check if your Raspberry Pi OS is Debian buster, because Noetic only officially supports Buster (Debian 10). To do this, run “lsb_release -sc” in the terminal and you should be able to see “buster” in the output.
To install ubuntu/debian on Rpi, follow the steps mentioned below:

- Download and install Etcher from the website.
- Click Select Image in Etcher and choose the OS Image (Download it from internet ubuntu official website).
- Attach your SD card to the computer. Etcher will select it automatically.
- Click Flash! to write the image file to the SD card.
- When done, remove the SD card, insert it into your Raspberry Pi.

We probably already know that we need a few things to get started with Raspberry Pi such as a mouse, keyboard, HDMI cable etc.

- Plug in a mouse and a keyboard.
- Connect the HDMI cable.
- Insert your MicroSD card.
- Plug in your Ethernet cable, if you’re using one.

When everything else is set up, power it on your Raspberry Pi by plugging the power cable. After powering on your Raspberry Pi, wait for the boot process to complete and then finish with the setup process as any other OS system has.

To make sure all dependencies are up to date, run the following command “sudo apt-get update”. And if we want to get the latest versions of software we have installed already, run:

“sudo apt-get upgrade”

Create some memory space to stop running out Rpi shutdown when it is out of memory due to heavy computing. Use the “fallocate” program to create a swap file with the following command in the terminal “sudo fallocate -l 2G /swapfile” allowing 2 GB space.
Then mark the file as:

“swap space” by typing: “sudo mkswap /swapfile”

Ask the system to start using our new swap file on each reboot/restart using this command:

“sudo swapon /swapfile”

The above command will only last until the next reboot. In order to make it permanent, add it to the /etc/fstab file:

“echo '/swapfile none swap sw 0 0' | sudo tee -a /etc/fstab”

Install ROS Desktop on Rpi which has ROS packaging, build, and communication libraries and tools like RQT and Rviz using this command:

“sudo apt install ros-noetic-desktop”

Then run “sudo apt-get install build-essential”. Make sure to source the setup file in bash rc file by typing:

“echo "source /opt/ros/noetic/setup.bash" >> ~/.bashrc”

Run “roscore” and check if it is installed successfully.

Connect your RPLiDAR to Raspberry Pi 4 Model B+ using Micro USB Cable. Flashing green light indicates normal activity of the sensor. Then provide permissions for the system to read data from the sensor connected:

"sudo chmod 666 /dev/ttyUSB0"

(here "USB0" is the port name, we can verify it by typing "ls /dev/tty*").
After creating a catkin root and source folder for creating new ROS packages in your system, install LiDAR ROS packages to support RPlidar features in ROS. Clone the ROS node for LiDAR in the catkin workspace inside the "src" directory using the command:

"sudo git clone https://github.com/Slamtec/rplidar_ros.git"

Launch the laser scan node provided by this package using a launch file:

"roslaunch rplidar_ros rplidar.launch"

This will publish the laser scan data from RPlidar to a topic named "/scan". To visualize it in the Rviz tool, open Rviz using:

"rosrun rviz rviz"

and add a laser scan topic from "Add" >> "By topic" >> "/scan - LaserScan". Now as the tool doesn't know the position of LiDAR in the world/map frame of ROS, we need to define a transformation (currently be it a static transform) between the world frame and lidar frame. As the RPlidar published laser scan data to "/scan" topic from "laser" frame, we will link the "laser" frame with "world" from using the following command in another terminal:

"rosrun tf static_transform_publisher 0 0 0 0 0 0 world laser 100"

Here the first three "0" signifies the positional transform "tf" i.e., xyz position is 0 units from the world coordinate frame. And other three "0" signifies the rotational transform "tf" i.e., the "roll pitch yaw" from "world" to "laser" frame. Then comes a "fixed frame id" i.e., the "world" frame preceding with "child frame id" i.e., the "laser" frame with a transform publish rate of "100". Now, we can see the 2D Laser scan data in Rviz with small red points /dots as per data provided by the sensor.

B 1.3. URDF/Joints/Transmissions/Plugins in ROS:
As the transformations made earlier for testing LiDAR sensor data in the Rviz tool, we noticed that we have transformed the "laser" frame to "world" frame using a static transform publisher, which means there will be no motion between the two frames (a type of fixed joint). But if the LiDAR is in a static position we can only generate point cloud data in a 2D plane. So, to add another third axis movement, we will need to set up joints and transmissions for the robot arm to provide additional DoF for LiDAR sensors in 3D space to scan objects and generate/assemble point cloud data in 3D. For this, we had setup joints to each 6 DoF of arm linkages discussed above in the "Robot Arm Design" section.

URDF (Unified Robot Description Format) is a file format for specifying the geometry and organization of robots in ROS. With the help of the URDF model of a robot arm with a sensor created in ROS, we can use model dynamics for simulation purposes in Rviz and Gazebo. URDF is an extension of XML (Extensible Markup Language) language wherein we can define Plugins for Gazebo, Joints for links, Transmission to every joint, Material, Robot Dynamic properties and many more.

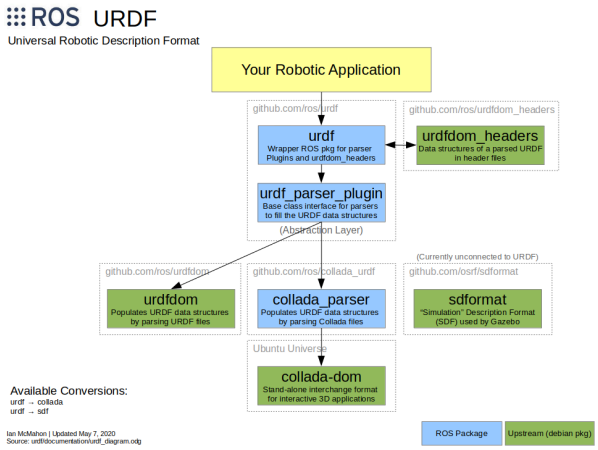

Figure 2.2: XML Specifications for Robot Model/Sensors/Scenes for URDF

So, firstly create a link named "world" having none of the inertial properties because ROS insists us to provide any first link to the ground plane without any inertial parameters. Then comes the "base_link" of the robot which is fixed to the ground plane. To fix the robot base to the world link at center we need a joint in URDF to place them together with these fixed mates.

The "joint" element in URDF has two attributes:
name - Specifying a unique name to the respective joint.
type - Specifying the type of join to be used in between two links. Types can be one of the following:

1. revolute — a hinge joint that rotates along the axis and has a limited range specified by the upper and lower limits.
2. continuous — a continuous hinge joint that rotates around the axis and has no upper and lower limits.
3. prismatic — a sliding joint that slides along the axis, and has a limited range specified by the upper and lower limits.
4. fixed — this is not really a joint because it cannot move. All degrees of freedom are locked. This type of joint does not require the <axis>, <calibration>, <dynamics>, <limits> or <safety_controller>.

The joint element has following elements:

<origin> - This is the transform from the parent link (world link) to the child link (base_link). The joint is located at the origin of the child link.

1. xyz - Represents the x, y, z offset. All positions are specified in meters.
2. rpy - Represents the rotation around fixed axis: first roll around x, then pitch around y and finally yaw around z. All angles are specified in radians.

<parent> (here world) - The name of the link that is the parent of this link in the robot tree structure.
<child> (here base_link) - The name of the link that is the child link.
<axis> (defaults to (1,0,0)) - The joint axis specified in the joint frame. This is the axis of rotation for revolute joints, the axis of translation for prismatic joints, and the surface normal for planar joints. The axis is specified in the joint frame of reference. Fixed and floating joints do not use the axis field.

1. xyz - Represents the (x, y, z) components of a vector. The vector should be normalized.

<dynamics> - An element specifying physical properties of the joint. These values are used to specify modeling properties of the joint, particularly useful for simulation.

1. damping (defaults to 0) - The physical damping value of the joint (in newton-seconds per metre [N·s/m] for prismatic joints, in newton-meter-seconds per radian [N·m·s/rad] for revolute joints).
2. friction (defaults to 0) - The physical static friction value of the joint (in newtons [N] for prismatic joints, in newton-meters [N·m] for revolute joints).

<limit> (required only for revolute and prismatic joint)

1. lower (defaults to 0) - An attribute specifying the lower joint limit (in radians for revolute joints, in meters for prismatic joints). Omit if the joint is continuous.
2. upper (defaults to 0) - An attribute specifying the upper joint limit (in radians for revolute joints, in meters for prismatic joints). Omit if the joint is continuous.
3. effort - An attribute for enforcing the maximum joint effort (|applied effort| < |effort|).
4. velocity - An attribute for enforcing the maximum joint velocity (in radians per second [rad/s] for revolute joints, in meters per second [m/s] for prismatic joints).
5. floating - this joint allows motion for all 6 degrees of freedom.
6. planar - this joint allows motion in a plane perpendicular to the axis.

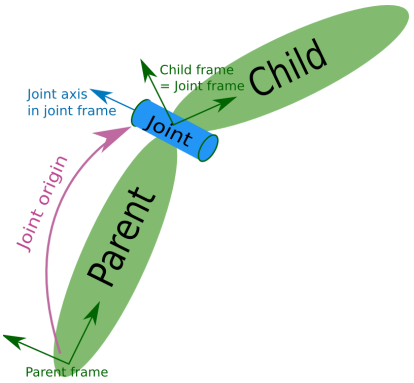

Figure 2.3: Joint elements description in URDF models

So, for the first joint between the "world" frame and "base_link" frame, we will apply a "fixed" joint and place the robot on ground having its base at the center of the world coordinate system. Then we will apply accordingly and we will apply further joints to respective links as per requirement. Their nomenclature and specifications are specified below:

Frames: (Link reference => Link name)
Link World => world
Link 0 => base_link
Link 1 => base_support
Link 2 => base_elbow
Link 3 => elbow_wrist_1
Link 4 => wrist_1_2
Link 5 => wrist_2_3
Link 6 => wrist_3_end_effector
Link Lidar => lidar

Joints: (Parent link, Child link => Joint name => Joint type)
world, base_link => base_footprint_joint => fixed
base_link, base_support => bj_joint => revolute
base_support, base_elbow => sj_joint => revolute
base_elbow, elbow_wrist_1 => ej_joint => revolute
elbow_wrist_1, wrist_1_2 => w1j_joint => revolute
wrist_1_2, wrist_2_3 => w2j_joint => revolute
wrist_2_3, wrist_3_end_effector => w3j_joint => revolute
wrist_3_end_effector, lidar => lidar_joint => fixed

After applying all related joints to respective links and sensors, apply transmission to respective joints so that we can control robots position, effort and velocity using ROS tools as well as while interfacing hardware. The transmission element is an extension to the URDF robot description model that is used to describe the relationship between an actuator and a joint. This allows one to model concepts such as gear ratios and parallel linkages. A transmission transforms efforts/flow variables such that their product - power - remains constant. Multiple actuators may be linked to multiple joints through complex transmission.

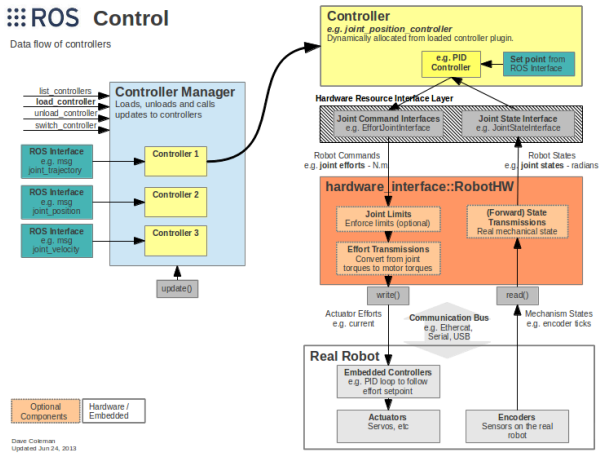

Figure 2.4: Hardware interface controls for Joint Interface

The transmission element has one attribute:
name: Specifying a unique name to the respective transmission.
The transmission has the following elements:
<type> (one occurrence) - Specifies the transmission type.
<joint> (one or more occurrences) - A joint the transmission is connected to. The joint is specified by its name attribute, and the following sub-elements:
<hardwareInterface> (one or more occurrences) - Specifies a supported joint-space hardware interface. Note that the value of this tag should be EffortJointInterface when this transmission is loaded in Gazebo and hardware_interface/EffortJointInterface when this transmission is loaded in RobotHW (Robot hardware).
<actuator> (one or more occurrences) - An actuator the transmission is connected to. The actuator is specified by its name attribute, and the following sub-elements:
<mechanicalReduction> (optional) - Specifies a mechanical reduction at the joint/actuator transmission. This tag may not be needed for all transmissions.
<hardwareInterface> (one or more occurrences) - Specifies a supported joint-space hardware interface.

So, to operate the robotic arm to desired location, we have to control the position of joints. For this we have used EffortJointInterface/effort_controllers. The controller outputs the desired effort (force/torque) to the Hardware Interface/Joint by having any one of these three types of inputs:

1. joint_position_controller - Receives a position input and sends an effort output, using a PID controller.
2. joint_velocity_controller - Receives a velocity input and sends an effort output, using a PID controller.
3. joint_effort_controller - Receives an effort input and sends an effort output, just transferring the input with the forward_command_controller.

We have used joint_position_controller to set/input the desired position of the respective joint.

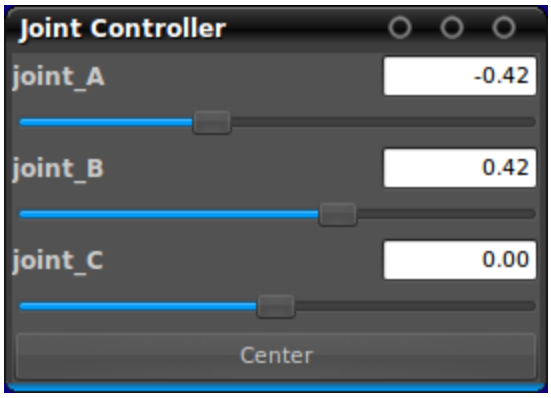

Figure 2.5: Trackbar joint state controller from robot state publisher

Now these joints are ready for transformation. When we load this model in the "Rviz" tool, we can add a "TF" topic and see the tf frames initialized for each joint and can operate each joint using the "joint_state_publisher_gui" trackbar. But to use the same in gazebo, we need to add a gazebo plugin in the URDF file so as to link all transforms and controllers with robots in "Gazebo" and "Rviz" together. Simulating a robot's controllers in Gazebo can be accomplished using "ros_control" and a simple Gazebo plugin adapter. To use ros_control with our robot, we need to add transmission elements to your URDF which we have already done before. A Gazebo plugin needs to be added to our URDF that actually parses the "transmission" tags and loads the appropriate "hardware interfaces" and "controller manager"

```
<gazebo>
        <plugin name = "gazebo_ros_control" filename =
        "libgazebo_ros_control.so">
        </plugin>
</gazebo>
```

By default, without a *<robotSimType>* tag, "gazebo_ros_control" will attempt to get all of the information it needs to interface with a ros_control - based controller out of the URDF. The default behavior provides the following ros_control interfaces:

- hardware_interface::JointStateInterface
- hardware_interface::EffortJointInterface

So, the PID gains and controller settings must be saved in a yaml file that gets loaded to the param server via the roslaunch file. The config folder of your robot package should have a "robot_control.yaml" file having "joint_state_controller" and "joint_position_controller" for respective individual joints mentioned in the "transmission" tag. Then in the launch file, make sure to load all the specified controllers in the ".yaml" file using a "controller spawner" node with a "controller_manager" package.

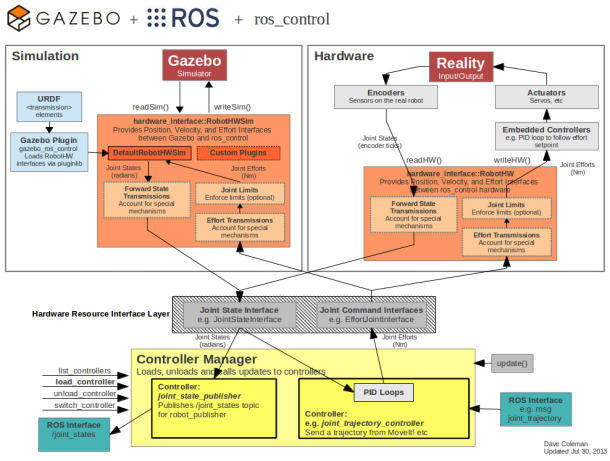

Figure 2.6: ROS Control Plugin for Gazebo with Hardware Interface

Also, if we want to see the LiDAR sensor rays in simulation in Gazebo, then add a laser plugin in the same URDF by referencing the "lidar" link. The sensor type is "ray" and set the parameter "visualize" to "True" if we want to see the rays in gazebo else "False" to scan objects but keeping the rays invisible as this plugin requires some computing power. We can set the number of samples, resolution, minimum angle, maximum angle, minimum range, maximum range, noise type and many other properties according to our required sensor, as this is a custom plugin in ROS.

B 1.4. Laser Assembler:

Laser rangefinder sensors (such the RPlidar A1M8) typically output a stream of scans, with each scan consisting of a set of range readings for objects the sensor has identified (in polar coordinates). To obtain a 3D perspective of the environment, many robotic systems, including PR2's tilting laser platform, articulate a laser rangefinder. A larger 3D Cartesian coordinate (XYZ) point cloud is created by the nodes in the laser assembler package by listening to streams of scans. We have interfaced with the laser_assembler package via ROS node:

- laser_scan_assembler: Assembles a stream of sensor_msgs/LaserScan messages into point clouds.

The laser_scan_assembler subscribes to sensor_msgs/LaserScan messages on the scan topic. The Projector and Transformer process these scans by projecting the scan into cartesian space and then transforming it into the fixed frame. A sensor_msgs/PointCloud is produced as a result, which can then be put to the rolling buffer. On service calls, clouds in the rolling buffer.

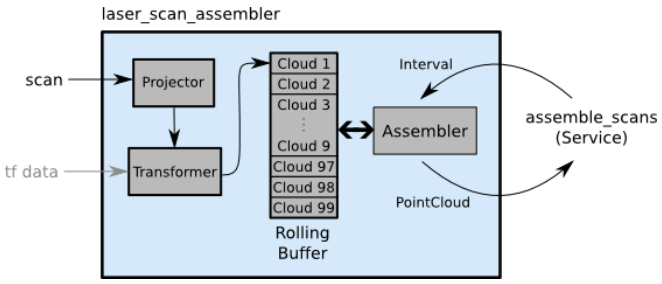

Figure 2.7: Laser Assembler service node data transfer

"assemble_scan" service - An assembler searches its rolling buffer for clouds that occur within the desired period when it receives an assemble_scan request (begin to end). The larger cloud created from these smaller ones is then sent to the caller in the service response in the frame determined by the fixed_frame option. This is a non-blocking process, and if no scans are received within the requested time frame, an empty cloud will be returned. As determined by the laser_geometry::LaserProjection library, the final cloud will have channels with names like intensities, index, distances, and stamps.

The Laser Scan Assembler - The laser_scan_assembler accumulates laser scans by listening to the appropriate topic and accumulating messages in a ring buffer of a specific size. When the assemble_scans service is called, the contents of the current buffer that fall between two times are converted into a single cloud and returned. We have remapped our robot's scanned topic from "/robot/scan" to "/scan" as this assembler searches for "/scan" topic to read laser scan data. Each single scan is converted into the fixed frame when it arrives, and no additional transforms are done to the data when the cloud is published. Therefore, we have chosen a frame that isn't moving i.e., the "world" frame.

C. Pseudo Code/ Testing

To run any node, before we need to initialize ROS Master. "roscore" is a collection of nodes and programs that are pre-requisites of a ROS-based system. We must have a roscore running in order for ROS nodes to communicate. It is launched using the "roscore" command.

We have created a "main.launch" file which first launches an empty world followed by setting arguments such as robot description, robot URDF file, robot name, robot home position (x y z). Then in a common group (namespace = robot_arm) we first launch the joint state controllers (robot_control.yaml files) which will publish the joint variables to "joint_state_controllers". Followed by setting parameters for robot description to get loaded and synced with Rviz "RobotModel". Now, we load our robot arm into the Gazebo environment with the help of the "gazebo_ros" package by spawning the model into the empty world created before with the home position arguments described above. Then to load the joint state variables to respective

joint topics, we load the "spawner" using the "controller_manager" package. And finally the "robot_state_publisher" node to publish joint states that can be utilized by ROS different tools and nodes.
Launching this "main.launch" file using "roslaunch robot_arm_scanner main.launch" will load the model into gazebo and set it ready for scanning.

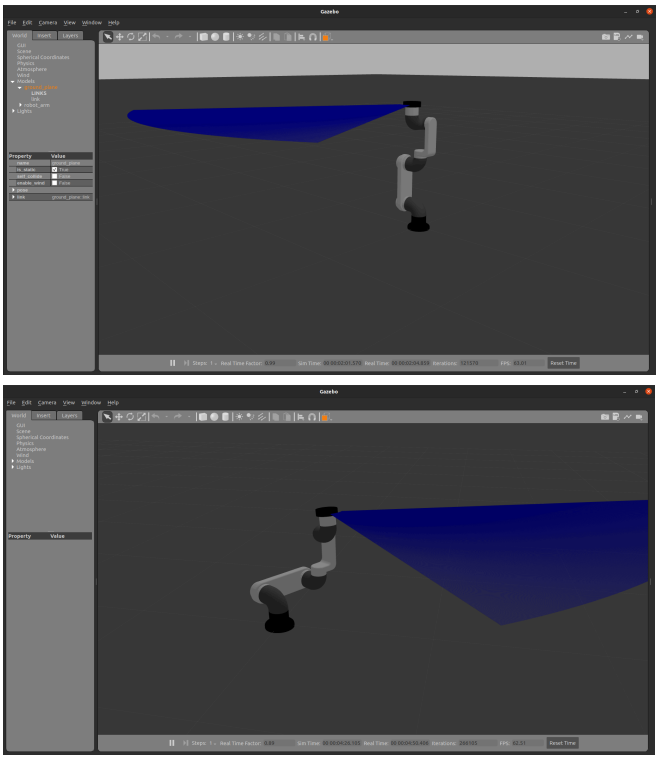

Figure 3.1: Loaded Robotic Arm URDF model in Gazebo Empty World

The "rostopic list" command lists out all of the active topics running by ROS Master. Here we can notice the "/scan" topic published by the sensor, "/joint_states" topic published by the "robot_state_publisher", individual joint state command topic by "joint_state_controller", "/tf" and "/tf_static" topics of transforms we wrote before.
You can notice that the joint state publisher have four topics for each joint out of which we are mainly interested in:

1. /command: This is to set/provide the desired position we want the joint to reach using the controller interface we defined in the transmission. Input a position value in radians and the joint will handle the effort by applying PID values set by us. (We had tuned the PID values for smoother and quicker operations).
2. /state: This gives position feedback of the joint at each instance of time (updates according to joint state publisher publish rate).

Using these topics we created a script, a node which subscribes to these nodes as feedback and publishes required goal positions to respective joint topics and perform the scanning operation easily.

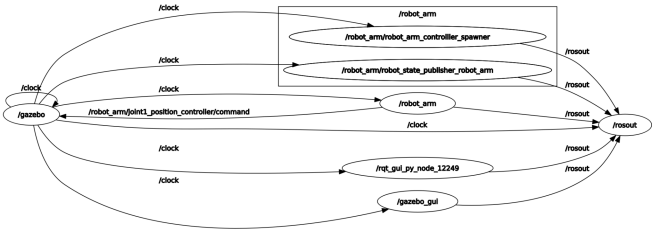

Figure 3.2: Rqt-Graph to visualize communication between all active nodes

Now as the robot is ready to scan the object we need to visualize the detected laser scan data and assemble them into point clouds. So, for this we used the Rviz tool which can help us visualize laser scan and point cloud data together. We launch Rviz using the "rosrun rviz rviz" command. First we load the "RobotModel" from the "By Display Type" option as soon as the Rviz is launched. And then set the fixed frame to "world" frame as we are performing all statics transformations w.r.t this frame. Now to visualize the Laser scan data, we add the "/Scan => Laserscan" topic in the "By Topic" option inside the "Add" tool. To see any object detected, insert a block in front of the laser in the gazebo and then you can see the reflected scanned laser data in the Rviz interface.

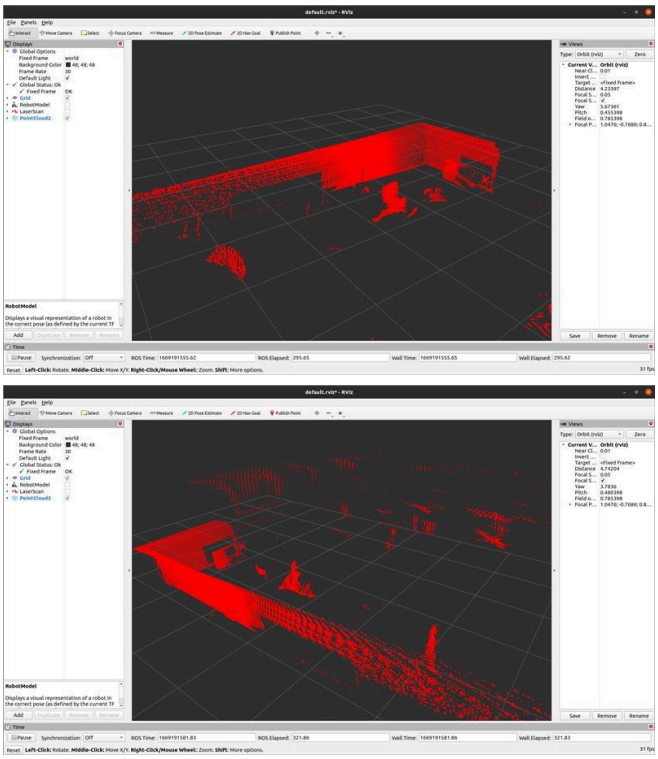

Figure 3.3: 3D environment scanning using RPLidar A1M8

The laser scan data which we get in this step are 2D data points. So, if we want to scan a 3D object then we need an additional third axis which can be in the format of 3D point cloud data. A Point Cloud is a 3D visualization made up of thousands or even millions of georeferenced points.

Therefore, we had assembled all laserscan data of scanning operation into a single 3D point cloud using the "laser_assembler" service provided by ROS. We wrote a script which collects laser scan data in a time interval specified with buffer from "/scan" topic (which is remapped from "lidar" frame to "scan" frame in the launch file before) and assembles them "PointCloud2" format so that we can visualize the data messages published on respective point cloud topic in Rviz. Similar to how we added the "Laserscan" topic from the "Add" tool, we have to add "PointCloud2" topic to visualize the object scanned by a robotic arm in gazebo using a script to control robot joints.

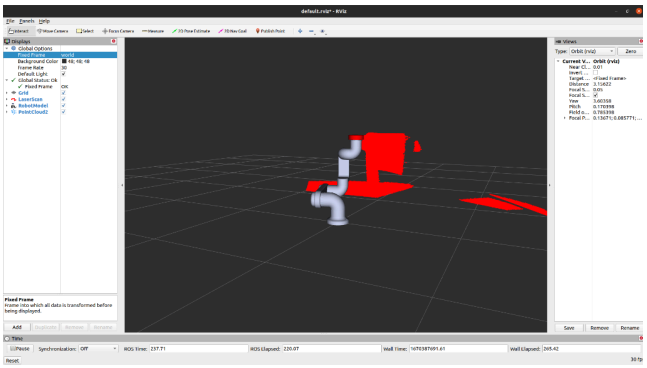

Figure 3.4: Simulation of Scanning a Trash-Can model in Rviz/Gazebo

IV. Results and Discussions

On the designed robotic arm - end effector, we apply the 3D laser scanner that has been simulated. The sensors are used for a variety of navigational functions, including mapping, localization, and drivability evaluation. Every time a custom 3D laser scanner is built, there are slight distortions. However, the system must be aware of its precise alignment in order to obtain accurate 3d models. Consequently, a calibration is required. Although discussing this would go beyond the scope of this essay, you can study publications on calibration for both spinning 2D and 3D laser scanners. Additionally, adding color information to the point cloud to obtain an intensity map of objects describing data at various surface levels may be valuable for some applications. The distortion of the 2D laser scans during a continuous rotation of the 2D laser scanner is another crucial point of which one should be aware. Combining alignment, the robotic arm's rotational speed, and the measurement frequency of the 2D laser scanner may result in a straightforward mathematical answer.

A rotating 2D laser scanner has several advantages over currently available ready-to-use 3D laser scanners, including a lower cost, a larger vertical field of view (FOV), and a higher vertical resolution achieved by using a lot more layers (Velodyne HDL-64/Hokuyo UTM-30LX). While 3D laser scanners that can be purchased use several laser beam transceivers simultaneously, 2D laserscanners rotate a 1D laser beam transceiver to produce this amount of layers. The ready-to-use systems can spin significantly faster with numerous scans per second by applying multiple distance measurements at once. A rotating 2D scanner may take several seconds depending on the horizontal resolution, however greater update rates are achievable if a lesser resolution is suitable, for example, for relocalization in a certain map or viewing a specific region of the world. Therefore, a system with a spinning 2D laser scanner should not move throughout the scanning process; otherwise, if there is no precise robot motion estimation, the resulting 3D scans are deformed. Therefore, although spinning 2D scanners are best suited for perception tasks requiring high precision and measurement density like traversability evaluation or item recognition, purchasable 3D scanners are better suited for extremely dynamic scenarios like metropolitan street scenes. They can be used in the manufacturing sector to identify flaws, scan items, and visualize them for analysis.

V. Limitations

We intend to look at the impact of odometry inaccuracies for short-range robot motion. Being a robotic arm, it has some restrictions regarding vibrational and translational inaccuracies while attempting to span intricate curves in three-dimensional space. We are therefore developing a smooth transition method for motion using appropriate PID adjusted settings as well as a mechanical damping factor to prevent overshoot at high speeds in order to avoid this issue. Another drawback that requires additional study is how the sensor's nodding tendency interacts with dynamic obstructions.

VI. Future Scope

The laser scan data that is currently combined into 3D point clouds using the laser assembler ROS service only offers point clouds on a topic rather than producing an STL model of it. In order to stay within the scope of this project, we will develop an application that can generate 3D CAD models as well, support a variety of CAD file formats and be coupled with meshing tools like Meshlab.

VII. Conclusion

The primary goal of this project is to create a low-cost 2D

LiDAR-based 3D object scanning system. The proposed system could successfully execute a 2D and 3D scan, hence the goal was accomplished by producing 3D point clouds from 2D laser scan data. But there are lots of areas where the system could be strengthened for use in the future. For instance, a better robotic arm might be used to speed up the scan and decrease vibration during the scanning test in order to decrease noisy data and boost scanning speed, respectively. The success of this research demonstrates the dependability and precision of a cheap LiDAR sensor for doing an object 3D scan. This experiment demonstrates the enormous potential of inexpensive LiDAR sensors for additional robotic applications, specifically for the visual system.

VIII. Acknowledgment

This work was supported by the Research and Development Team at the Kalyani Strategic Management Services by Kalyani Group, Pune, India.

IX. References

[1] A. Hornung, M. Phillips, E. Gil Jones, M. Bennewitz, M. Likhachev, and S. Chitta, "Navigation in three-dimensional cluttered environments for mobile manipulation," in 2012 IEEE International Conference on Robotics and Automation, pp. 423–429, Saint Paul, MN, USA, May 2012. [Google Scholar]

[2] M. Schadler, J. Stückler, and S. Behnke, "Rough terrain 3D mapping and navigation using a continuously rotating 2D laser scanner," KI - Künstliche Intelligenz, vol. 28, No. 2, pp. 93–99, 2014. [Google Scholar]

[3] R. Zlot and M. Bosse, "Efficient large-scale three-dimensional mobile mapping for underground mines," Journal of Field Robotics, vol. 31, no. 5, pp. 758–779, 2014. [Google Scholar]

[4] S. Vidas, P. Moghadam, and M. Bosse, "3D thermal mapping of building interiors using an RGB-D and thermal camera," in 2013 IEEE International Conference on Robotics and Automation, pp. 2311–2318, Karlsruhe, Germany, May 2013. [Google Scholar]

[5] P. Henry, M. Krainin, E. Herbst, X. Ren, and D. Fox, "RGB-D mapping: using Kinect-style depth cameras for dense 3D modeling of indoor environments," The International Journal of Robotics Research, vol. 31, no. 5, pp. 647–663, 2012. [Google Scholar]

[6] F. Pomerleau, F. Colas, and R. Siegwart, "A review of point cloud registration algorithms for mobile robotics," Foundations and Trends in Robotics, vol. 4, no. 1, pp. 1–104, 2015. [Google Scholar]

[7] J. L. Martínez, J. Morales, A. J. Reina, A. Mandow, A. Pequeño-Boter, and A. García-Cerezo, "Construction and calibration of a low-cost 3D laser scanner with 360° field of view for mobile robots," in 2015 IEEE International Conference on Industrial Technology (ICIT), pp. 149–154, Seville, Spain, March 2015. [Google Scholar]

[8] J. Morales, J. Martínez, A. Mandow, A. Reina, A. Pequeño-Boter, and A. García-Cerezo, "Boresight calibration of construction misalignments for 3D scanners built with a 2D laser rangefinder rotating on its optical center," Sensors, vol. 14, no. 11, pp. 20025–20040, 2014. [Google Scholar]

[9] H. Surmann, A. N¨uchter, and J. Hertzberg. An autonomous mobile robot with a 3d laser range finder for 3d exploration and digitalization of indoor environments. Robotics and Autonomous Systems, 2003

[10] Katsumi Kimoto et al., "Development of small size 3D LIDAR," in: Proceedings of the IEEE International Conference on Robotics and Automation, ICRA, 2014.

[11] J. S. David Droeschel and S. Behnke, "Local multi-resolution representation for 6D motion estimation and mapping with a continuously rotating 3D laser scanner," in: Proceedings of the IEEE International Conference on Robotics and Automation, ICRA, 2014.

[12] D. Van der Zande et al., "Influence of measurement set-up of ground based LIDAR for derivation of tree structure," Agricultural and Forest Meteorology, Elsevier, vol. 141, no. 2, pp. 147–160, 2006.

[13] J. R. Rosell et al., "Obtaining the three-dimensional structure of tree orchards from remote 2D terrestrial LIDAR scanning," Agricultural and Forest Meteorology, Elsevier, no. 149, pp. 1505–1515, 2009.

[14] U. Weiss, and P. Biber, "Plant detection and mapping for agricultural robots using a 3D LIDAR sensor," Robotics and Autonomous Systems, no. 59, pp. 265–273, 2011.

[15] H. M. Biruk A. Gebre and K. Pochiraju, "Remotely operated and autonomous mapping system (ROAMS)," in: Proceedings of IEEE International Conference on Technologies for Practical Robot Applications, TePRA, 2009.

[16] R. Sheh et al., "A low-cost, compact, lightweight 3D range sensor," in: Proceedings of Australian Conference on Robotics and Automation, Auckland, New Zealand, 2006.

[17] J. Morales et al., "Design and development of a fast and precise low-cost 3D laser rangefinder," in: Proceedings of IEEE International Conference on Mechatronics (ICM), 2011.

[18] R. Sheh et al., "A 3D laser scanner system for autonomous vehicle navigation," in: Proceedings of International Conference on Advanced Robotics, ICAR, 2009.