DreamVu

PAL Person & Object Detection Starter Kit

API Documentation

To contact DreamVu, please drop an email at:
For general queries - info@dreamvu.com
For technical support - support@dreamvu.com

DreamVu Inc.
3675 Market Street, Suite 200, Philadelphia, PA, 19104
www.dreamvu.com

Revision History

Revision	Author	Date	Status and Description
1	Piyush	25/02/2020	• Added DetectionMode camera property • Updated Explorer Section & Images

Table of Contents

1. Introduction and Background

DreamVu's PAL is the world's first 360° stereo and depth sensor capable of providing real time situational awareness to autonomous machines. Its innovative binocular mirrored optics enables the capture of 360° environment in stereo. You can capture the following information in real time using the APIs provided in this document:

- Stereo panoramas (left and right views of a stereo system)
- Disparity Image
- Depth-map
- Point cloud
- People Detection data

The objective of this document is to serve as an all in one guide for the end user. This document explains the setup procedure, includes tutorials on how to use the APIs, provides an API reference and lists the known issues & troubleshooting steps.

Note: Unless stated otherwise, all the relative paths are with respect to the directory where PAL_SDK is extracted.

Figure 1.1: Front view of PAL [USB]

Figure 1.2: Front view of PAL[POE]

2. Quick Start

The PAL Starter Kit comes with the preinstalled SDK placed on the Desktop of the Nvidia Jetson NX board. Please connect an HDMI monitor, a keyboard and a mouse to start the evaluation.

2.1 Explorer Application

Upon unboxing the starter kit, you can quickly get started with exploring the features of the camera by using the Explorer Application provided on the Desktop. This application allows you to preview the live feed of the stereo panoramas, depth map and Point Cloud, record videos and access or change the resolution and other camera parameters . You can also use this application to check whether the camera is connected properly and is being detected by the API backend.

- Plug in PAL into a USB 3.0 port.
- Launch Explorer Application from Desktop.
- The Explorer application launches with default sensor properties for generic viewing conditions. These properties can be adjusted in the application to suit the scene.

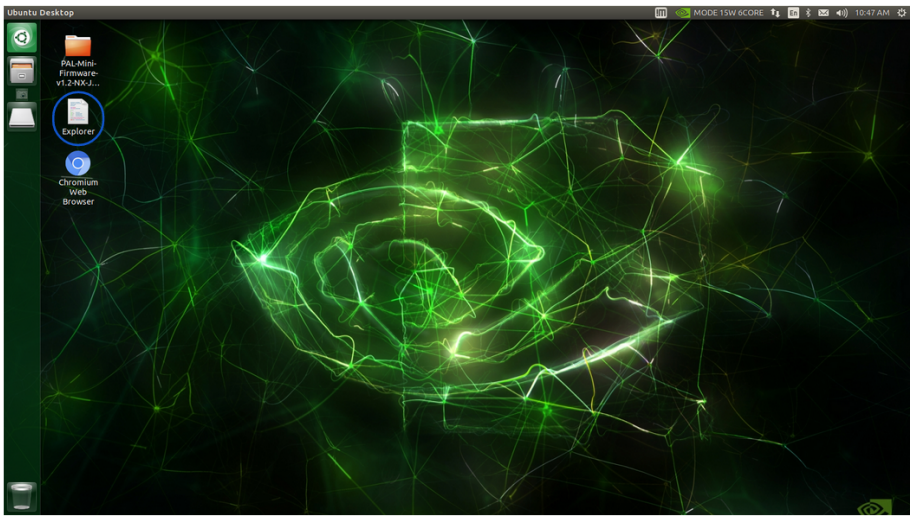


Figure 2.1: Desktop view of Kit

The Explorer helps users to quickly explore the features and the different modes of operation that are supported by the PAL camera. When the camera is connected and the Explorer application is opened you should see an output as shown below, displaying the left, right and depth panoramas.

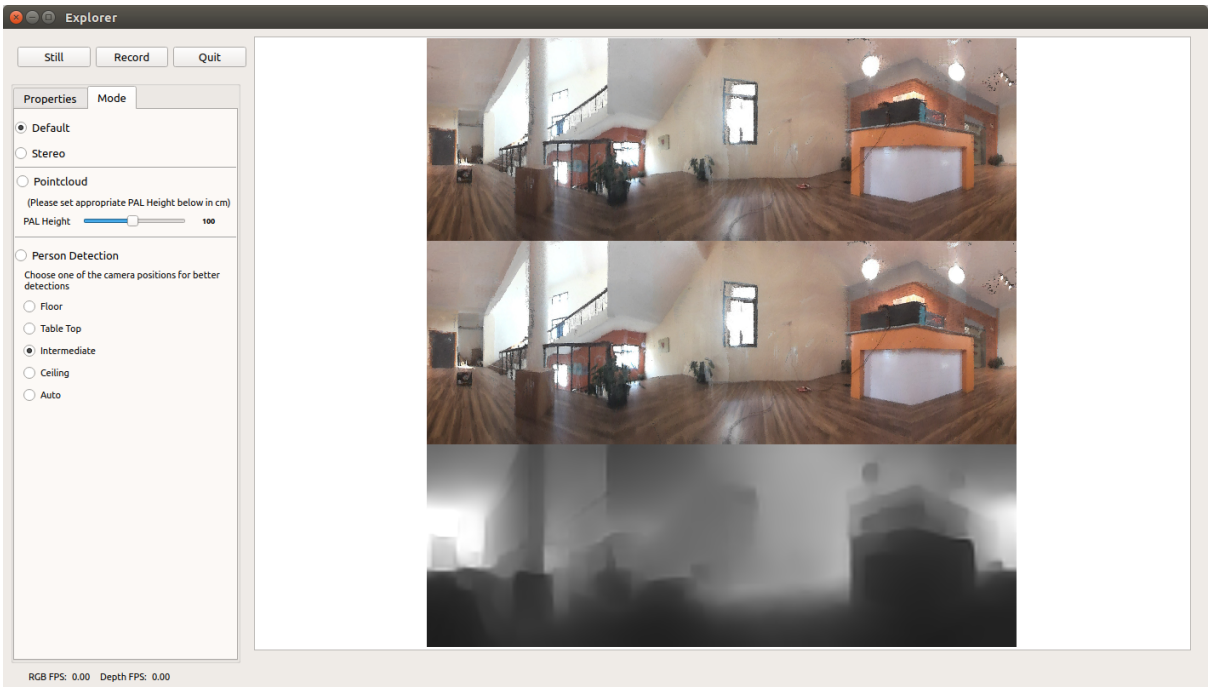


Figure 2.2: Default view of the explorer

The following sections provide a detailed outline of the features of the Explorer.

2.1.1 Viewing modes

There are four viewing modes in the Explorer to view the output from the PAL camera. The Mode pane allows you to switch between these modes:

1. Default mode
2. Stereo mode
3. Point Cloud mode
4. Person Detection mode

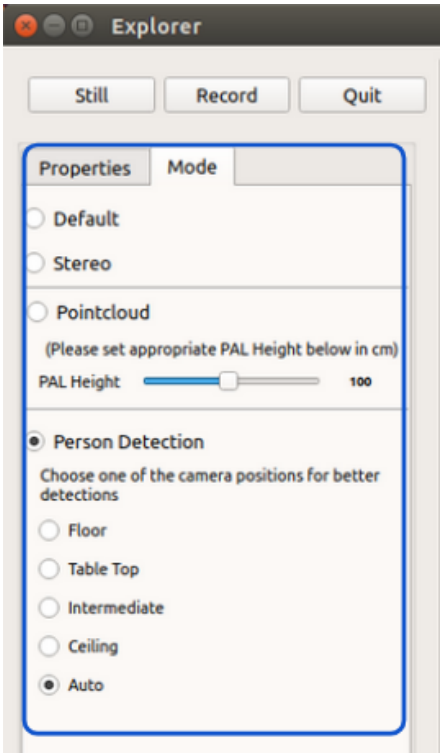


Figure 2.3: Mode pane highlighted

The Default mode shows the left, right and depth panoramas generated from a single image from the PAL Camera.

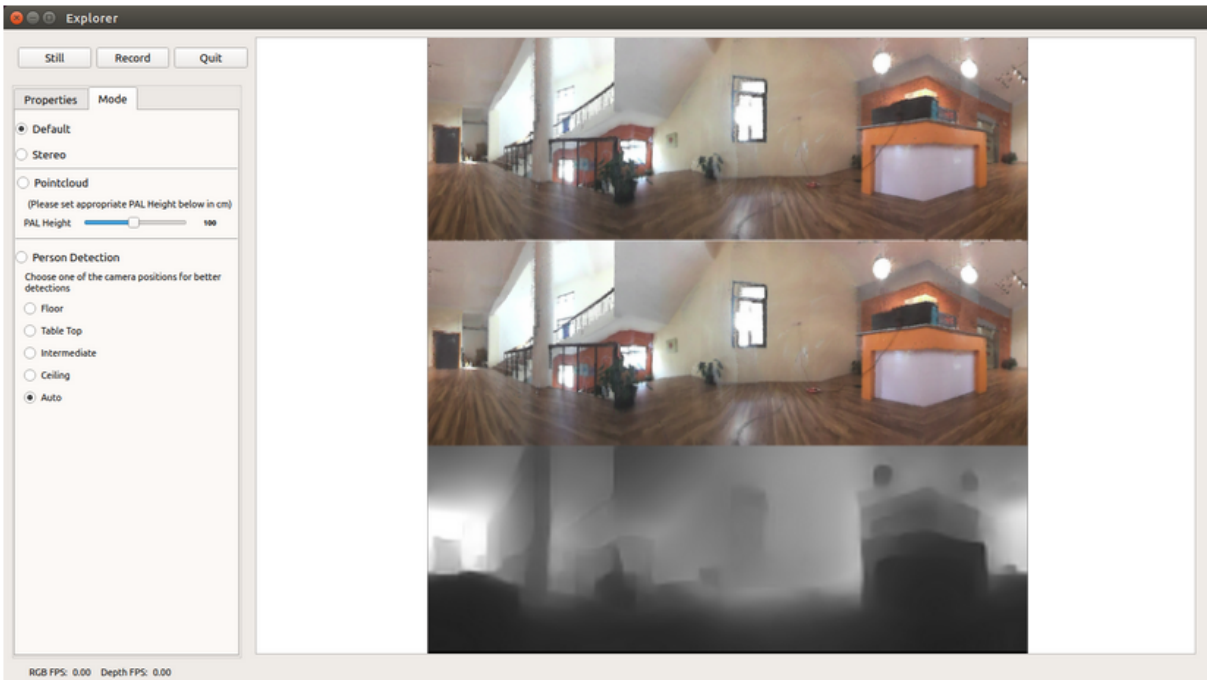


Figure 2.4: Default mode

The Stereo mode displays stereoscopic left and right panoramic views captured by the PAL Camera:

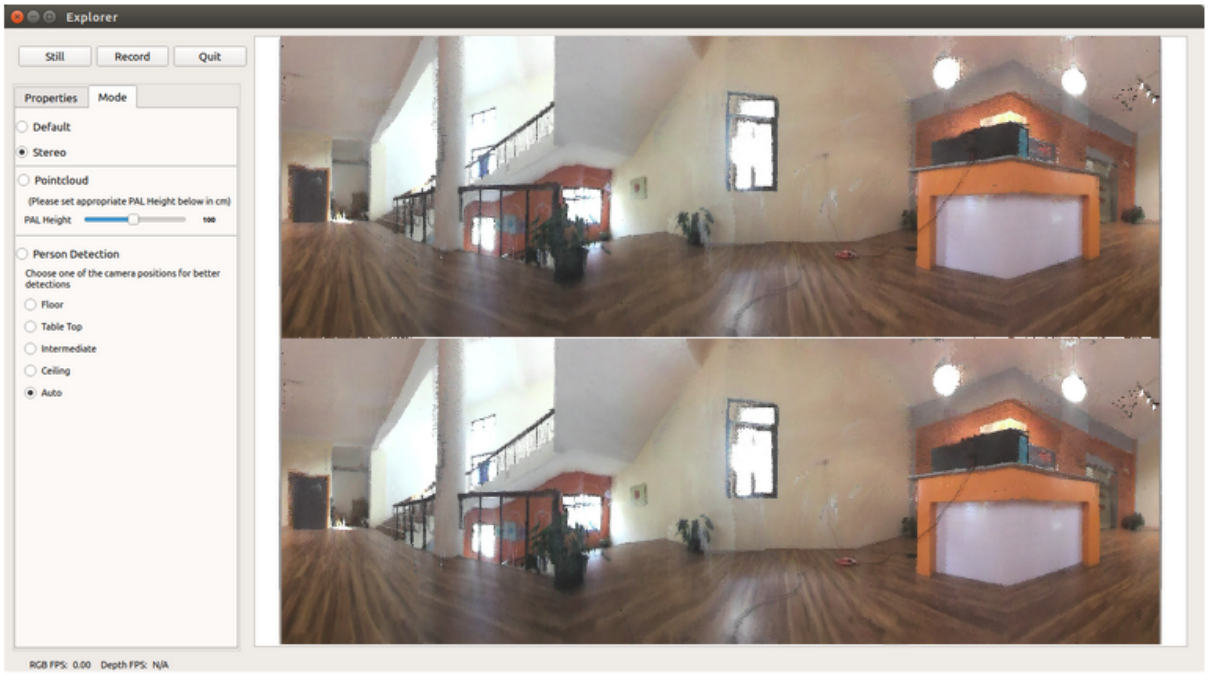


Figure 2.5: Stereo mode

The Point cloud mode shows a full 360 degree 3D point cloud visualized in the Explorer window. The height of the PAL camera from the floor needs to be

specified in this mode.

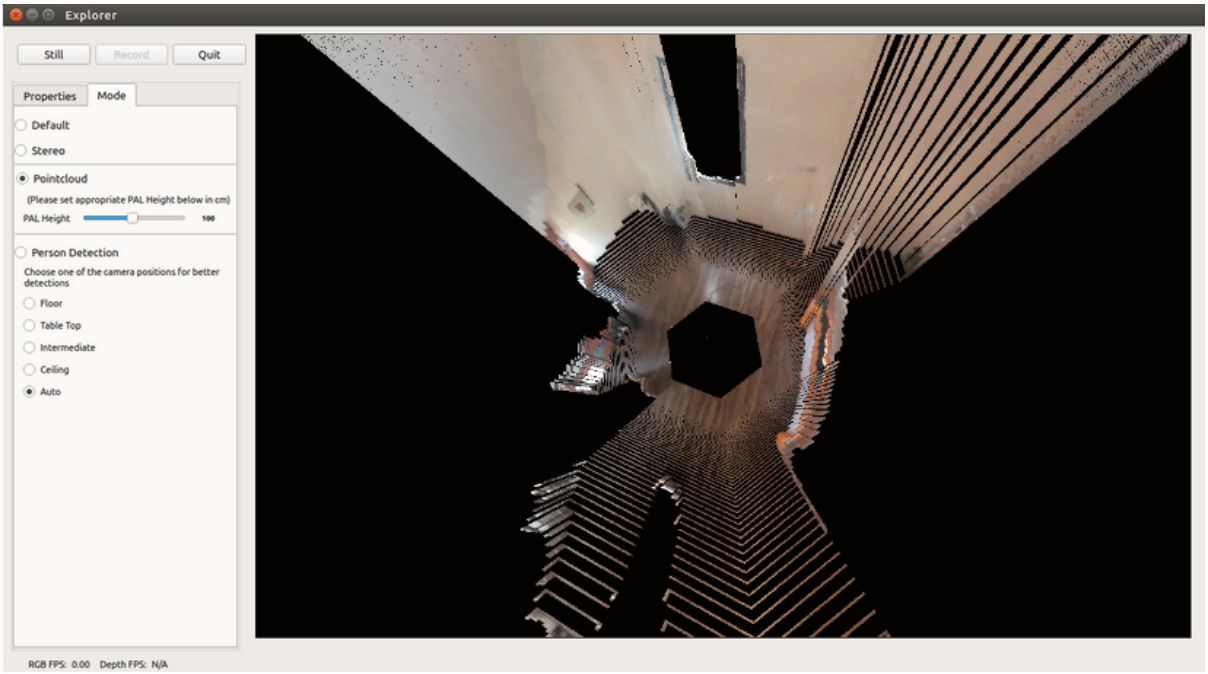
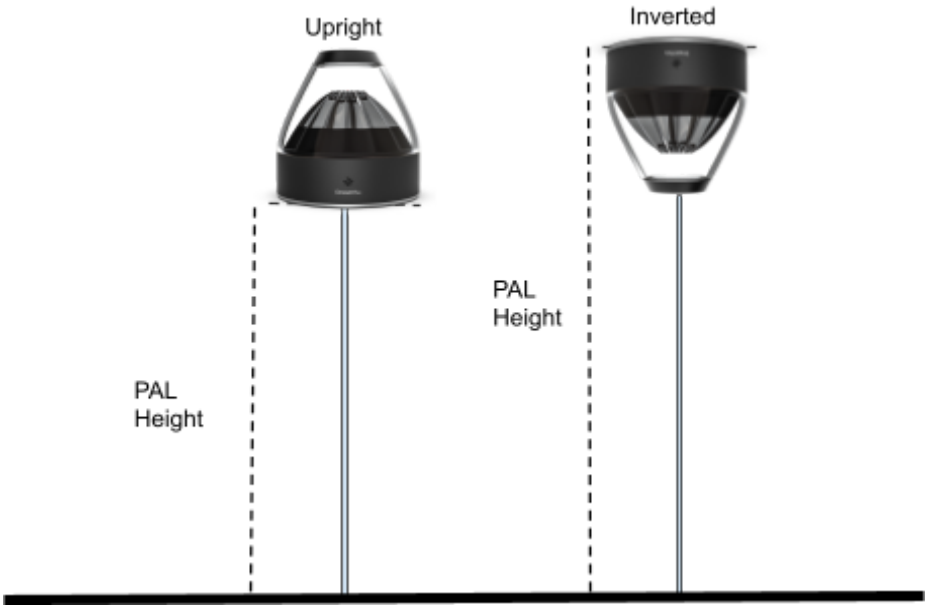


Figure 2.6 Pointcloud mode

The height of the PAL camera can be measured in centimeters as shown below:



The Person Detection Mode shows the output of PAL's person detection software with a bounding box drawn around the detected persons.

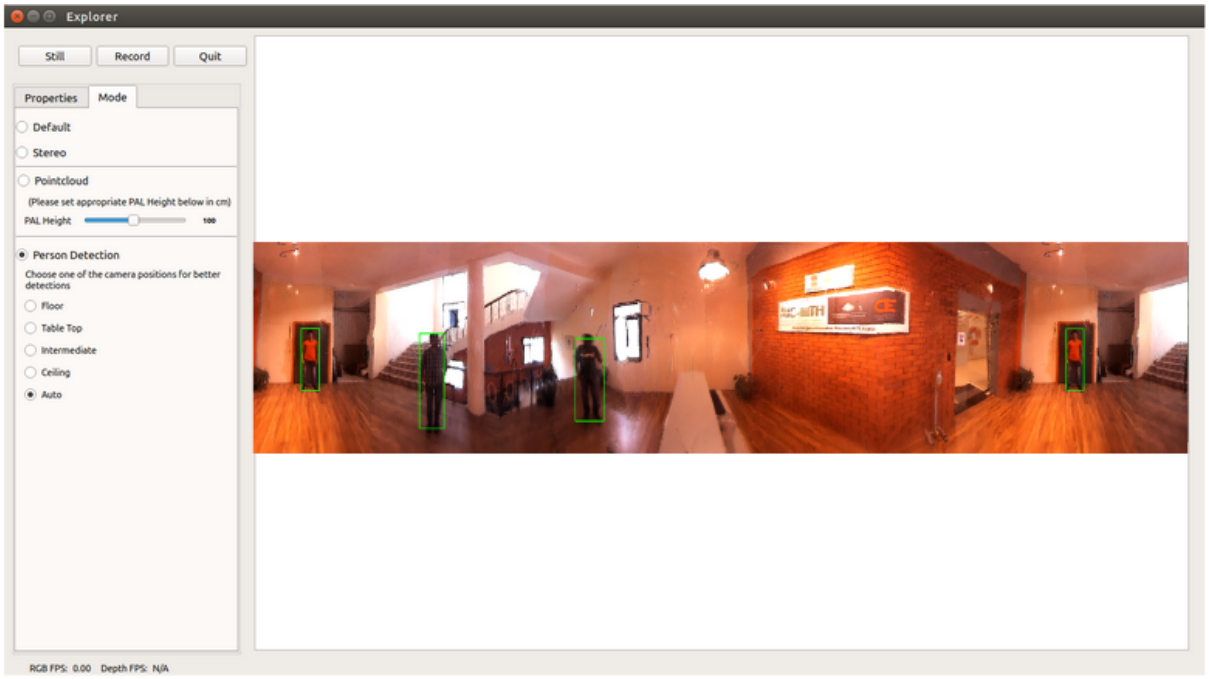


Figure 2.7: Person Detection mode

By selecting the mounting position of the PAL camera, the best parameters for person detection are automatically loaded by the Explorer application. The available mounting positions are:

Floor Mode: Suitable for camera mounting at low height (0-40cm from floor) with the camera kept in the inverted orientation.

Table-Top Mode: Suitable for camera kept on a table-top (40cm-100cm from floor) and kept in the inverted orientation.

Intermediate Mode: Suitable for camera at intermediate mounting heights (100cm-180cm from floor) and kept in the upright orientation.

Ceiling Mode: Suitable for camera mounting at high mounting locations (>180cm from floor) and camera kept in the upright orientation.

Auto Mode: Automatically selects the appropriate mounting position between *Floor*, *Table-Top*, *Intermediate* or *Ceiling* based on the *camera_height* parameter.

2.1.2 Changing Properties

PAL properties can be configured in the properties pane. This pane is exposed by clicking on the properties tab highlighted in the image below:

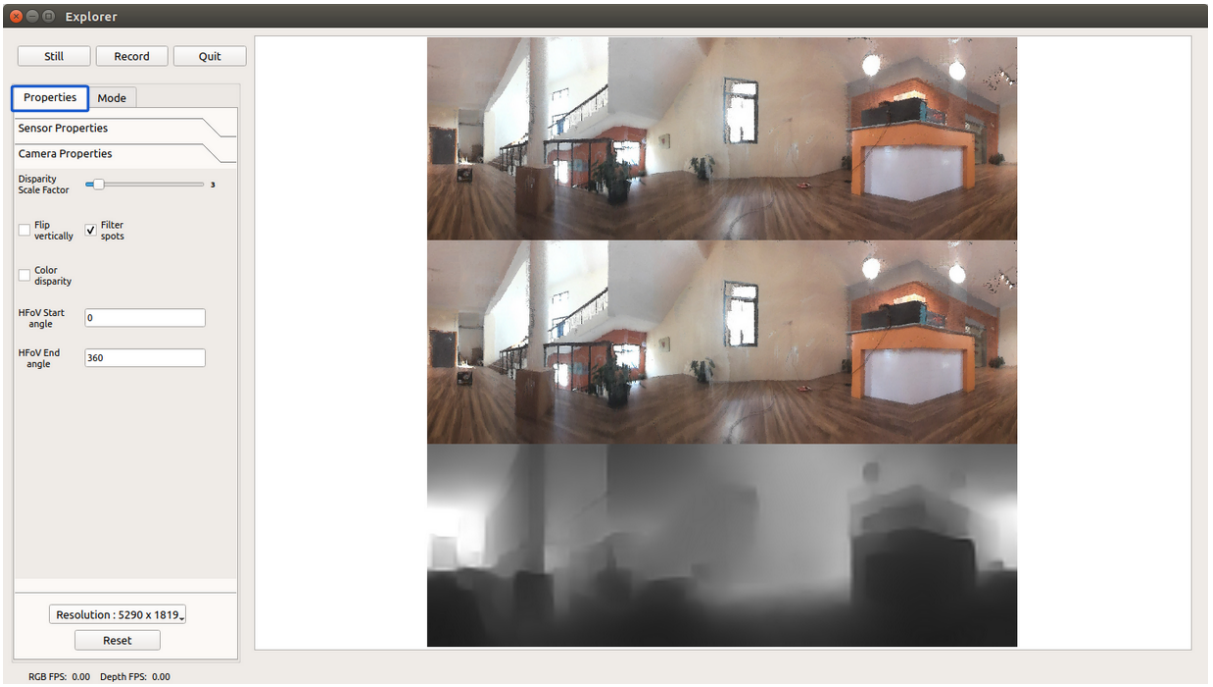


Figure 2.8: Properties Tab

The properties are divided into two sections:

- Sensor properties
- Camera properties

2.1.2.1 Sensor Properties

This group of properties is associated with the imaging sensor used in the PAL camera. All the properties in this section have corresponding members in the API CameraPropertiesStructure.

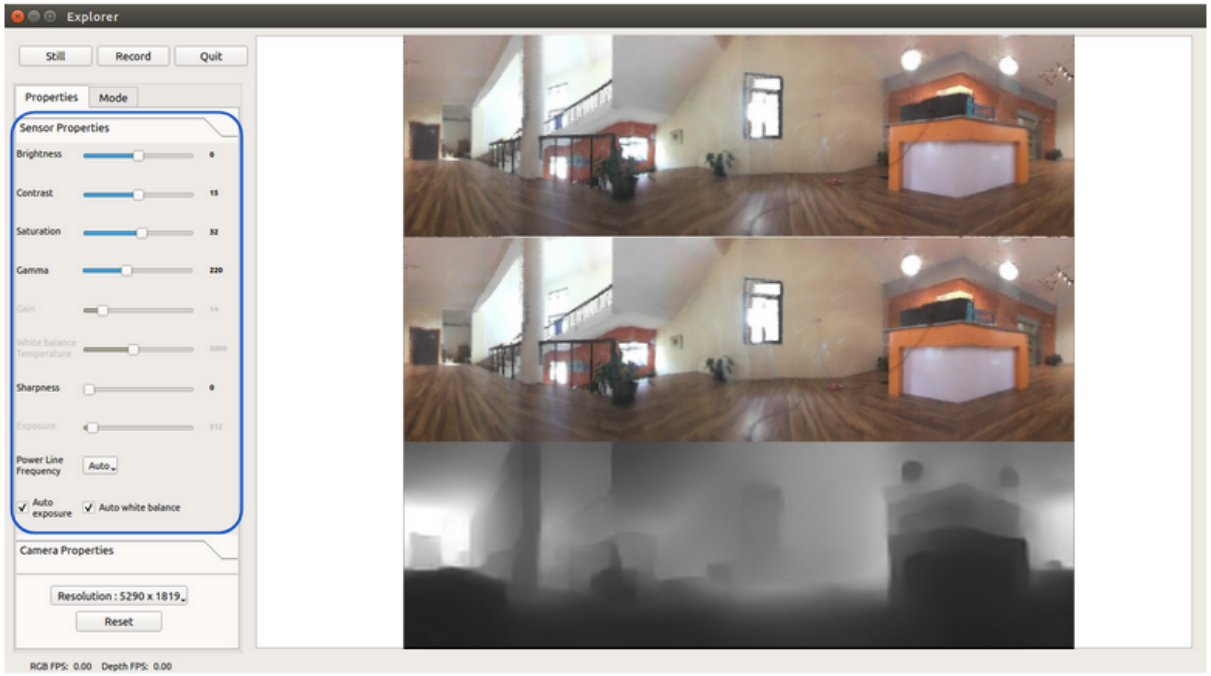


Figure 2.9: Sensor Properties

Brightness, Contrast, Saturation, Gamma, Gain, White balance temperature, Sharpness and Exposure can be modified by dragging the corresponding sliders.

Auto Exposure checkbox can be checked or unchecked to enable or disable automatically adjusting the exposure of PAL respectively.

- *Note:* When Auto Exposure is checked the Gain and Exposure sliders are disabled.

Auto White Balance checkbox can be checked or unchecked to enable or disable automatically adjusting the white balance temperature of PAL respectively.

- *Note:* When Auto White Balance is checked White Balance Temperature slider is disabled.

2.1.2.2 Camera Properties

This grouping of properties is associated with the PAL Camera itself. Most of the properties in this section have corresponding members in the API CameraProperties structure. Exceptions are called out in the description.

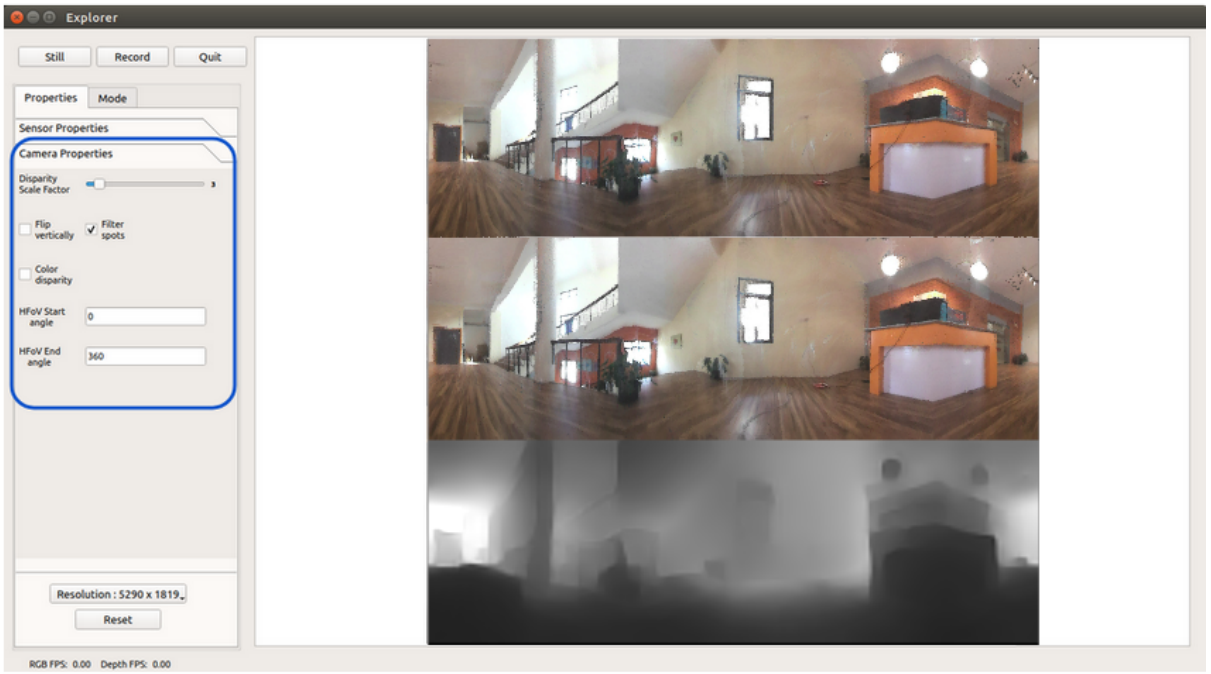


Figure 2.10: Camera Properties

Disparity Scale Factor is a property exclusive to the Explorer application to allow users to scale and visualize the disparity computed by the API backend. It is a simple scalar multiplier applied to disparity values output by the API. It allows users to visually perceive disparity of the scene better over a larger depth range.

Flip Vertically checkbox can be checked / unchecked to enable / disable vertically flipping the output data frames respectively. This is particularly useful if the device is being used in a top-down orientation.

Filter Disparity checkbox can be checked or unchecked to view filtered disparity or raw disparity respectively.

Filter Spots checkbox can be checked or unchecked to enable / disable filtering of bright spots (Eg. specular reflections) respectively. This feature is useful when the PAL camera is upright and/or exposed to harsh lighting conditions.

Color Disparity checkbox can be checked or unchecked to apply or not apply a color map to disparity respectively. This property is exclusive to the Explorer application.

HFoV Start Angle & **HFoV End Angle** can be used to select a subsection of the complete 360° horizontal field of view PAL offers. The inputs should be entered in degrees.

2.1.2.3 Ungrouped elements

This section describes ungrouped properties and UI elements in the properties pane.

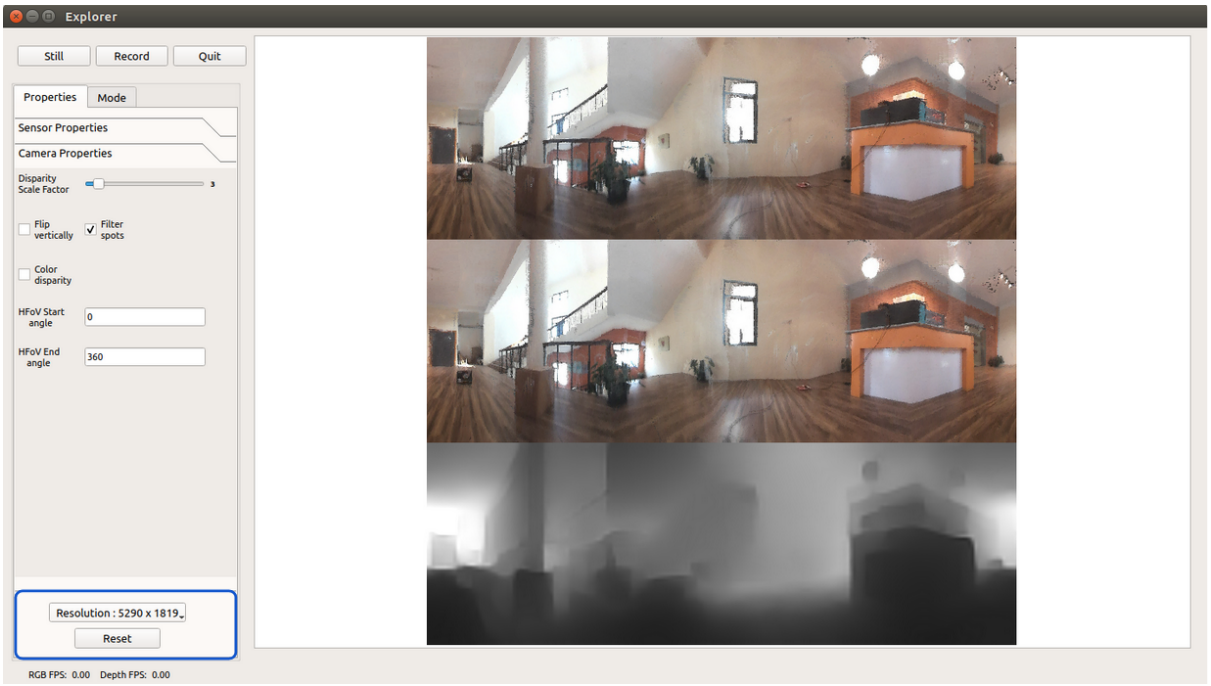


Figure 2.11: Ungrouped properties & UI elements

Resolution Dropdown can be used to switch resolutions dynamically.
Note: It is not possible to switch resolutions when video recording is in progress.

Reset Button:

The Reset button (at the bottom of the Properties pane) resets the camera properties to what they were at the beginning of the session i.e. it undoes the changes made in the current session.

Note: To reset camera properties to default values, close the Explorer application, delete the `SavedPalProperties.txt` file and restart the application.

2.1.3 Capturing Images

Capturing images is as simple as clicking the Still button highlighted in the image below:

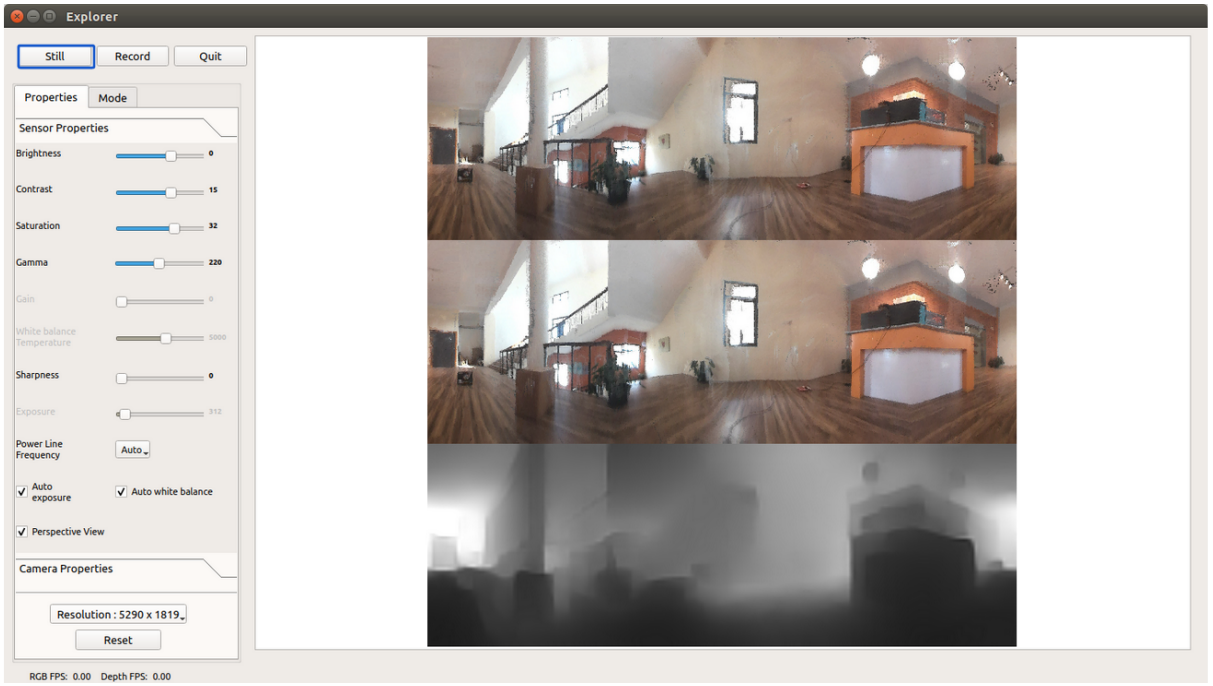


Figure 2.12: Still button highlighted

The captured image is saved in the `./Explorer/images/` folder. The image is saved uncompressed in a .png format. The image captured corresponds to the mode selected i.e. whatever panoramas are displayed on the screen in the current mode are captured as a single image.

2.1.4 Recording Video

To begin recording a video click the Record button highlighted in the image below:

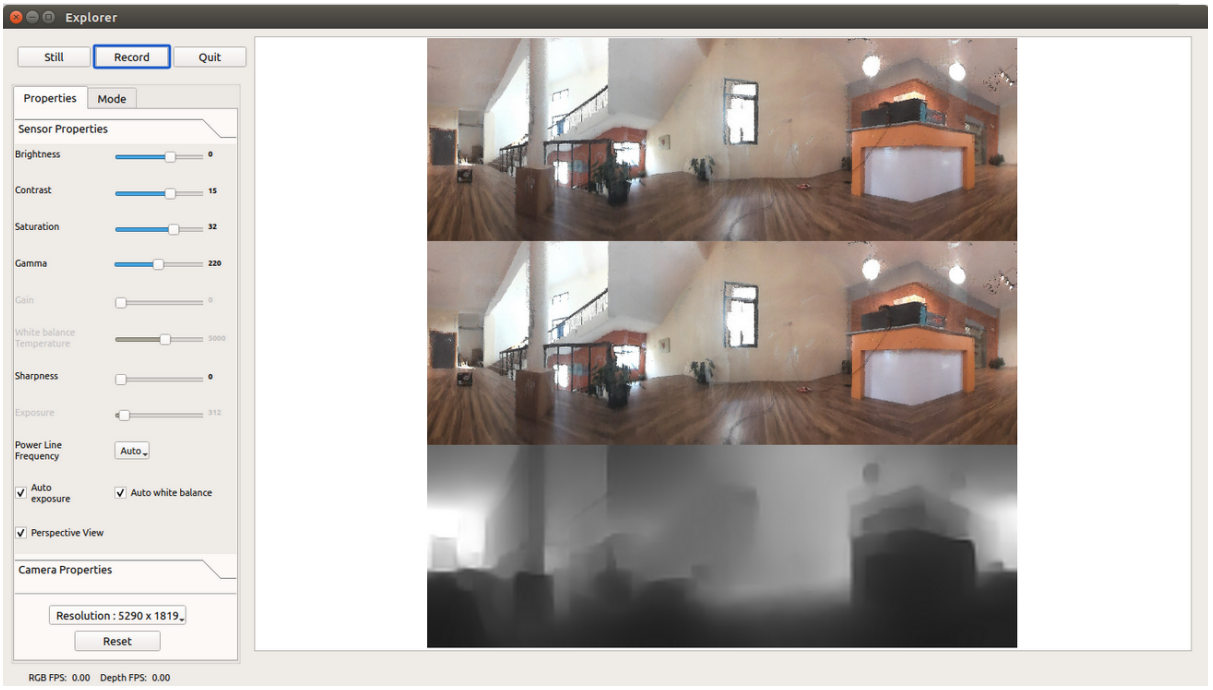


Figure 2.13: Record button highlighted

Once it is clicked, the Record button changes state to the Stop button highlighted in the figure below:

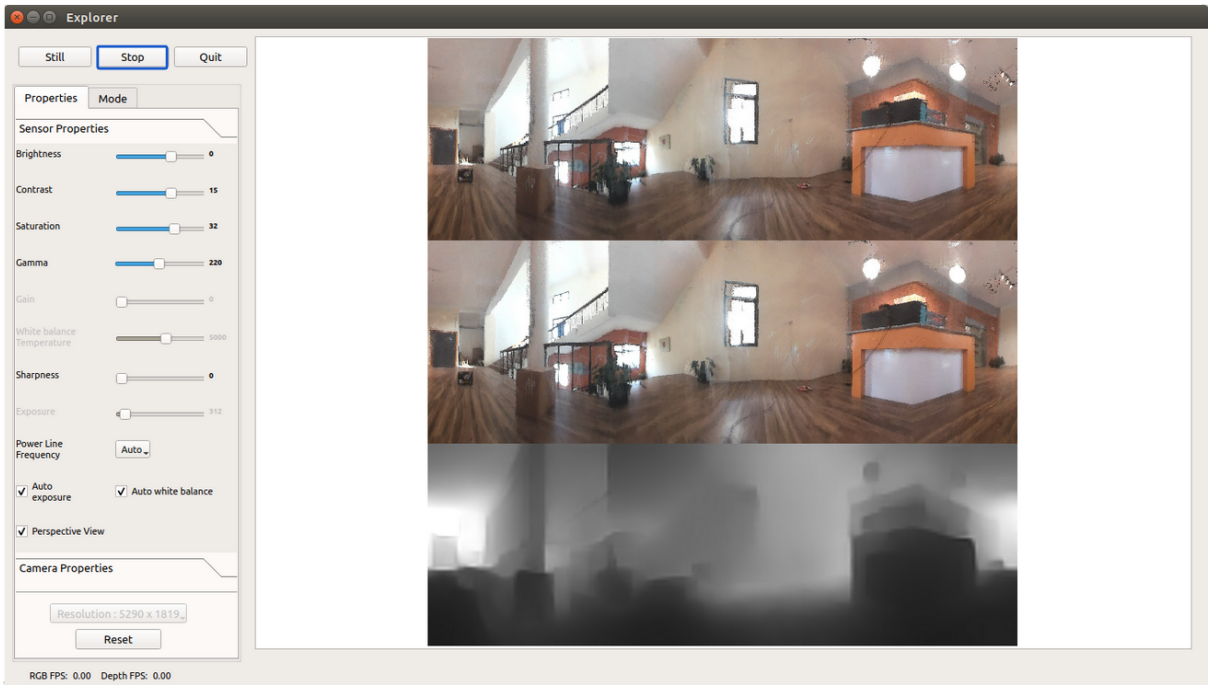


Figure 2.14: Stop button highlighted

You can click the Stop button to stop recording the video. It is saved to an automatically named file in the `./Explorer/videos/` folder. The video captured

corresponds to the mode selected i.e. whatever panoramas are displayed on the screen in the current mode are captured in the video. The video is recorded uncompressed using YUV420 format in an .avi file container.

Note: When recording a video, changing modes or resolution is not allowed.

2.1.5 Quit Application

You can quit the application by clicking the quit button highlighted in the following image:

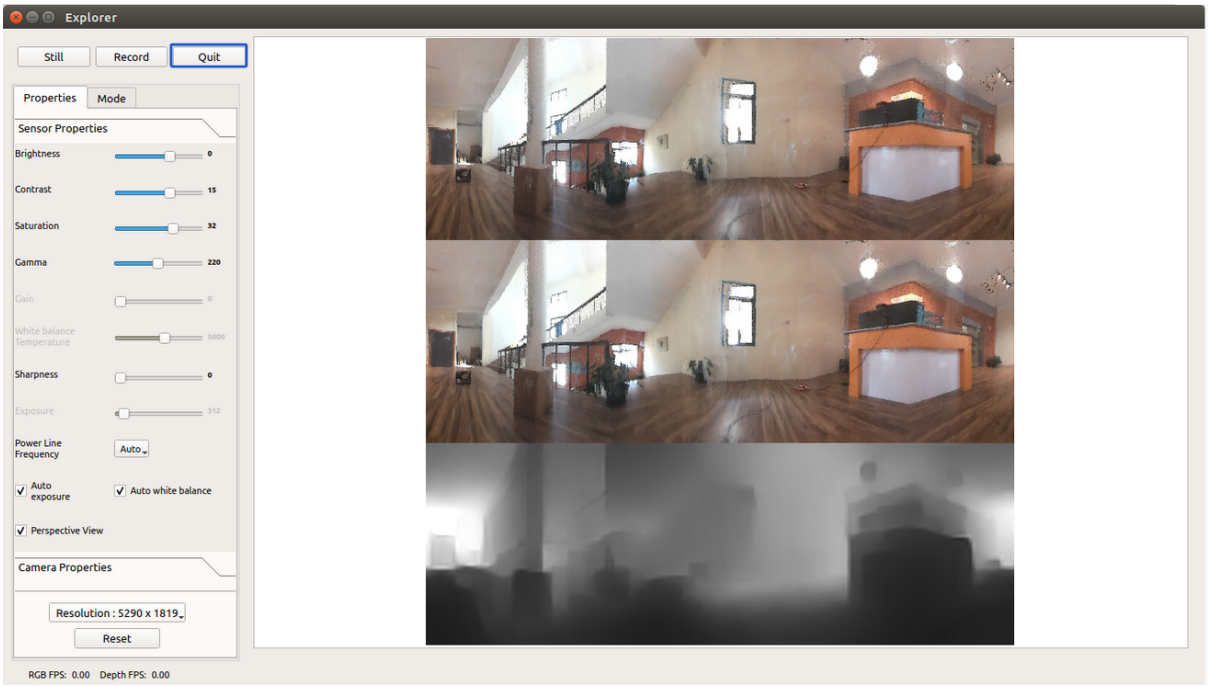


Figure 2.15: Quit button highlighted

Note:

- When the explorer is quit it saves the current properties configuration to a `SavedPalProperties.txt` file in the same directory as the application. This file can be used to initialize the ROS pal_camera_node node or as part of the LoadProperties call.
- When the Explorer is launched, it checks if the `SavedPalProperties.txt` file exists, if it does, it loads those parameters, if not, it loads the default properties as discussed in the API section.

2.2 Mapping & ODOA with PAL

This section provides a detailed outline for using the PAL camera for 2D Mapping and 3D ODOA in ROS. The PAL camera generates a 2D laserscan with 3D context that can be used to identify the distance of the nearest obstacle in all directions around the camera. The laserscan can be used for precise 2D mapping of the environment around the camera. We provide a tutorial demonstration for the same in this section. We mount the camera on an off-the-shelf Turtlebot 2 and demonstrate 2D mapping using the GMapping package in ROS.

The PAL SDK includes a wrapper package `dreamvu_pal_navigation` folder which facilitates the integration of PAL with ROS. The RGB stereo panoramas, the depth map and the laserscan (depth-scan) generated by the PAL camera can be accessed in ROS for mapping as shown in the upcoming sections.

Note: It is assumed here that the PAL starter kit is setup and the *pal_stream* executable is running as described in Section 3.4

2.2.1 Building the ROS package:

The `dreamvu_pal_navigation` package in the SDK can be compiled using the following steps:

- Place the `dreamvu_pal_navigation` package inside *catkin_ws*
- Build the package by running the following commands in a terminal window.

```
cd ~/catkin_ws
catkin_make
source ./devel/setup.bash
```

2.2.2 PAL Publisher Node:

The `dreamvu_pal_navigation` package consists of a node of `publisher` type with the name `pal_publisher_node`. This node publishes ROS topics corresponding to different informations streams from PAL that are listed in the following table:

Data published	ROS Topic	Message Type
Left panorama	/dreamvu/pal/get/left	sensor_msgs::Image
Right Panorama	/dreamvu/pal/get/right	sensor_msgs::Image
Depth panorama	/dreamvu/pal/get/depth	sensor_msgs::Image
Laser-Scan	/dreamvu/pal/get/scan	sensor_msgs::LaserScan

Table 4.1: List of Rostopics published and subscribed by capture node

The laserscan from the PAL camera is published using the `sensor_msgs::LaserScan` in a reference frame that is denoted as the `pal` frame. Note: The laserscan consists of 1720 points at an angular resolution of 0.2 degrees per element.

2.2.3 Laser Scan Parameters

For Mapping, ODOA and Navigation, the laser-scan from the PAL camera (published as the scan) topic can be used. This scan provides the 2D occupancy map of the 360 view around the camera, but utilizes a vertical 3D context to find the nearest obstacle. While this is similar to a scan generated from laser-scanners, the occupancy is created from 360 stereoscopic panoramas and therefore includes a set of parameters for ideal operation and performance.

The `dreamvu_pal_navigation` package consists of a set of parameters contained within a metadata file named `odoa.yml`. This file contains the parameters that define the laser-scan generated from the PAL camera. Based on the end application - mapping, obstacle detection or navigation, these parameters can be adaptively modified during deployment. The default parameters, along with descriptions of the parameters are provided below. We use the default parameters listed here for in the following sections of the document.

Parameters	Remark	Default value
depth_context_threshold	Indicates the sensitivity of the laserscan to obstacles	150

	present on the floor. Higher values (150-180) are suitable for textured floors while lower values (120-150) are suitable for non-textured, dull floors.	
depth_context_sigma	This parameter is required for estimation of the floor regions. Values from (11-25) are suitable for textured floors, with lower values (0-10) for non-textured floors.	0
stereo_threshold	This parameter specifies the near-range sensitivity of the laser. For low mounting heights set this value to zero. This value should not exceed 100.	70
odoa_start_hfov and odoa_end_hfov	Since the PAL camera spans a fully 360 view, it often happens that a part of the robot occludes some portion of the horizontal field-of-view. The h_fov parameters specify subset of the horizontal field of view for computing the laserscan. It is recommended that the robot parts should be removed from the horizontal field-of-view for ideal operation.	0-360
odoa_start_vfov and odoa_end_vfov	These parameters specify the subset (rows) of the vertical field of view in which one wishes to provide near-range sensitivity.	670-1217
camera_height	This parameter denotes the height at which the camera is mounted from the surface of the floor. The height can be measured as shown in the Figure 4.1.	50
depth_context_temporal	This parameter can be used to control the amount of jitter in	3

	the laserscan. Higher values will stabilize the laserscan at the cost of latency, while lower values have more jitter and less latency.	

Table 4.2: List of parameters for the Laserscan from the PAL Camera

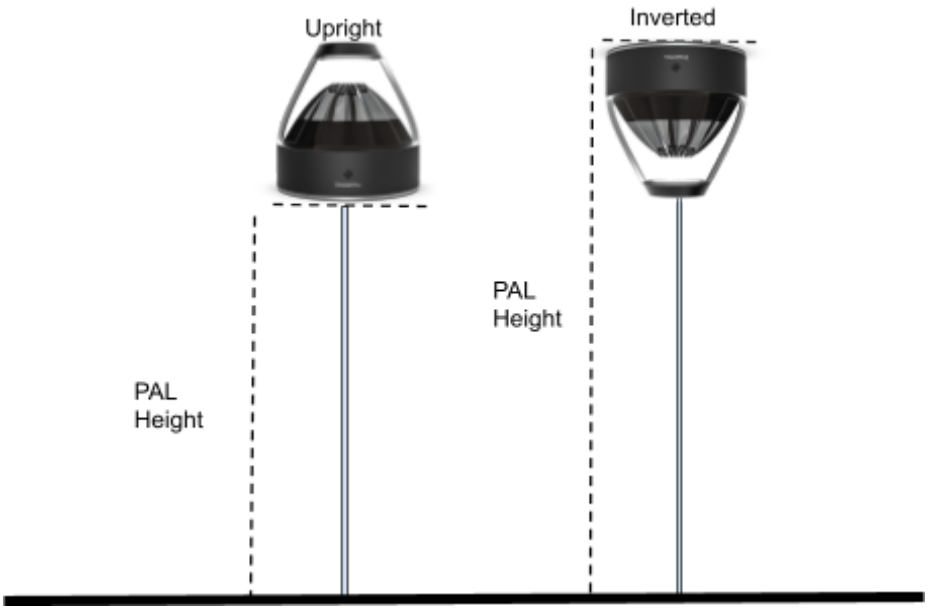


Figure 4.1 - Height of the PAL Camera from the Floor

2.2.4 Mounting PAL on the Robot

The PAL camera has an asymmetric field-of-view about the horizon and therefore needs to be mounted appropriately onto the robot for ideal operation. More details on the field-of-view are provided in the Technical Documentation. For mapping, obstacle detection and navigation using a PAL, the camera needs to be **flat mounted** on the robot in the **upright** orientation for ideal operation.

Once the camera is mounted, the camera's coordinate system needs to be linked with the robot's baselink coordinate system. A detailed diagram explaining this relationship is provided in Figure 4.2. This can be achieved in ROS by publishing a static transform using the following steps.

1. Open a new terminal

2. Run the following command in the terminal and add values for the translational (x,y,z) offset and rotational (yaw) offset; (specified in Fig 4.2)
    a. It is assumed that the pitch and roll parameters are 0.0
    b. The units of (x,y,z) offsets must match the units in the tf-tree.
    c. The command adds a static transform between the `base_link` frame of the robot and the `pal` camera frame

```
$ rosrun tf static_transform_publisher <x-offset> <y-offset> <z-offset>
<yaw> <pitch=0.0> <roll=0.0> base_link pal 10
```

The yaw-offset can be computed by identifying the *front* or the *heading* of the robot in the horizontal field-of-view of the PAL camera. As an example, consider that the PAL panorama (at the highest resolution) consists of `5290` columns and the *heading* or *front* of the robot corresponds to the column `1500`. Then the yaw offset (in radians) can be computed using the following formula:

$$yaw\ offset\ = \frac{2^{*}\ \Pi\ ^{*}1500}{5290} - \frac{\Pi}{6}$$

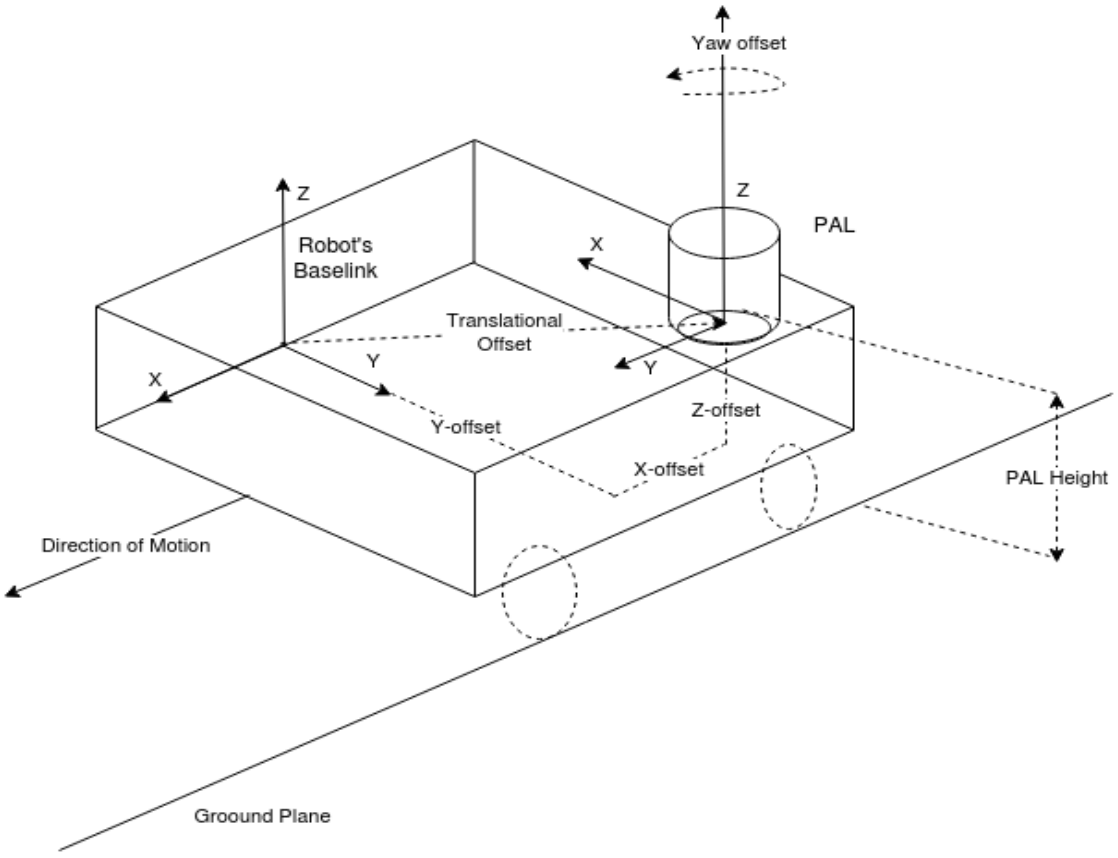


Figure 4.2 - Mounting the PAL Camera on the Robot. The translational and rotational offsets from the

robot's baselink need to be specified carefully.

2.2.5 Mapping using the PAL Camera and Robot

With the mounting and the setup of the SDK achieved, one can now use the PAL camera with the robot to map the scene in 2D using the PAL Laserscan. This section describes the steps involved in mapping the scene using the ROS Gmapping package and the PAL Camera mounted at a height of 50cm from the floor on a Turtlebot 2 robot.

The first step is to run the publisher node by running the following commands:
The `roscore` process by running this command in a terminal window:

```
$ roscore
```

In another terminal window, the bash setup file for the catkin workspace can be sourced using the following command

```
$ source ./devel/setup.bash
```

Please Note: This step has to be repeated for each new terminal opened in catkin_ws.

Next, the following command needs to be executed to start the publisher node

```
$ rosrun dreamvu_pal_navigation publisher
```

The node will now advertise the list of topics mentioned in Table 4.1 to the master. Once any subscriber is registered to a topic, the node will start publishing messages corresponding to that topic.

The `dreamvu_pal_navigation` package also provides a launch file to preview the camera information being published in Rviz. To launch Rviz the following command can be executed in a terminal window, which will display the window shown below

```
$ roslaunch dreamvu_pal_navigation scan_rviz.launch
```

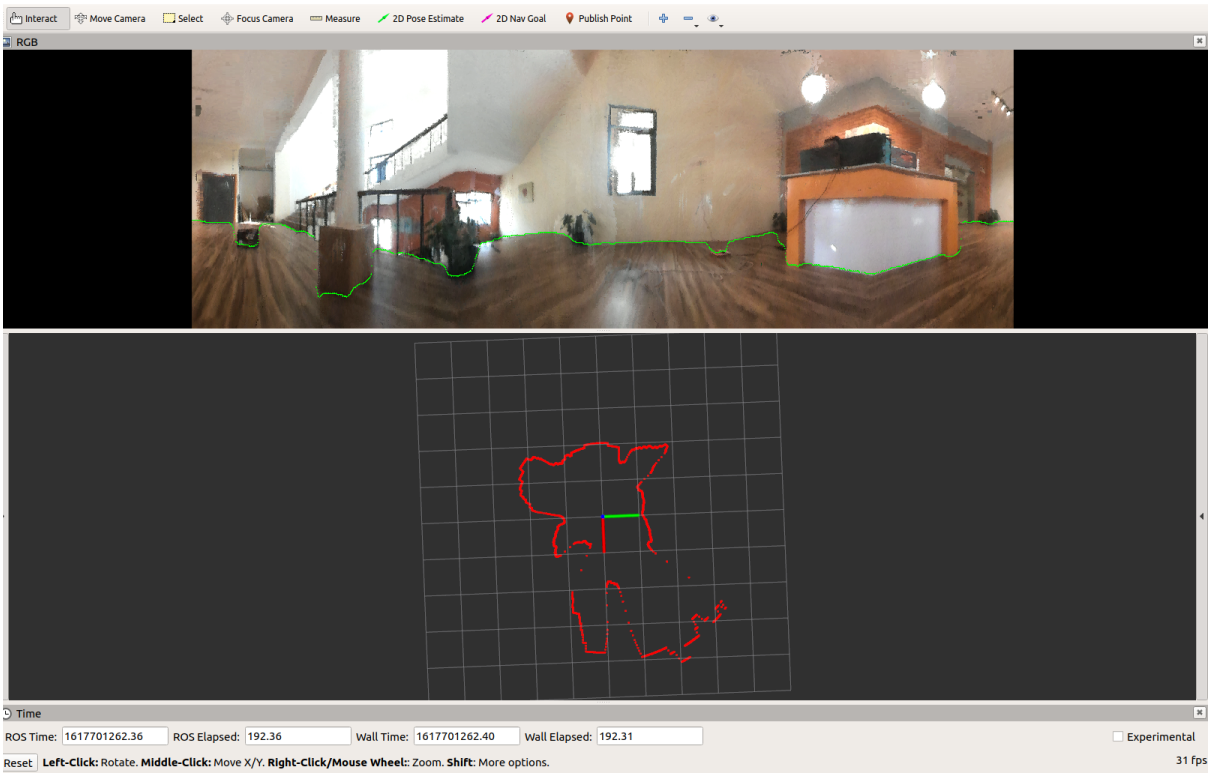


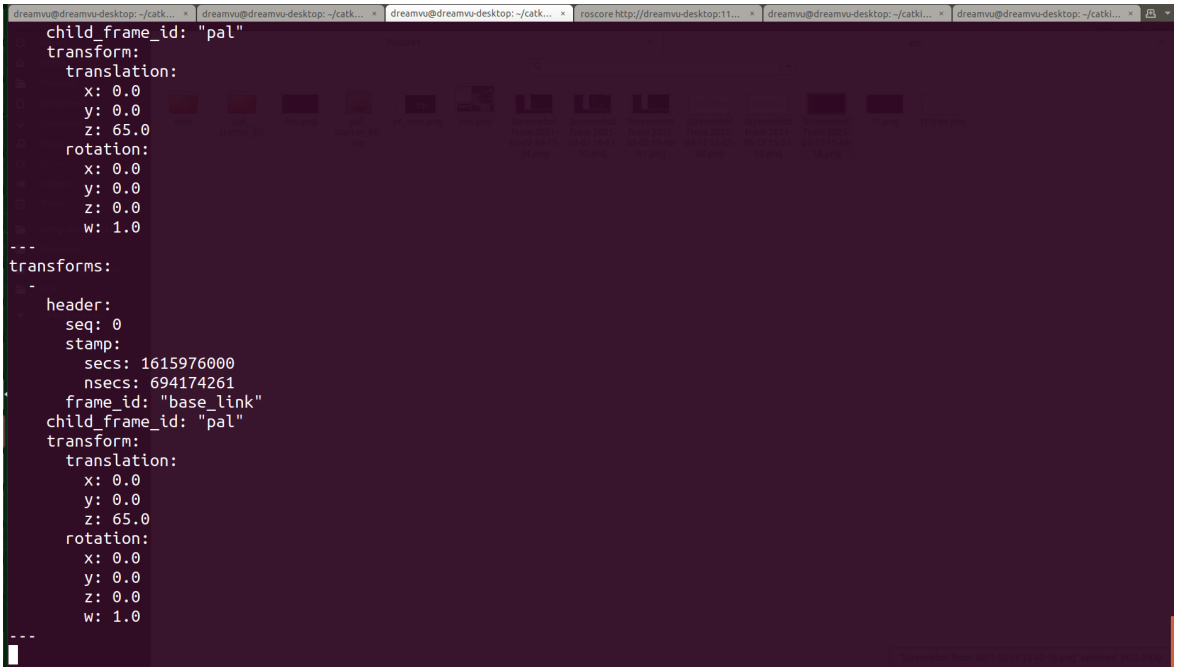
Figure 4.3 RVIZ window on launching the `scan_rviz.launch` file

2.2.5.1 TurtleBot 2 Bringup and Tf-Tree

As per the mounting instructions provided in Section 4.2.4, we mount the PAL camera on the TurtleBot 2 in the inverted orientation. To ensure the correct transform between the `base_link` frame of the robot and the `pal` frame of the camera, the following command can be executed in a terminal window.

```
$ rostopic echo /tf
```

This command will display the following information, which should be the same as the specified transformations:



The TurtleBot 2 can now be powered on and the bring-up activity can be performed by executing the following commands in a terminal window:

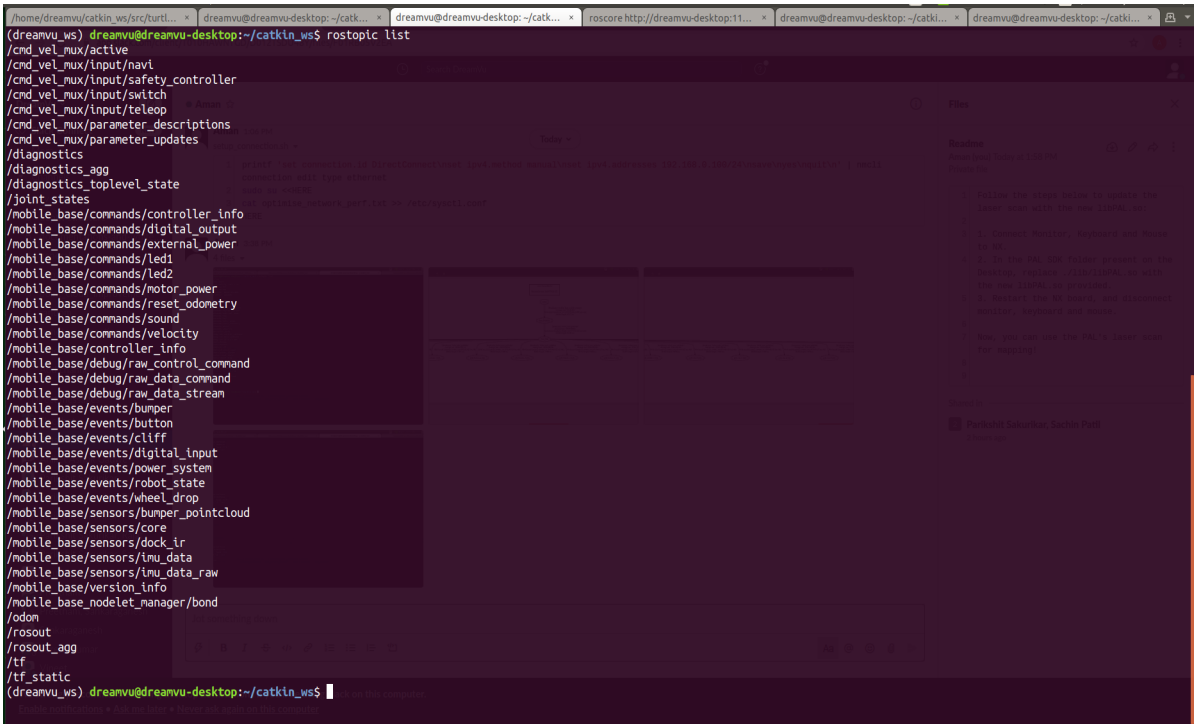
```
$ cd catkin_ws
$ source devel/setup.bash
$ roslaunch turtlebot_bringup minimal.launch
```

A sound from the TurtleBot 2 will confirm that the bring-up was successful.

Note: It must be ensured that the TurtleBot is publishing to the `/odom` topic and that the tf-tree is updated with the tf between the `odom` and the `base_link` of the TurtleBot.

To check the list of topics being published, the following command can be used which splashes the topic list on the terminal window as shown below:

```
$ rostopic list
```



The following command can be used to check the tf-tree:

```
$ rosrun tf view_frames && evince frames.pdf
```

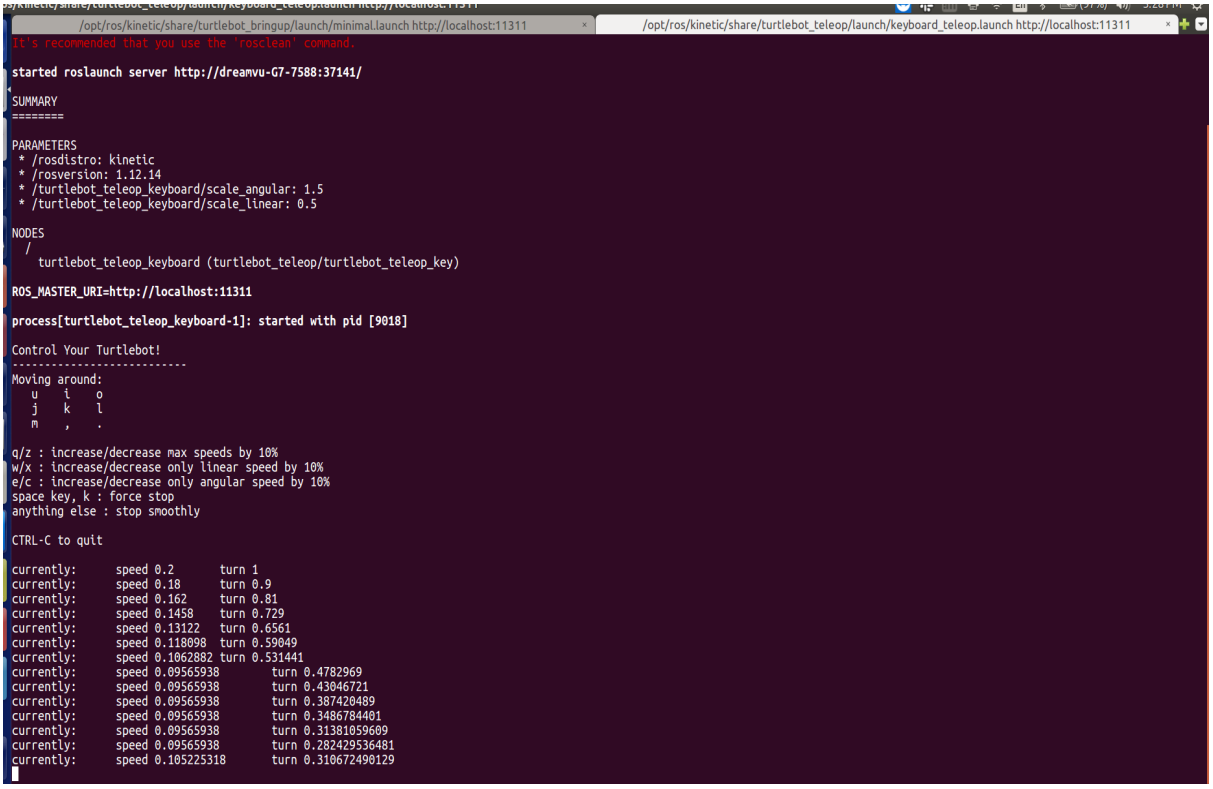
Note: This command generates a transform-tree and saves it in a file called frames.pdf. The tf-tree must be equivalent to the tf-tree shown here.

2.2.5.2 TurtleBot Teleoperation

To navigate the turtlebot, the following commands can be used in a terminal window to launch the teleoperation program.

```
$ cd catkin_ws
$ source devel/setup.bash
$ roslaunch turtlebot_teleop keyboard_teleop.launch
```

The image shown below is the default view of the teleoperation window for the TurtleBot 2. For the purposes of this tutorial, we recommend and set the translation speed to 0.1m/s and the rotational speed to 0.3 rad/s.



2.2.5.3 GMapping

The GMapping ROS package can now be launched using the provided `gmapping.launch` file in the `dreamvu_pal_navigation` package of the SDK using the following commands:

```
$ cd catkin_ws
$ source devel/setup.bash
$ roslaunch dreamvu_pal_navigation gmapping.launch
```

This launch file will open an Rviz window and one can see the live map being generated from the Laserscan of the PAL camera. It is now recommended to move the robot around the environment that is required to be mapped. It may be noted that since the laserscan captures full 360 information, the robot may only need to span the environment once.

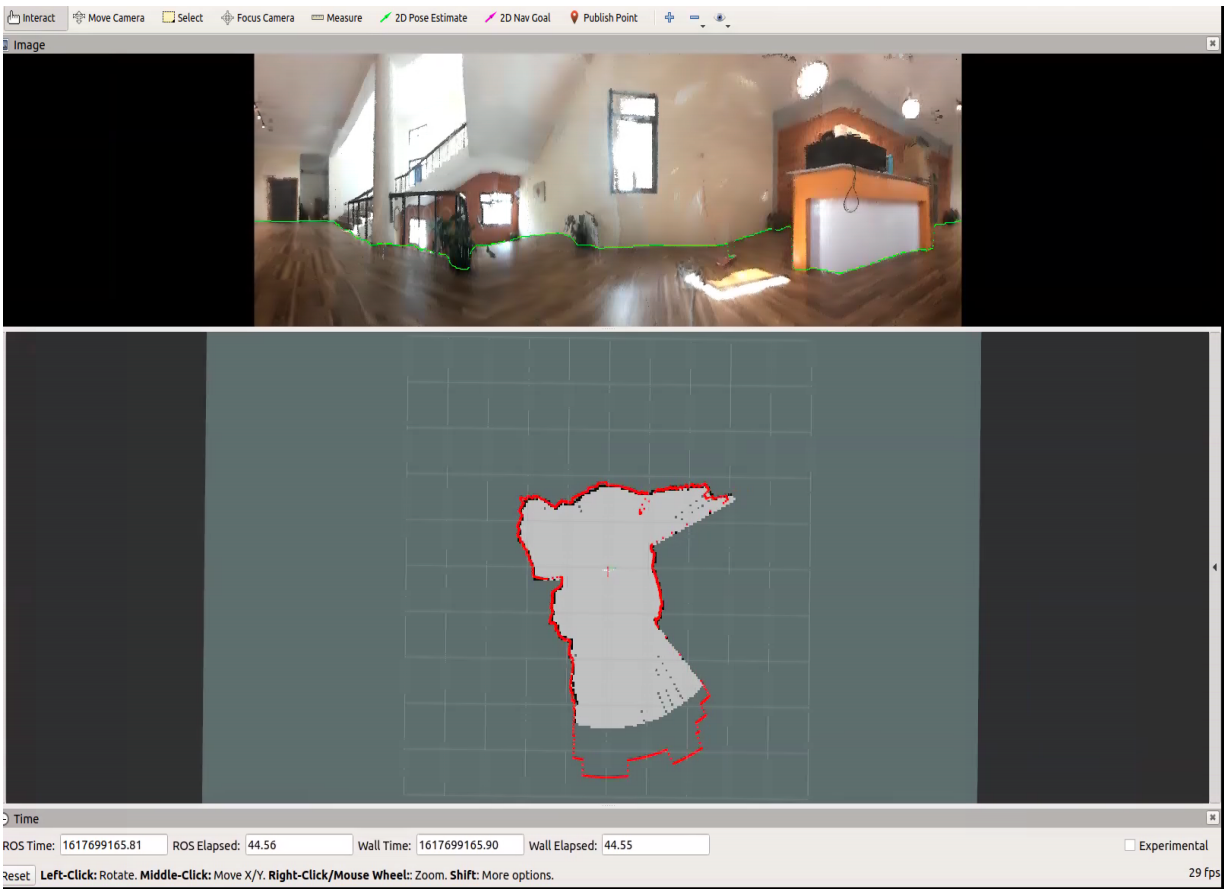


Figure 4.4 Rviz window showcasing GMapping with the PAL laserscan

Once the mapping is complete, the map can be saved using the following command.

```
$ rosrun map_server map_saver -f <filename>
```

This command saves the map in two files named `<filename>.pgm` and `<filename>.yaml`. This map can be used further for ODOA and navigation.

2.2.5.4 Recommended parameters for GMapping

A list of recommended parameters for GMapping with PAL are provided here:

Parameter Name	Value
angularUpdate	0.1
linearUpdate	0.1
particles	80
xmin	-10.0
xmax	10.0

ymin	-10.0
ymax	10.0
maxUrange	3.5
occ_thresh	0.25

Table 4.3 Recommended GMapping Parameters

These parameters can be by editing the launch file using this command:

```
$ gedit ~/catkin_ws/src/dreamvu_pal_navigation/launch/gmapping.launch
```

Additional Notes :

1. The `maxUrange` parameter is dependent on the mounting height of the camera and the following table specifies the recommended values:

Mounting Type	Mounting Height	MaxUrange
Low Mounting	Less than 20cm	1.5m - 2m
Moderate Mounting	20cm - 60cm	2m - 3m

2. The `occ_thresh` parameter controls the sensitivity to obstacles during mapping. Lower values (0.10 -0.15) increase the sensitivity to obstacles during mapping.
3. The `minimumScore` parameter can be set to a high value (>10000) if the size of the room is large and if the robot is unable to localize well enough This increases the weighted dependence on mechanical odometry from the robot.

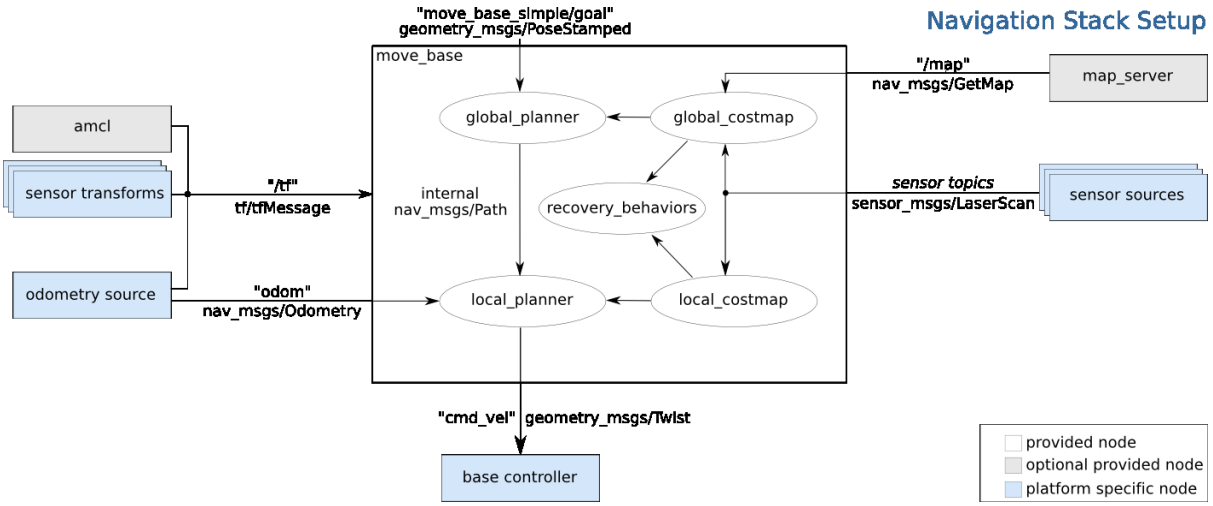
2.3 Navigation with PAL

This section provides a detailed outline for autonomous robot navigation using the PAL camera. The 2D laserscan from the PAL camera has a 3D context and can be used to identify the distance of the nearest obstacle in all directions around the camera. The laserscan can be used for navigation in a mapped environment around the camera. We provide a tutorial demonstration for the same in this section. We

mount the camera on an off-the-shelf Turtlebot 2 and demonstrate navigation using the Adaptive Monte-Carlo Localization (AMCL) package in ROS.

Given a *start* and a *goal* position, the navigation framework identifies a path around the various static and dynamic obstacles in the scene, and provides command velocities to the wheels of the robot to reach the goal position. The static (or global) obstacles are identified using the map of the scene and for the dynamic (or local) obstacles we use PAL's laser scan. For path-planning and generating the command velocities for the robot, we use the `move_base` package in ROS. Once the command velocities are generated, the `base_controller` of the robot drives the robot to goal position.

The overall architecture of the Navigation stack in ROS is defined in this image:



We use this prescribed Navigation stack along with the laserscan generated from the PAL camera for navigating an off-the-shelf TurtleBot 2 in a mapped scene. We assume that the map is created as described in Section 4.2 and the camera is mounted on the TurtleBot (Section 4.2.4) and the TurtleBot bringup is completed (Section 4.2.5.1).

2.3.1 AMCL Bringup

The `dreamvu_pal_navigation` package in ROS includes a launch file to run AMCL based navigation. The `amcl.launch` includes the path to the map file. This path needs to be updated to the path on the machine where the map is saved as shown

in the example below:

```
<arg name="map_file" default="/home/dreamvu/catkin_ws/map.yaml"/>
```

Once the path to the map is provide, the AMCL package can be launched as follows :

```
$ cd ~/catkin_ws
$ source devel/setup.bash
$ roslaunch dreamvu_pal_navigation amcl.launch
```

This will open an Rviz window similar to the one shown below :

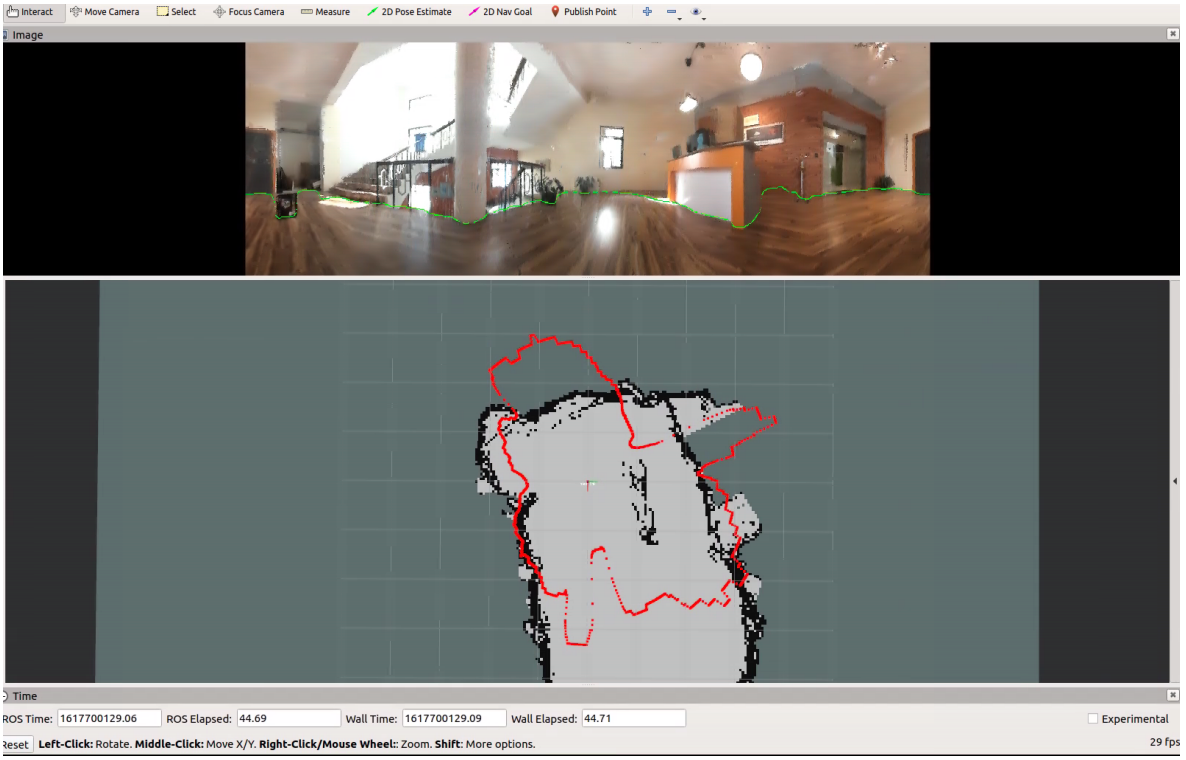


Figure 4.5 Rviz window showcasing AMCL with the PAL laserscan

The laserscan and the map will be seen in the Rviz window, albeit without alignment between the scan and the map. The initial pose estimate needs to be provided before navigation and for aligning the scan to the map. In order to provide the initial pose and location estimate of the robot, the following steps need to be executed:

1. Change the global frame to `map` in the rviz window
2. Click on the `2D Pose Estimate` tool in the Rviz taskbar and hover your mouse over the estimate of the robot's position.

3. Now click on the position of the robot in the map, and drag the green arrow to the current heading of the robot as shown in Figure 4.6. While providing the initial estimate, the laser-scan can be used as a guidance because the laserscan must align with the map once the pose is identified.

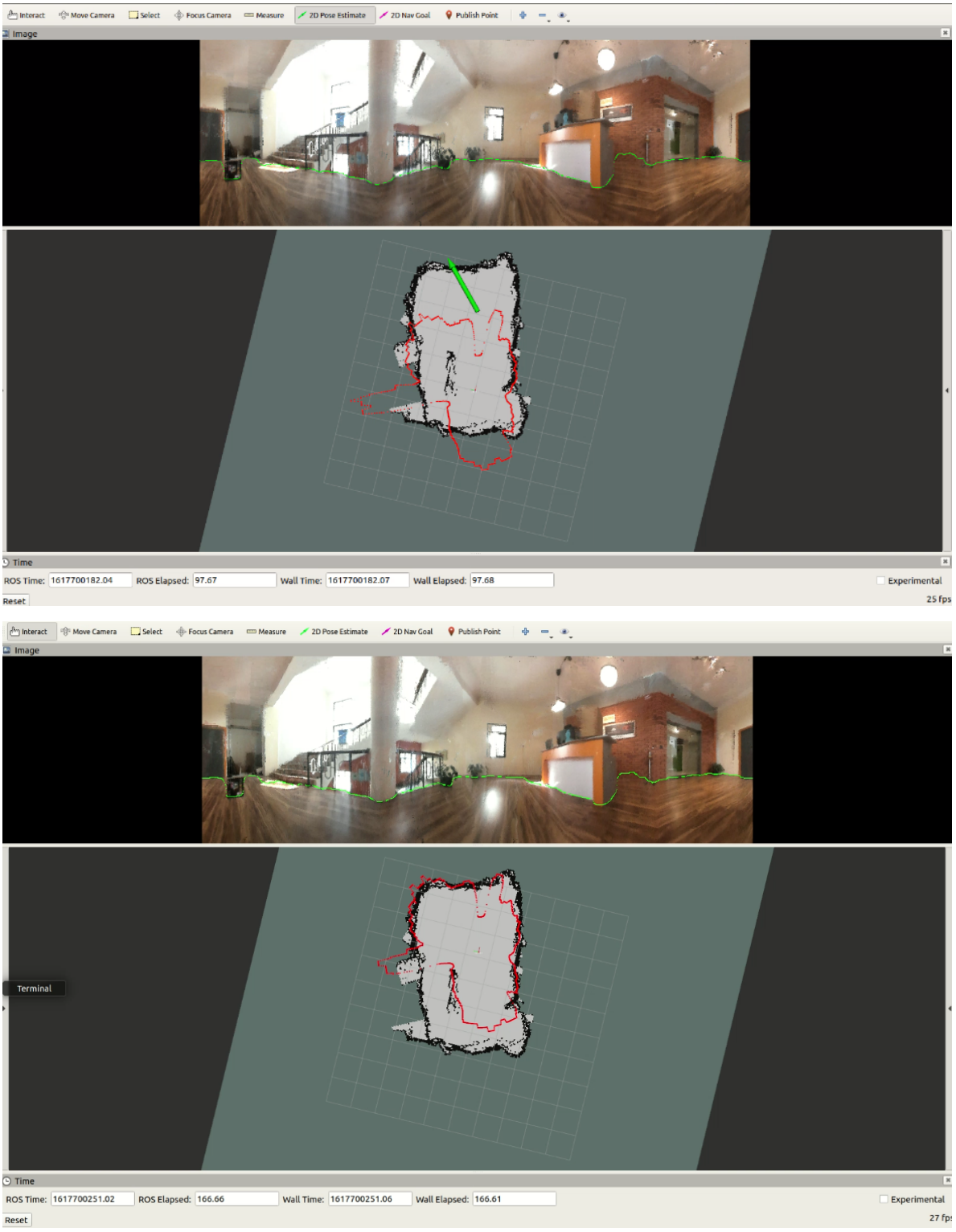


Figure 4.6 Selecting the Initial Pose Estimate of the Robot

2.3.2 Navigating the Robot with PAL

The `move_base` package in ROS is used for path planning and computing the command velocities for each wheel of the robot. This package also provides the capability to configure the local and global costmaps and the local and global planners. Once the initial pose is established and the robot is set to navigate, the `move_base` package can be launched by running the following commands in a new terminal.

```
$ cd ~/catkin_ws
$ source devel/setup.bash
$ roslaunch dreamvu_pal_navigation move_base.launch
```

Once launched, the user can subscribe to the local costmap, global costmap, local plan, global plan, robot's foot print etc. Illustrations of the global and local costmaps are shown in Figures 4.7 and 4.8.

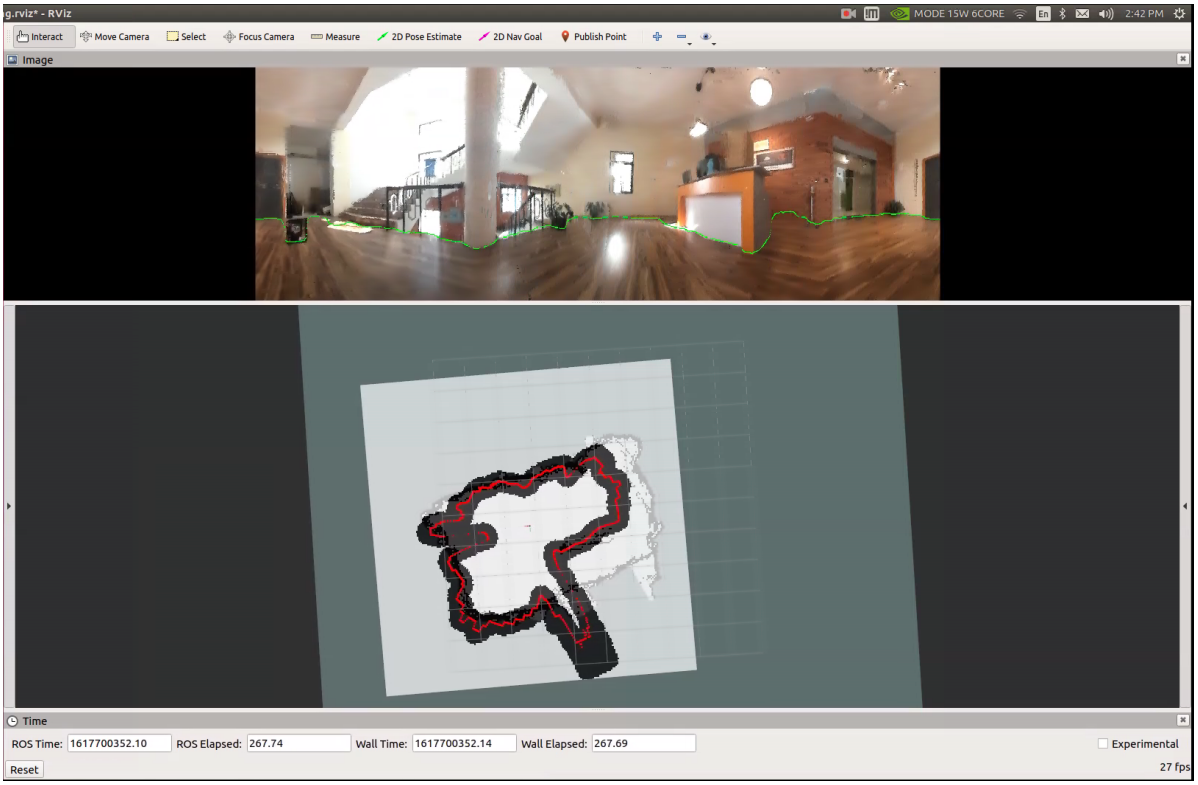


Figure 4.7 Local Cost Map surrounding the Robot

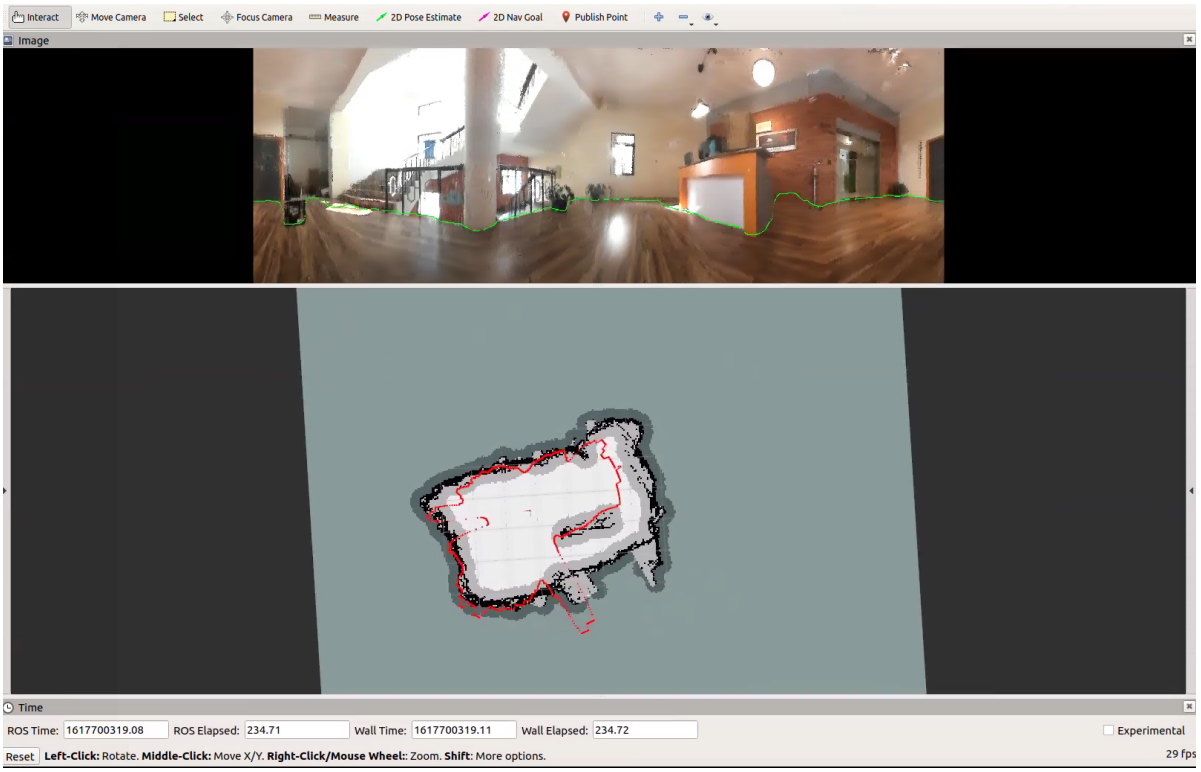


Figure 4.8 Global Cost Map across the full map of the environment

The costmaps provide an idea of the portions of the map where the robot can successfully navigate. In order to autonomously navigate the robot, a *goal* position needs to be specified to a physically approachable location in the map. In order to do so, the user can click on the `2D Nav Goal` tool in the Rviz taskbar and provide the goal position and heading as shown in the figure below :

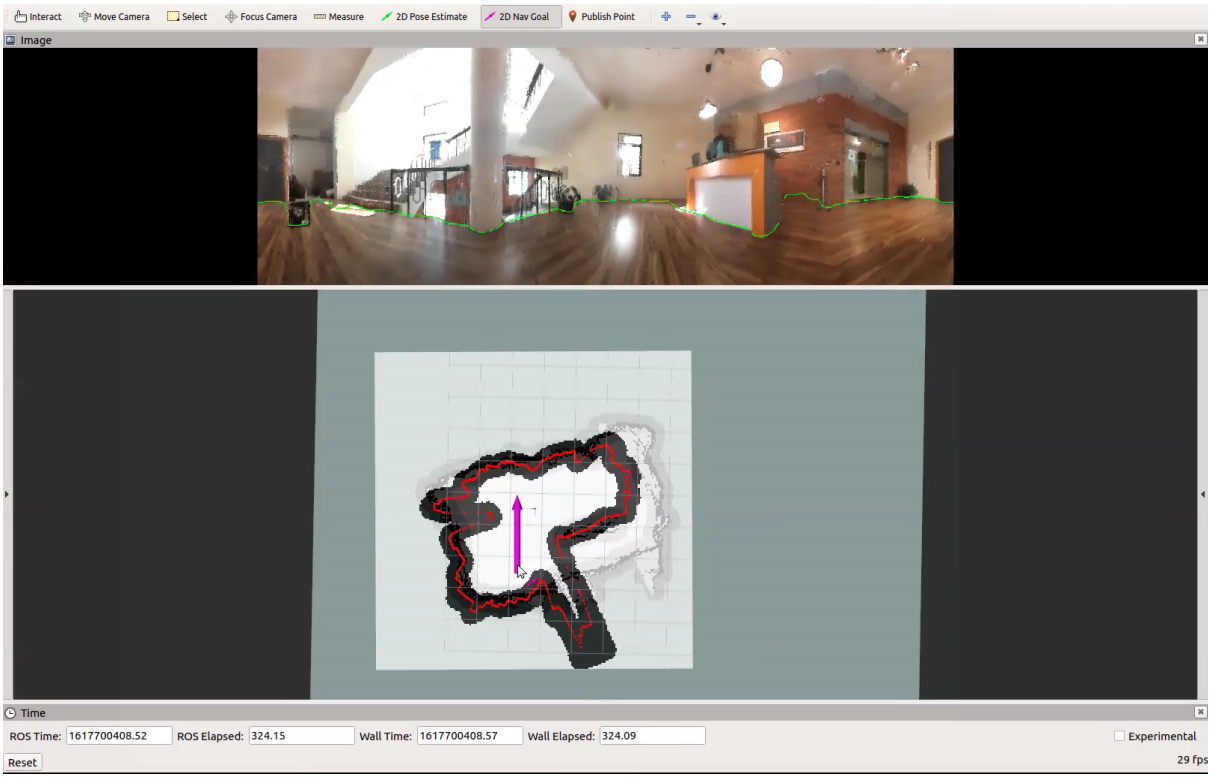


Figure 4.9 Setting the Goal Position and Target Heading for Navigation

Once the goal position is provided, the user may subscribe to the global `plan` and `footprint` of the robot to anticipate the trajectory to be followed by the robot as shown in Figure 4.10.

Once the goal position is provided, the robot will autonomously navigate and follow the anticipated trajectory to the target position, while avoiding any static or dynamic obstacles in its path that may be detected by the PAL laserscan.

In the next section, we provide an overview of the recommended parameters for AMCL and the costmaps used in our navigation tutorial. These may be used as a reference for adapting the PAL laserscan to other navigation and path-planning stacks.

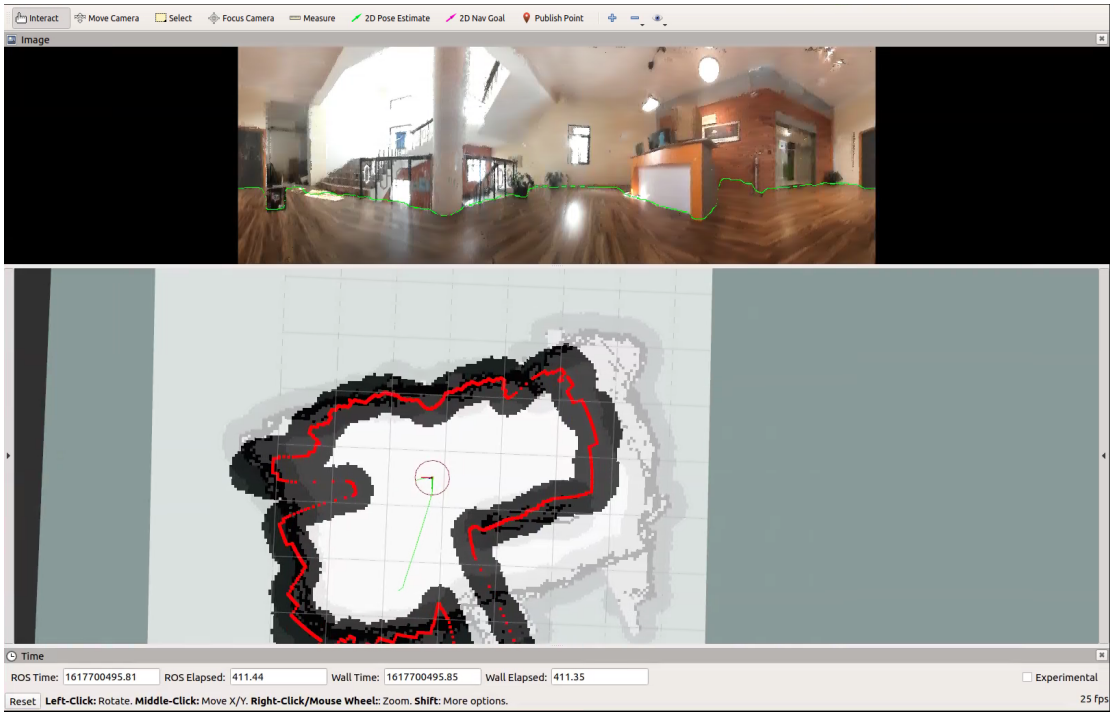


Figure 4.10 The anticipated trajectory of the robot.

2.3.3 Recommended Parameters for the Navigation Stack

Parameters for AMCL

Parameter Name	Value
odom_model_type	diff-corrected
odom_alpha1	0.002
odom_alpha2	0.002
odom_alpha3	0.002
odom_alpha4	0.002
laser_z_hit	0.1
laser_z_rand	0.9
transform_tolerance	5.0

Parameters for the Local Costmap

Parameter Name	Value
obstacle_range	2.5

raytrace_range	3.0
robot_radius	0.3
inflation_radius	0.3
cost_scaling_factor	1
transform_tolerance	10.0
global_frame	odom
robot_base_frame	base_link
update_frequency	1.0
publish_frequency	1.0
static_map	false
rolling_window	true
width	10.0
height	10.0
resolution	0.05

Parameters for the Global Costmap

Parameter Name	Value
obstacle_range	2.5
raytrace_range	3.0
robot_radius	0.3
inflation_radius	0.3
cost_scaling_factor	1
transform_tolerance	10.0
global_frame	map
robot_base_frame	base_link
update_frequency	1.0
static_map	true

Parameters for the Base Local Planner

Parameter Name	Value
controller_frequency	5.0

max_vel_x	0.10
min_vel_x	0.01
max_vel_theta	0.3
min_in_place_vel_theta	0.3
acc_lim_theta	3.2
acc_lim_x	2.5
acc_lim_y	2.5

2.4 Sample Applications

As part of the SDK users are provided with source code for sample applications. They are grouped into "tutorials" and "code samples". Tutorials are simple and complete examples, provided to quickly get started with the PAL API. They can be found in the `./tutorials/` folder. Code samples are slightly more complex and can be found in the `./code_samples/` folder.

2.4.1 Running Sample Applications

In this section we will show you how to compile and run a sample application - specifically, the 001_cv_png.cpp tutorial.

First, let us change our working directory to the tutorials folder.

```
cd tutorials/
```

To compile all the tutorials you can run the compile script:

```
./compile.sh
```

To compile just the first tutorial, run the following command:

```
g++      001_cv_png.cpp      ../lib/libPAL.so      ../lib/libPAL_CAMERA.so
../lib/libPAL_DE.so  ../lib/libPAL_SSD.so  ../libPAL_DEPTH.so  `pkg-config
```

```
--libs --cflags opencv` -g -o 001_cv_png.out -I../include/ -lv4l2
-lpthread
```

Notice that as part of the compilation libPAL.so, OpenCV, pthread and V4l2 libraries are being linked. These are the only libraries the API backend depends upon.

Checkout the first tutorial by opening `001_cv_png.cpp`, read the code and inline comments to get started. You can run the corresponding executable as follows:

```
./001_cv_png.out
```

When the executable is successfully executed, it should create 2 different files: 'left.png' and 'right.png' in the same folder. The output looks like:

Figure 4.1: Left & right panoramas saved to the file when first tutorial is run

You can follow a similar process for the other sample applications to begin your understanding of the PAL API.

3. PAL API Reference

3.1 Terminology

Let us define some terminology to help you with understanding the rest of this section.

Callee	The function being invoked
Caller	The function invoking the Callee
Write Argument	The Callee writes data into this argument, the Caller reads it.
Read Argument	The Callee reads data from this argument, the Caller provides it.
Optional Argument	This argument can be optionally not provided. When not provided it will take on a default value.

3.2 Initialization and Destruction

There are two important calls related to managing the resources associated with the camera: `Init` and `Destroy`.

3.2.1 Init Call

```
PAL::Acknowledgement Init(int& panoramaWidth, int& panoramaHeight,
int cameraIndex = -1);
```

Overview:

Initializes the PAL API backend by allocating the necessary resources and interfacing with the camera hardware. This function should be called before any other API function call.

Note: As part of the initialization CameraProperties are set to default values. You can change these camera parameters with the help of APIs discussed in section

3.4.8 and 3.4.12.

Arguments:

Name	`panoramaWidth`
Type	Write
Optional	No
Description	Width of the panoramas returned in *GrabFrames Call.*

Name	`panoramaHeight`
Type	Write
Optional	No
Description	Height of the panoramas returned in *GrabFrames.*

Name	`camera_index`
Type	Read
Optional	Yes, with default value -1

Description	Device index of the camera. (same as the index for opencv camera capture). This index provides a way of identifying different camera inputs connected to a system. When set to the default value -1, the API backend automatically tries to detect the correct index of PAL and uses it internally. In case the automatic detection does not work, or users want to explicitly specify the camera index, they can do so using this argument.

Return

PAL::SUCCESS	If initialization is successful.
PAL::FAILURE	If initialization failed. A common cause for this is that the camera is not connected.
PAL::IGNORED	If Init call is made multiple times without a Destroy call.

3.2.1.1 Determining Camera Index Manually

Every time a camera is connected to the machine a file is created in `/dev/` folder with a name like `video0`, `video1`, etc. A new file with an incremented number is created for each new camera added. This number is the camera index.

To determine which `video<No.>` file corresponds to PAL, please navigate to the `/dev/` folder. Then unplug & plug the camera from the USB port to see which `video<No.>` file is removed & added respectively. You should give the system 5-10 seconds each time you unplug / plug the camera for the file changes to take effect. Once the file is identified, the number at the end of the filename is the camera index corresponding to PAL.

3.2.2 Destroy Call

```
PAL::Destroy();
```

Overview:

Releases all the resources that were initialized as part of program execution in the API backend.

It is not required to call this function usually. The API implementation takes care of calling this function automatically at the time of application exit. However, if the user needs to call `PAL::Init` multiple number of times in the same application, then `PAL::Destroy` should be called before the next `PAL::Init` is called.

Arguments: None
Return: Void

3.3 Capture Image

This section talks about the APIs associated with image capture from the camera. First let us take a look at the `Image` structure associated with these calls.

3.3.1 Image Structure

```
struct Image
{
	union RawData
	{
		void* data;
		unsigned char* u8_data;
		unsigned short* u16_data;
		float* f32_data;
	}Raw;

	int rows;
	int cols;
	int channels;
	int bytesPerChannel;
	int stride;
	int size;
	timeval timestamp;
```

```
};
```

This structure encapsulates an image and the associated metadata. The members of the structure are explained below:

Raw	Allows users to access the start of the raw pixel memory and interprets it as different data types depending on the type of image stored
rows	Height of the image
cols	Width of the image
channels	Number of channels (for eg., 3 for RGB, 1 for depth and disparity)
bytesPerChannel	Number of bytes required to represent one channel of a single pixel (for e.g., 1 for RGB, 4 for depth)
stride	Number of bytes occupied by a single row of pixels
size	Overall size of the image in bytes
timestamp	Represents the time at which the data was captured. *Note 1*: This time instant corresponds to the moment when the first byte of data is transferred from PAL to the system. It is represented as epoch time. *Note 2*: The time difference between the real world event and the time returned in this structure will vary based on lighting conditions, camera properties, etc. which affects the exposure time. *Note 3*: For information about timeval structure please this link.

3.3.1.1 SetDimensions Method

```
void SetDimensions(int width, int height, int channelCount,
int bytesPerChannel)
```

This method allows us to set the dimensions of the image like width, height, number of channels and the number of bytes each channel takes per pixel. It also computes and sets the `stride` and `size` members of the `Image` structure from the provided arguments.

3.3.1.2 Set Method

```
void Set(void* ptr, int width, int height, int channelCount = 3,
int bytesPerChannel = 1)
```

This method is same as the `SetDimensions` method except that additionally the memory location of the raw pixel data can be set with the ptr argument.

3.3.1.3 Create Method

```
void Create(int width, int height, int channelCount = 3,
int bytesPerChannel = 1)
```

This method is same as the `Set` method except that it additionally allocates new memory corresponding to the image dimensions provided as arguments.

3.3.1.4 Destroy Method

```
void Destroy();
```

Releases the memory associated with the image structure.

3.3.1.5 Converting PAL::Image to cv::Mat

```
int width = 256;
int height = 256;
int channels = 3;
int bytesPerChannel = 1;
```

```
PAL::Image img;
img.Create(width, height, channels, bytesPerChannel);
cv::Mat m = cv::Mat(rgb.rows, rgb.cols, CV_8UC3, rgb.Raw.data);
```

Note: Please make sure that the channel depth, number of channels of the input image and the selected OpenCV image format match.

3.3.1.6 Converting from cv::Mat to PAL::Image

```
cv::Mat m = cv::Mat::zeros(256, 256, CV_16SC1);
PAL::Image img;
int channels = 1;
int bytesPerChannel = 2;
img.Set(m.data, m.cols, m.rows, channels, bytesPerChannel);
```

The set function of the `PAL::Image` structure can be used for the conversion as shown above.

Note: Please make sure that the channel depth, number of channels of the input image and the selected OpenCV image format match.

3.3.2 GrabFrames Call

```
PAL::Acknowledgement GrabFrames(PAL::Image* left,
                                PAL::Image* right,
                                PAL::Image* depth = 0,
                                PAL::Image *disparity = 0,
                                bool normalize = false,
                                bool asynchronous = true);
```

Overview:

This function helps us retrieve the latest available left, right, depth and disparity panoramas.

Arguments:

Name	`left`

Type	Write
Optional	No (But, it can be NULL)
Description	Panorama image as seen by the left view in a stereo pair. This will be a 8 bit, 3 channel image. Equivalent to CV_8UC3 of OpenCV.

Name	`right`
Type	Write
Optional	No (But, it can be NULL)
Description	Panorama image as seen by the right view in a stereo pair. This will be a 8 bit, 3 channel image. Equivalent to CV_8UC3 of OpenCV.

Name	`depth`
Type	Write
Optional	Yes, the default value is 0
Description	Panorama image representing the depth perceived by the stereo system. This image will be 32 bit (float), 1 channel. Equivalent to CV_32FC1 of OpenCV. The unit of measurement is centimeter.

Name	`disparity`
Type	Write
Optional	Yes, the default value is 0

Description	Panorama image representing the disparity perceived by the stereo system. This image will be: 8 bit, 1 channel when normalize argument is set to true. Equivalent to CV_8UC1 of OpenCV. 16 bit, 1 channel when normalize argument is set to false. Equivalent to CV_16SC1 of OpenCV. *Note:* When normalization is enabled, internally the 16 bit unnormalized information is mapped to 8 bits, such that 0 and 255 are represented by the minimum and maximum values in the current depth image respectively. As a consequence of this, it should be noted that disparity values for objects at same depth across multiple GrabFrames calls may not be the same (when normalization is enabled).

Name	`normalize`
Type	Read
Optional	Yes, the default value is false
Description	This argument enables / disables disparity normalization when it is set to true / false respectively.

Name	`asynchronous`
Type	Read
Optional	Yes, the default value is true

Description	When this argument is set to true, the call immediately returns and the last successfully computed image information is returned i.e all the images returned will not be in sync (from a single instant of time) because of differing computation speeds of individual images. When this argument is set to false, the call blocks till the current frame is completely processed and then returns the image information i.e. all the images returned are in sync (from a single instant of time).

Note: The memory location corresponding to raw pixel data written to the `left`, `right`, `depth` & `disparity` `PAL::Image` structures will contain the address to the same memory locations across multiple `GrabFrames` calls. If you wish to operate on data from more than one frame simultaneously please copy the data as required.

Return:

`PAL::SUCCESS`	If the call succeeds.
`PAL::FAILURE`	If the call fails (eg. when the USB is loose/removed).
`PAL::ERROR_CAMERA_NOT_INITIALIZED`	If this call is made before an Init call.
`PAL::IGNORED`	This can happen in the following cases: 1. If all the 4 Image arguments are null.

Note: The left and right panoramas returned by this call are always the latest, while the disparity and depth panorama are not. This is because they are computed at a different speed. The last computed depth / disparity panorama is returned at the time a `GrabFrames` call is made (if `asynchronous` is set to true).

3.3.3 Synchronize Call

```
void Synchronize();
```

Overview:

This is a "barrier call" that blocks execution until all the threads have completed computation for the latest frame.

This call is useful to force "sync" computation of panoramas when `GrabFrames` was called asynchronously. After this call is returned it is assured that no memory will be updated by PAL API until another `GrabFrames` call is made.

Arguments: None
Return: Void

3.4 Camera Properties

This section details API calls related to setting and getting different camera properties. First let us take a look at some relevant structures and enums.

3.4.1 Resolution Structure

```
struct Resolution
{
      int width;
      int height;
};
```

This structure encapsulates the width and height of the panorama image.

3.4.2 ColorSpace Enum

```
enum ColorSpace
{
      RGB,
      YUV444
};
```

This enum is used for configuring the color space of PAL.

3.4.3 PowerLineFrequency Enum

```
enum PowerLineFrequency
{
	_AUTO,
	_50HZ,
	_60HZ
};
```

This enum is used for configuring the power line frequency in camera properties.

Note: Sometimes AUTO mode cannot detect the powerline frequency correctly. When flickering is observed in AUTO mode, please choose the appropriate power line frequency based on your region. For example, in India it is 50Hz.

3.4.4 Projection Enum

```
enum Projection
{
	EQUI_RECTANGULAR = 0,
	PERSPECTIVE = 1,
};
```

This enum can be used to indicate if the PAL camera should interpret the captured panorama as an image captured using perspective projection - or using equi-rectangular projection. When the projection is set to equi-rectangular mode, the panoramas appear as if it is wrapped around a sphere. When the perspective projection is used, the panoramas appear as if they are formed by combining multiple perspective view frustums.

3.4.5 DisparityComputation Enum

```
enum DisparityComputation
{
	FAST = 0,
	HIGH_QUALITY_A = 1,
	HIGH_QUALITY_B = 2,
};
```

This enum can be used to indicate the computation mode used for disparity and depth calculation.

3.4.5 DetectionMode Enum

```
enum DetectionMode
{
       FLOOR = 1,
       INTERMEDIATE = 2,
       TABLE_TOP = 3,
       CEILING = 4,
       AUTO = 5,
};
```

This enum can be used to indicate the compute mode used for person detection.

3.4.6 CameraProperties Structure

```
struct CameraProperties
{
      int brightness;
      int contrast;
      int saturation;
      int gamma;
      int gain;
      int white_bal_temp;
      int sharpness;
      int exposure;
      bool  auto_white_bal;
      bool  auto_exposure;

      Resolution resolution;
      ColorSpace color_space;
      PowerLineFrequency power_line_frequency;

      bool  vertical_flip;
      bool  filter_disparity;
      bool  filter_spots;
      int   fov_start;
      int   fov_end;

      Projection projection;
      DispartyComputation computation;

      int camera_height;
```

```
    DetectionMode detection_mode;
};
```

This structure encapsulates the various properties of PAL. The following sections go into more detail about each of the camera properties.

Note: Please refer to the table below for the range of possible values of the camera properties.

Property	Minimum Value	Maximum Value	Valid Step Size	Default Value
Brightness	-15	15	1	0
Contrast	0	30	1	15
Saturation	0	60	1	32
Gamma	40	500	1	220
Gain	0	100	1	0
White balance temperature	1000	10000	50	5000
Sharpness	0	127	1	0
Exposure	1	10000	1	312
FOV start	0	360	1	0
FOV end	0	360	1	360

3.4.6.1 Brightness

Represents the brightness of the left and right panoramas. The minimum, maximum and default values for this property can be accessed as follows:

```
PAL::CameraProperties::MIN_BRIGHTNESS
PAL::CameraProperties::MAX_BRIGHTNESS
PAL::CameraProperties::DEFAULT_BRIGHTNESS
```

Valid values for this property should lie between these limits.

3.4.6.2 Contrast

Represents the contrast of the left and right panoramas. The minimum, maximum and default values for this property can be accessed as follows:

```
PAL::CameraProperties::MIN_CONTRAST
PAL::CameraProperties::MAX_CONTRAST
PAL::CameraProperties::DEFAULT_CONTRAST
```

Valid values for this property should lie between these limits.

3.4.6.3 Saturation

Represents the colour saturation of the left and right panoramas. The minimum, maximum and default values for this property can be accessed as follows:

```
PAL::CameraProperties::MIN_SATURATION
PAL::CameraProperties::MAX_SATURATION
PAL::CameraProperties::DEFAULT_SATURATION
```

Valid values for this property should lie between these limits.

3.4.6.4 Gamma

The minimum, maximum and default values for this property can be accessed as follows. Valid values for this property should lie between these limits.

```
PAL::CameraProperties::MIN_GAMMA
PAL::CameraProperties::MAX_GAMMA
PAL::CameraProperties::DEFAULT_GAMMA
```

3.4.6.5 Gain

The minimum, maximum and default values for this property can be accessed as follows and valid values for this property should lie between these limits.

```
PAL::CameraProperties::MIN_GAIN
PAL::CameraProperties::MAX_GAIN
PAL::CameraProperties::DEFAULT_GAIN
```

Note: When `auto_exposure` is enabled, `gain` is ignored. Also, when `auto_exposure` is disabled the `gain` is reset to the last successfully set value.

3.4.6.6 White Balance Temperature (white_bal_temp)

The minimum, maximum and default values for this property can be accessed as follows and valid values for this property should lie between these limits.

```
PAL::CameraProperties::MIN_WHITE_BAL_TEMP
PAL::CameraProperties::MAX_WHITE_BAL_TEMP
PAL::CameraProperties::DEFAULT_WHITE_BAL_TEMP
```

Note: When `auto_white_bal` is enabled, `white_bal_temp` is ignored. Also, when `auto_white_bal` is disabled the `white_bal_temp` is reset to the last successfully set value.

3.4.6.7 Sharpness

The minimum, maximum and default values for this property can be accessed as follows and valid values for this property should lie between these limits.

```
PAL::CameraProperties::MIN_SHARPNESS
PAL::CameraProperties::MAX_SHARPNESS
PAL::CameraProperties::DEFAULT_SHARPNESS
```

3.4.6.8 Exposure

The minimum, maximum and default values for this property can be accessed as follows and valid values for this property should lie between these limits.

```
PAL::CameraProperties::MAX_EXPOSURE
PAL::CameraProperties::MIN_EXPOSURE
PAL::CameraProperties::DEFAULT_EXPOSURE
```

Note 1: When `auto_exposure` is enabled, `exposure` is ignored. Also, when `auto_exposure` is disabled the `exposure` is reset to the last successfully set value.

Note 2: Exposure units are 100 microseconds i.e. a setting of 200 corresponds to 200 * 100 microseconds which is 20 milliseconds.

3.4.6.9 Auto White Balance (auto_white_bal)

When this property is set to `true`, the White Balance Temperature of the camera is automatically adjusted. The default value can be accessed as follows:

```
PAL::CameraProperties::DEFAULT_AUTO_WHITE_BAL
```

Note: When enabled, the value set for `white_bal_temp` is ignored (while setting

properties). When a `GetCameraProperties` call is made and `auto_white_bal` is enabled the last successfully set value is returned for `white_bal_temp`.

3.4.6.10 Auto Exposure (auto_exposure)

When the value of this property is set to `false` the Exposure of the left and right panoramas are automatically adjusted. The default value can be accessed as follows:

```
PAL::CameraProperties::DEFAULT_AUTO_EXPOSURE
```

Note: When enabled, the value set for `exposure` is ignored (while setting properties). When a `GetCameraProperties` call is made and `auto_exposure` is enabled the last successfully set value is returned for `exposure`.

3.4.6.11 Resolution

The resolution of the panoramas returned in `GrabFrames`. This property affects the processing that follows like disparity, depth & point-cloud. It can be one of the values mentioned below:

```
PAL::AvailableResolutions::_5290x1819
PAL::AvailableResolutions::_3544x1218
PAL::AvailableResolutions::_1322x454
PAL::AvailableResolutions::_660x227
```

The corresponding resolutions are as shown in the following table

The default value of resolution can be accessed as follows:

```
PAL::CameraProperties::DEFAULT_RESOLUTION
```

Note: In the current version of the SDK the default resolution is set to `PAL::AvailableResolutions::_5290x1819`.

3.4.6.12 Color Space (color_space)

The color space of the panoramas returned in `GrabFrames`. It can be set to any of the ColorSpace enumerations. The default value can be accessed as follows:

```
PAL::CameraProperties::DEFAULT_COLOR_SPACE
```

3.4.6.13 Power Line Frequency (power_line_frequency)

Powerline frequency is used to specify the frequency of electricity being supplied to any artificial light source illuminating the scene. This is required to correct a flickering effect observed sometimes depending on the exposure time, the frequency of the power line and the light source involved.

See PowerLineFrequency enum for more information about what values can be set for this property. The default property values can be accessed as follows:

```
PAL::CameraProperties::DEFAULT_POWER_LINE_FREQUENCY
```

3.4.6.14 Vertical Flip (vertical_flip)

This property will flip the output of the camera vertically if set to `true`. The default value can be accessed as follows:

```
PAL::CameraProperties::DEFAULT_VERTICAL_FLIP
```

3.4.6.15 Filter Disparity (filter_disparity)

This property will enable / disable filtering of raw disparity when set to `true` or `false` respectively. Filtering is done to produce a smoother disparity output. The default property value can be accessed as follows:

```
PAL::CameraProperties::DEFAULT_FILTER_DISPARITY
```

3.4.6.16 Filter Spots (filter_spots)

This property will enable / disable the reduction in spurious bright-spots in the output RGB images when set to `true` / `false` respectively. The default property value can be accessed as follows:

```
PAL::CameraProperties::DEFAULT_FILTER_SPOTS
```

Setting this property to `true` will have the following effects:

1. Post-processing is performed on the regions identified with spurious bright spots (eg: specular reflections) to minimize them. This step is performed for each output image frame as long as `filter_spots` setting is set to true.

3.4.6.17 Setting Field of View (fov_start & fov_end)

PAL allows you to select a horizontal field of view less than 360°. This can be done by setting `fov_start` and `fov_end` values, which define the start and end of the region of interest respectively.

The default property values can be accessed as follows:

```
PAL::CameraProperties::DEFAULT_FOV_START
PAL::CameraProperties::DEFAULT_FOV_END
```

Note 1: It is okay for `fov_start` to be greater than `fov_end`. This is useful, for example, when trying to select a region that wraps around the right border of the default view — i.e a region which begins close to the right edge of the default view, wraps around, and stops somewhere close to the left edge of the default view.

For example:

This is how a default panorama view looks like:

0° 90° 180° 270° 360°

If fov_start is selected as 90° and fov_end as 270° the final output you'll get will look like this:

Note 2: Disparity cannot be computed when the absolute angular difference between `fov_start` and `fov_end` is less than 2°. i.e the selected field of view is very small.

3.4.6.18 Projection

This property allows users to change the projection used when panoramas are accessed from PAL. It can be set to any of the Projections. The default property value can be accessed as follows:

```
PAL::CameraProperties::DEFAULT_PROJECTION
```

3.4.6.19 DispartyComputation

This property allows users to change the computation mode used when disparity and depth panoramas are computed. It can be set to any of the Computations. The default property value can be accessed as follows:

```
PAL::CameraProperties::DEFAULT_COMPUTATION
```

3.4.6.20 Camera Height (camera_height)

This property allows users to change the height of the camera from the floor. Height can be measured from the vertical centre of the reflective silver mirror to the floor in **centimeters**. The default property value can be accessed as follows:

```
PAL::CameraProperties::DEFAULT_CAMERA_HEIGHT
```

3.4.6.21 DetectionMode

This property allows users to change the detection mode used when person

detection API is used. It can be set to any of the Computations. The default property value can be accessed as follows:

```
PAL::CameraProperties::DEFAULT_DETECTION_MODE
```

3.4.7 CameraPropertyFlags Enum

This enum is used for selecting some or all the camera properties using bit flags. This is used in the `SetCameraProperties` call to specify which properties need to be set.

```
enum CameraPropertyFlags
{
	BRIGHTNESS = 0x1,
	CONTRAST = 0x2,
	SATURATION = 0x4,
	GAMMA = 0x8,
	GAIN = 0x10,
	WHITE_BAL_TEMP = 0x20,
	SHARPNESS = 0x40,
	EXPOSURE = 0x80,
	AUTO_WHITE_BAL = 0x100,
	AUTO_EXPOSURE = 0x200,
	RESOLUTION = 0x400,
	COLOR_SPACE = 0x800,
	POWER_LINE_FREQUENCY = 0x1000,
	VERTICAL_FLIP = 0x2000,
	FILTER_DISPARITY = 0x4000,
	FILTER_SPOTS = 0x8000,
	FOV = 0x10000,
	PROJECTION = 0x20000,
	DISPARITY_COMPUTATION = 0x40000,
	CAMERA_HEIGHT = 0x80000,
	DETECTION_MODE = 0x100000,
	ALL = 0x1FFFFF,
};
```

Note:

1. If you want to learn more about bit flags see this thread.
2. If you want to see how the enum is used in the API see SetCameraProperties

section.

3.4.8 SetCameraProperties Call.

```
PAL::Acknowledgement
SetCameraProperties(PAL::CameraProperties* properties,
unsigned int* flags = 0);
```

Overview:

Allows the user to set different camera properties.

Arguments:

Name	`properties`
Type	Read
Optional	No
Description	Contains the values of the properties to be set.

Name	`flags`
Type	Read & Write
Optional	Yes, the default value is 0. When this argument is not specified, all the properties are set.
Description	This argument is read by the function to determine the selected properties that are to be set to the provided values in `properties` argument. If any of the provided values for the selected properties are invalid, then this argument is modified by the function to indicate which properties are invalid. This argument is to be used like a bit mask created using the CameraPropertyFlags enum. Read the following subsections for examples.

Return:

PAL::SUCCESS	If the call succeeds.
PAL::FAILURE	If the call fails.
PAL::INVALID_PROPERTY_VALUED	If any of the camera properties are invalid this is returned and the flags are updated to indicate the invalid properties. The current camera Properties remain unchanged in this case.
PAL::ERROR_CAMERA_NOT_INITIALIZED	If this call is made before an Init call.

Note: The API backend computes the left and right panoramas first, then the depth panorama and lastly the point cloud. Each stage feeds into the next. Therefore, the values you set for the camera properties here will affect the entire pipeline.

3.4.8.1 Setting the flags argument

Selecting all the properties:

```
unsigned int flags = PAL::ALL
```

For selecting a single property, simply set it to its corresponding `CameraPropertyFlags` enum value. For example, to select the `saturation` property:

```
unsigned int flags = PAL::SATURATION
```

For selecting multiple properties, simply bitwise OR the corresponding `CameraPropertyFlags` enum values. To select `saturation` and `gain`:

```
unsigned int flags = PAL::SATURATION | PAL::GAIN
```

3.4.8.2 Reading the flags argument

To check if a particular property is invalid you just have to bitwise AND with the flags argument after the call returns a `PAL::INVALID_PROPERTY_VALUE` response. For example, to check if `saturation` is invalid:

```
bool isSaturationInvalid = flags & PAL::SATURATION
```

3.4.8.3 SetCameraProperties Call Examples

An example for setting all camera properties at once

```
PAL::CameraProperties p;
p.brightness = -10;
p.contrast = 10;
p.saturation = 10;
p.gamma = 50;
p.gain = 50;
p.white_bal_temp = 1500;
p.sharpness = 100;
p.exposure = 100;
p.auto_white_bal = true;
p.auto_exposure = true;
p.resolution = PAL::AvailableResolutions::_1322x454;
p.color_space = PAL::RGB;
p.power_line_frequency = PAL::_50HZ;
p.vertical_flip = true;
p.filter_disparity = false;
p.filter_spots = true;
p.fov_start = 180;
p.fov_end = 270;
p.projection = PAL::PERSPECTIVE
PAL::SetCameraProperties(&p);
```

An example for setting only saturation and gamma:

```
PAL::CameraProperties properties;
int flags = PAL::SATURATION | PAL::GAMMA;
properties.saturation = 10;
properties.gamma = 50;
PAL::SetCameraProperties(&properties, &flags);
```

An example for reading invalid flags:

```
PAL::CameraProperties properties;
properties.saturation = 2;
properties.gain = 12345;
properties.white_bal_temp = 500;

int flags = PAL::GAIN | PAL::SATURATION | PAL::WHITE_BAL_TEMP;
PAL::SetCameraProperties(&properties, &flags);
```

In the code above `gain` and `white_bal_temp` are being set to invalid values.

Therefore, after the `SetCameraProperties` call the value of the `flags` argument will be 48. Which is a combination of the bit flags `PAL::WHITE_BAL_TEMP` (32) and `PAL::GAIN` (16). This means that the following statements are also correct:

- `flags` equals `(PAL::WHITE_BAL_TEMP | PAL::GAIN)`
- `(flags & PAL::WHITE_BAL_TEMP)` will be true
- `(flags & PAL::GAIN)` will be true
- Bitwise AND operation on `flags` and any other enum of `CameraPropertyFlags` will be false.

3.4.9 SetDefaultCameraProperties Call

```
PAL::Acknowledgement
SetDefaultCameraProperties(PAL::CameraProperties* properties = 0);
```

Overview:

Resets the camera properties to default values. Optionally, the users can provide a pointer to a `CameraProperties` structure as argument to retrieve the default camera property values.

Arguments:

Name	`Properties`
Type	Read

Optional	Yes, default value is NULL
Description	When the value is not NULL, the default property values are written to the CameraProperties structure pointed to by this argument.

Return:

`PAL::SUCCESS`	If the call succeeds.
`PAL::FAILURE`	Mostly if the cable is loose/removed.
`PAL::ERROR_CAMERA_NOT_INITIALIZED`	If this call is made before an Init call.

3.4.10 GetCameraProperties Call

```
PAL::Acknowledgement
GetCameraProperties(PAL::CameraProperties* properties);
```

Overview:

Allows users to retrieve the current camera properties.

Arguments:

Name	`Properties`
Type	Write
Optional	No
Description	The retrieved camera properties are written into this structure.

Return:

`PAL::SUCCESS`	If the call succeeds.
`PAL::FAILURE`	If the call fails.

PAL::ERROR_CAMERA_NOT_INITIALIZED	If this call is made before an Init call.

3.4.11 SaveProperties Call

```
PAL::Acknowledgement SaveProperties(const char*fileName);
```

Overview:

Saves the current camera properties into a text file.

Arguments:

Name	fileName
Type	Read
Optional	No
Description	The file into which the current camera properties are to be saved. *Note 1:* If the file path does not exist, the call fails. *Note 2:* If the file exists, it is overwritten.

Return:

PAL::SUCCESS	If the call succeeds.
PAL::FAILURE	If the file can't be created with write access.
PAL::ERROR_CAMERA_NOT_INITIALIZED	If this call is made before an Init call.

3.4.12 LoadProperties Call

```
PAL::Acknowledgement LoadProperties(const char* fileName,
PAL::CameraProperties* data = 0);
```

Overview:

This call reads an existing properties file saved with the `SaveProperties` call, then loads the values into memory and also sets them on the camera by calling `SetCameraProperties` internally.

Arguments:

Name	`fileName`
Type	Read
Optional	No
Description	The file from which camera properties are to be read.

Name	`data`
Type	Write
Optional	No
Description	The structure into which the camera parameters read from file are loaded.

Return:

`PAL::SUCCESS`	If the call succeeds.
`PAL::FAILURE`	If the properties file is not found.
`PAL::INVALID_PROPERTY_VALUE`	If data in the properties file is corrupted.
`PAL::ERROR_CAMERA_NOT_INITIALIZED`	If this call is made before an Init call.

3.4.13 GetAvailableResolutions Call

```
std::vector<PAL::Resolution> GetAvailableResolutions();
```

Overview:

This call is used to access the list of valid resolutions PAL can support at runtime.

Arguments: None

Return: A `std::vector` of `Resolution` structures representing the valid resolutions that can be set on the `resolution` camera property.

3.5 Point Cloud

This section details the API calls related to point clouds. First let us take a look at the point structure.

3.5.1 Point Structure

```
struct Point
{
      float x,y,z;
      unsigned char r, g, b, a;
};
```

This structure is used to represent a point in the point cloud.

x, y & z members represent the coordinates in the x, y & z dimension respectively. Together they represent the location of a 3D point (corresponding to a 2D panorama pixel, the values will be returned in centimeters).

Note: The points more than 3km away from PAL are discarded.

r, g, b & a members represent the red, blue, green and alpha (transparency) colour values respectively.

Frame of reference: The point cloud is computed with the following frame of reference:

- X-axis points towards the first column of the left panorama image and this implies the center of the left panorama points towards the negative direction

of the X-axis.

- Y-axis is perpendicular to the base of the camera and pointing upwards.
- Z-axis can be determined by the right-hand rule.

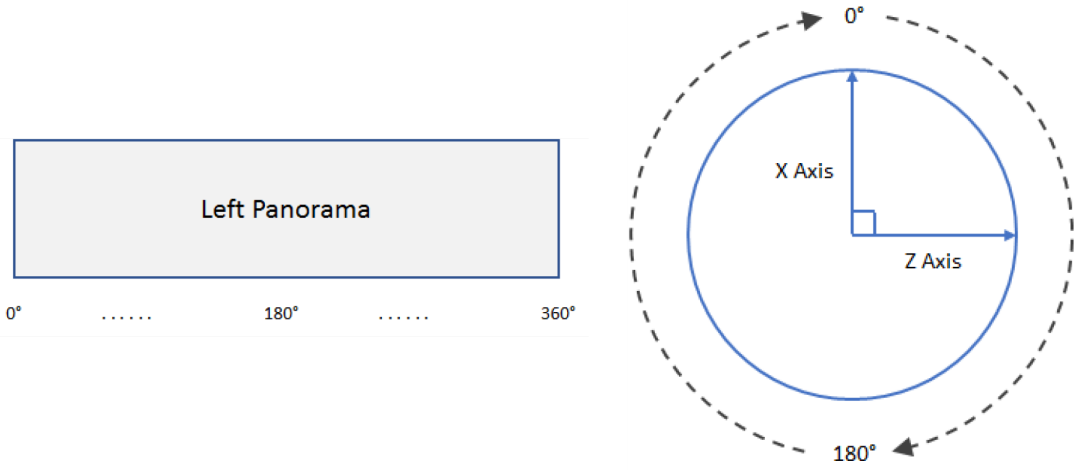


Figure 5.1: Left – Mapping columns of left panorama image into 360° horizontal field of view in the real world. Right – Top view of the left panorama wrapped around a cylinder. It also shows the X-axis mapping to 0° (i.e. start / end of the left panorama) in the real world.

3.5.2 GetPointCloud Call

```
PAL::Acknowledgement GetPointCloud(std::vector<PAL::Point> *pc,
      timeval *timestamp = 0, PAL::Image *left = 0, PAL::Image *right =
      0, PAL::Image *depth = 0, PAL::Image *disparity = 0);
```

Overview:

Allows users to retrieve the point cloud information along with optional left, right, disparity and depth panoramas. For every pixel in the panorama, its corresponding location in the 3D world the (x,y,z) position and (r,g,b) information is calculated.

This is a synchronous call. It calls `GrabFrames` internally and computes the point cloud information from the latest images. It returns only after the point cloud computation is finished for the latest frames.

These points would be pushed back into the provided vector. If the vector has prior information, that information stays intact. If you don't want the data to be accumulated, clear the vector before sending it as the argument.

Note: The Point member 'a' representing "alpha" value is set to 255 (fully opaque) for all returned points in the current implementation of the API.

Arguments:

Name	pointCloud
Type	Write
Optional	No
Description	Pointer to a valid std::vector<PAL::Point> into which the points will be pushed.

Name	timestamp
Type	Write
Optional	Yes, the default value is 0.
Description	Pointer to a valid timeval struct into which the capture time will be pushed. Time is represented as epoch time.

Name	left
Type	Write
Optional	Yes, the default value is 0
Description	Panorama image as seen by the left view in a stereo pair. This will be a 8 bit, 3 channel image. Equivalent to CV_8UC3 of OpenCV.

Name	`right`
Type	Write
Optional	Yes, the default value is 0
Description	Panorama image as seen by the right view in a stereo pair. This will be a 8 bit, 3 channel image. Equivalent to CV_8UC3 of OpenCV.

Name	`depth`
Type	Write
Optional	Yes, the default value is 0
Description	Panorama image representing the depth perceived by the stereo system. This image will be 32 bit (float), 1 channel. Equivalent to CV_32FC1 of OpenCV. The unit of measurement is centimeter.

Name	`disparity`
Type	Write
Optional	Yes, the default value is 0
Description	Panorama image representing the disparity perceived by the stereo system. This image will be 16 bit, 1 channel. Equivalent to CV_16SC1 of OpenCV.

Return:

`PAL::SUCCESS`	If the call succeeds.
`PAL::FAILURE`	It can happen in the following cases: 1. When the camera fails to capture the image. 2. When there's an internal error in point cloud computation.
`PAL::INVALID_PROPERTY_VALUE`	If this call is made before an Init call.

3.5.3 SavePointCloud Call

```
PAL::Acknowledgement SavePointCloud(const char* fileName,
std::vector<PAL::Point> *pointCloud = 0);
```

Overview:

Saves point cloud information (represented as an STL vector of Point) into a .ply file. These .ply files are recognized by 3D tools like MeshLab, Blender, etc.

Arguments:

Name	`fileName`
Type	Read
Optional	No
Description	The file into which the point cloud has to be saved. *Note 1:* The file name is appended with a .ply extension if not already provided by the user. *Note 2:* If the provided file path does not exist, then the corresponding directories are created and the file is saved in the provided path. *Note 3:* If the file already exists, it will be

	overwritten.
Name	`pointCloud`
Type	Read
Optional	Yes, default value is `NULL`
Description	The pointer to a vector of points that should be saved. When this argument is not provided or `NULL`, `GetPointCloud` function would be called internally and the latest point cloud will be saved.

Return:

`PAL::SUCCESS`	If the call succeeds.
`PAL::FAILURE`	If the file cannot be created with write access.
`PAL::IGNORED`	When the `pointCloud` vector provided is empty.

3.5.4 Point Cloud Example

```
std::vector<PAL::Point> pc;
if (PAL::GetPointCloud(&pc) == PAL::SUCCESS)
{
      PAL::SavePointCloud("point_cloud.ply", &pc);
}
```

This example demonstrates how to retrieve the latest computed point-cloud from the API and save it to a .ply file.

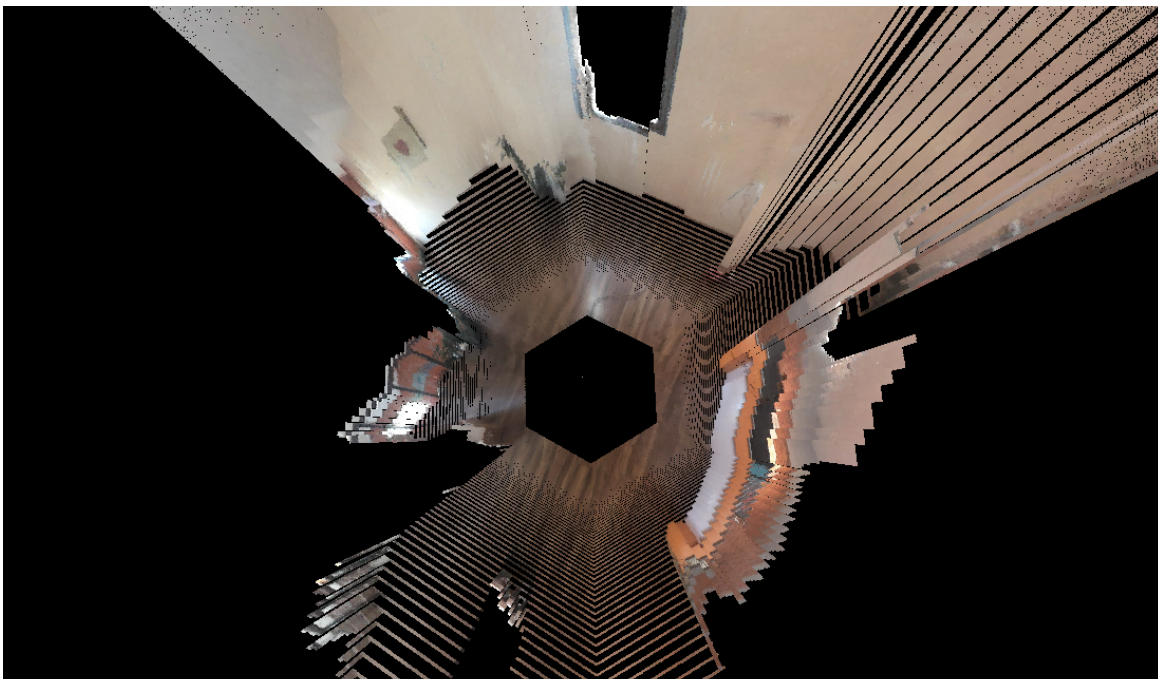


Figure 5.2: The .ply file is loaded in Meshlab and the point cloud is visualised

3.5.5 Floor Mapping Call

```
PAL::Acknowledgement SetFloorMappingParameters(int camera_height = 0);
```

Overview:

This API enables users to apply Floor mapping to Point Cloud data retrieved from APIs discussed in previous sections. It initialises resources required for floor mapping and also sets height of camera from floor required for the mapping.

Arguments:

Name	map_floor
Type	Read
Optional	Yes, default value is True
Description	If true, it enables floor mapping in pointcloud otherwise it is disabled.

Name	camera_height

Type	Read
Optional	Yes, default value is 0cm
Description	The height of the camera from the floor in cm.

Return:

`PAL::SUCCESS`	If the call succeeds.
`PAL::FAILURE`	If the floor mapping fails.

3.6 People Detection

3.6.1 BoundingBox Structure

```
struct BoundingBox
{
      int x1, y1, x2, y2;
};
```

This structure is used to represent a detection box in which the person lies. Point (x1,y1) represents the top left corner and Point(x2,y2) represents the bottom right corner of the box.

3.6.2 InitPeopleDetection Call

```
PAL::Acknowledgement InitPeopleDetection(float threshold = 0.3);
```

Overview:

Initializes the People Detection API backend by allocating the necessary resources. This function should be called after the camera is Initialized.

Arguments:

Name	`threshold`

Type	Read
Optional	Yes
Description	Defines the confidence threshold of the detections. It can be set to a float value ranging from 0 to 1.

3.6.3 GetPeopleDetection Call

```
PAL::Acknowledgement GetPeopleDetection(cv::Mat& rgb, cv::Mat& depth,
      std::vector<PAL::BoundingBox>* BoundingBoxes,
      timeval *timestamp = 0);
```

Overview:

Allows users to retrieve the image data containing rgb panorama, depth panorama and person detection data having bounding boxes of each detection. It also has optional arguments like 3D location of persons & timestamp of data captured.

It calls `GrabFrames` internally and computes the people detection information from the latest images.

Note: Please refer to code sample **006_person_detections.cpp** present in the code_samples folder of the SDK to get familiar with the structures and APIs mentioned above.

Arguments:

Name	`rgb`
Type	Write
Optional	No
Description	cv::Mat to which rgb data will be written to. Equivalent to CV_8UC3 of OpenCV.

Name	depth
Type	Write
Optional	No
Description	cv::Mat image to which depth data will be written to. This image will be 32 bit (float), 1 channel. Equivalent to CV_32FC1 of OpenCV. The unit of measurement is centimeter.

Name	BoundingBoxes
Type	Write
Optional	No
Description	Pointer to a vector of structure BoundingBox. It is populated with bounding box data of each detected person.

Name	timestamp
Type	Write
Optional	Yes, the default value is 0
Description	Pointer to a valid timeval struct into which the capture time will be pushed. Time is represented as epoch time.

Return:

PAL::SUCCESS	If the call succeeds.

`PAL::FAILURE`	It can happen in the following cases: 1. When there's an internal error in person detection computation.
`PAL::ERROR_CAMERA_NOT_INITIALIZED`	If this call is made before an Init call.

3.6.2 SetDetectionMode Call

```
PAL::Acknowledgement SetDetectionMode(PAL::DetectionMode m);
```

Overview:

Set appropriate mode for person detection API based on the position of the camera

Arguments:

Name	`m`
Type	Read
Optional	Yes
Description	Defines the mode of the detection pipeline. It can be set to the enum values.

3.7 Acknowledgement Enum

```
enum Acknowledgement
{
      IGNORED,
      SUCCESS,
      FAILURE,
      INVALID_PROPERTY_VALUE,
```

```
      ERROR_CAMERA_NOT_INITIALIZED
};
```

This enum is used to indicate if the API call has been successfully executed or not. Ideally all the API Calls should return `PAL::SUCCESS`. It is recommended to check if the return value is `PAL::SUCCESS` or not – for every API function call.

4. ROS Support

The PAL SDK supports a wrapper package ./`dreamvu_pal_camera` folder which allows you to easily integrate with Robot Operating System (ROS). This section describes this wrapper in more detail.

4.1 Building the ROS package

1. `dreamvu_pal_camera` package will be placed in the catkin_ws
2. Build the package by running the following code in a terminal.

```
cd ~/catkin_ws
catkin_make
source ./devel/setup.bash
```

4.2 Capture node

The `dreamvu_pal_camera` package has a node of `capture` type with the name `pal_camera_node`. It handles all of the publishing and subscribing responsibilities. The following table provides a list of ROS topics and the corresponding information being published and subscribed by it.

Data published	ROS Topic	Message Type
Left panorama	/dreamvu/pal/get/left	sensor_msgs::Image
Right panorama	/dreamvu/pal/get/right	sensor_msgs::Image
Depth panorama	/dreamvu/pal/get/depth	sensor_msgs::Image
PointCloud	/dreamvu/pal/get/pointcloud	sensor_msgs::PointCloud2
Data Subscribed	**ROS Topic**	**Message Type**
PAL Properties	/dreamvu/pal/set/properties	dreamvu_pal_camera::Properties

Table 8.1: List of ros topics published and subscribed by capture node

Note 1: The point cloud distance information (x, y & z data) is published in meters by the `pal_camera_node`.

4.3 Detect node

The `dreamvu_pal_camera` package has a node of `detect` type with the name `person_detection_node`. It handles all of the publishing and subscribing responsibilities of person detection data. The following table provides a list of ROS topics and the corresponding information being published and subscribed by it.

Data published	ROS Topic	Message Type
Left panorama	/dreamvu/pal/persons/get//left	sensor_msgs::Image
Depth panorama	/dreamvu/pal/persons/get/t/depth	sensor_msgs::Image
Person Detection bounding boxes	/dreamvu/pal/persons/get/detection_boxes	dreamvu_pal_camera::BoundingBoxArray

Table 4.1: List of ros topics published and subscribed by detect node

4.3.1 Bounding Message

The custom message `dreamvu_pal_camera::BoundingBox` maps directly to the BoundingBox structure in the PAL Person Detection API. It gives you the image coordinates of the person detected by the camera.

4.3.2 BoundingBoxArray Message

The custom message `dreamvu_pal_camera::BoundingBoxArray` is an array of the above custom message and contains bounding box information of all the detections.

4.4 Launching the "capture" & "detect" node

Once the package has been built as described in the previous section, to run the node, do the following. First, start the `roscore` process by running the following in your terminal:

```
roscore
```

Then, open another terminal, and source the bash setup file for that catkin workspace.

```
source ./devel/setup.bash
```

Note: This step will have to be repeated every time a new terminal is opened in the catkin workspace.

Now, run the following command to start nodes of type `capture or detect`:

```
rosrun dreamvu_pal_camera capture
```

or

```
rosrun dreamvu_pal_camera detect
```

Now, the node will advertise the list of topics mentioned in Table 4.1,4.2, to the master. Once any subscriber is registered to a topic, the node will start publishing messages corresponding to that topic.

4.5 Launching rviz preview

The `dreamvu_pal_camera` provides a launch file to preview the camera information being published in rviz.
To launch rviz for previewing the camera feed run the following command in a terminal:

```
roslaunch dreamvu_pal_camera pal_rviz.launch
```

To launch rviz for previewing the person detections feed run the following command in a terminal:

```
roslaunch dreamvu_pal_camera detect_rviz.launch
```

Running pal_rviz.launch should open an rviz window as shown below:

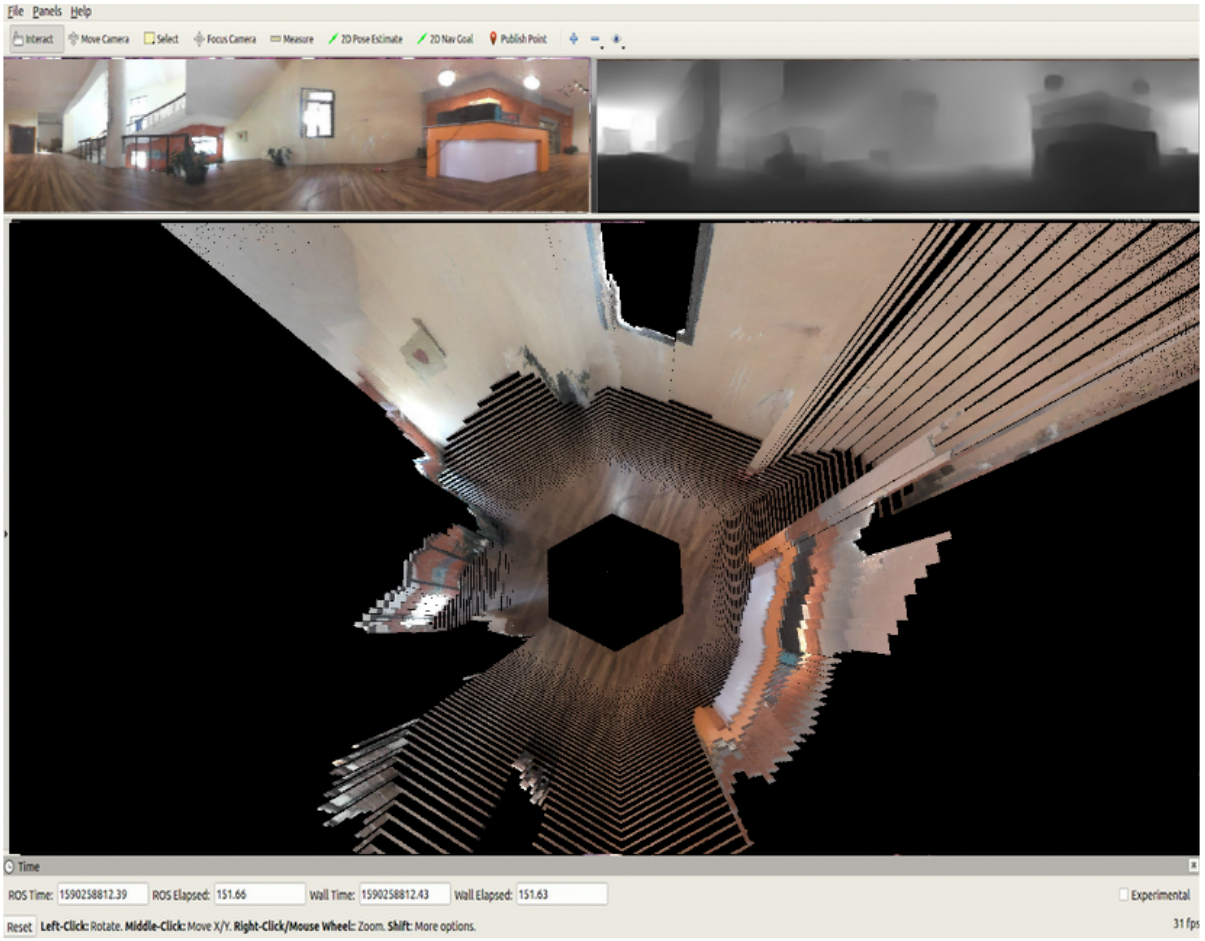


Figure 4.1: rviz preview of PAL data

4.6 Launching ROS Wrapper with Properties File

The `pal_camera_node` node launches with default properties defined in the SDK. If users want to initialize it with different values they can do so using a properties file. This file can be obtained as follows:

- File saved while Explorer application quits
- File saved using the SaveProperties call.

The `pal_camera_node` node expects a file with name and path as specified below (relative to ROS home environment variable):

```
../catkin_ws/src/dreamvu_pal_camera/src/SavedPalProperties.txt
```

Note: If the properties file is corrupted or cannot be opened then the default parameters are loaded from the SDK.

4.6.1 Copying the properties file to the default location

If you've followed the same ROS setup process as noted in this document you can just copy the `SavedPalProperties.txt` file saved by Explorer application to the following path:

```
~/catkin_ws/src/dreamvu_pal_camera/src/SavedPalProperties.txt
```

4.6.2 Explicitly specifying path to the properties file

Users can also explicitly set the file path by redefining the `PROPERTIES_FILE_PATH` macro in the pal_camera_node.cpp file. This file is located in the following path in the SDK root folder:

```
dreamvu_pal_camera/src/pal_camera_node.cpp
```

Note: The ROS package will have to be compiled again using catkin_make in this case.

4.7 Capturing ROS Bag

ROS allows users to record and replay sensor data. Please see this tutorial to understand how to capture and playback ROS bag files.

5. Benchmarking

This section details the benchmarking performed on the API and Explorer application. Particularly the following parameters are benchmarked:

1. Memory
2. Frame Rate
3. CPU Cycles
4. Latency

Performance on Reference Hardware		
	Jetson NX	Jetson AGX
Frame Rate	6 FPS	30 FPS
Latency	300 ms	200 ms
Memory Requirement	75% consumption, 5GB/7.8GB	18% consumption, 5GB/ 32GB
Compute Requirement	30% of 1.4Ghz, 6 cores, 50% GPU	15% of 1.1Ghz, 8 cores, 50% CPU
Power Mode	15W 6 core	30W ALL

5.1 System Configuration

The configuration of the system used for benchmarking is as follows:

Processor	Nvidia Jetson Xavier NX
OS Type	Ubuntu 18.04 64 bit
RAM	8 GB 128-bit LPDDR4x
OpenCV Version	4.4.0

Note: All the measurements in the benchmarking section were made with properties set to default values; However, the following exceptions apply:

- Auto exposure was disabled.
- Absolute exposure was set to 100 (i.e 10 msec).

This was done to ensure repeatability of the benchmarks under differing scene conditions.

5.2 Memory Usage

This section details the peak memory usage of the API. It was measured by using a minimal application to perform just the operations / API calls under consideration.

Operations Performed in Application	Peak Memory Usage (GB)			
	Resolution: 5290x1819	Resolution: 3544x1218	Resolution: 1322x454	Resolution: 660x227
Init and Destroy	0.8	0.8	0.8	0.8
GrabFrames for Left & Right Panoramas	1.1	1.1	1.1	1.1
GrabFrames for Left, Right & Disparity Panoramas	1.1	1.1	1.1	1.1
GrabFrames for Left, Right, Disparity & Depth Panoramas	1.2	1.2	1.2	1.2
Get PointCloud	1.30	1.25	1.25	1.25

5.3 Frame Rate

This section details the average frame rate measured under different conditions using the PAL API.

5.3.1 GrabFrames Call

This section details the average frame rate measured for different combinations using the `GrabFrames` call averaged over 50 frames.

		RGB Color Space	
	Panoramas Queried	Async. mode	Sync. mode
Resolution: 5290x1819	Left & Right	10	10
	Left, Right, & Normalized Disparity	10	1.83
	Left, Right, Normalized Disparity & Depth	9.5	1.81
Resolution: 3544x1218	Left & Right	18	20
	Left, Right, & Normalized Disparity	18	3.53
	Left, Right, Normalized Disparity & Depth	17.50	3.48
Resolution: 1322x454	Left & Right	39.95	39.52
	Left, Right, & Normalized Disparity	39.84	10.84
	Left, Right, Normalized Disparity & Depth	39.99	10.16
Resolution: 660x227	Left & Right	100.12	100
	Left, Right, & Normalized Disparity	98.57	14.19

	Left, Right, Normalized Disparity & Depth	100.10	14.01

5.3.2 GetPointCloud Call

This section details the average frame rate measured using the `GetPointCloud` call averaged over 50 calls. Please note that this call internally calls `GrabFrames` in a synchronous fashion.

	RGB Color Space
Resolution: 5290x1819	1.02
Resolution: 3544x1218	2.11
Resolution: 1322x454	5.28
Resolution: 660x227	6.95

5.4 CPU Cycles

This section details the clock cycles taken by the `GrabFrames` call for different combinations averaged over 50 calls.

The table below lists values in Millions of CPU cycles.

		RGB Color Space	
	Panoramas Queried	Async. mode	Sync. mode
Resolution: 5290x1819	Left & Right	219	224
	Left, Right, & Normalized Disparity	299	1219
	Left, Right, Normalized Disparity	340	1143

	& Depth		
Resolution: 3544x1218	Left & Right	121	125
	Left, Right, & Normalized Disparity	137	402
	Left, Right, Normalized Disparity & Depth	125	420
Resolution: 1322x454	Left & Right	55	56
	Left, Right, & Normalized Disparity	65	67
	Left, Right, Normalized Disparity & Depth	65	75
Resolution: 660x227	Left & Right	21	27
	Left, Right, & Normalized Disparity	22	57
	Left, Right, Normalized Disparity & Depth	22	61

5.5 Latency

This section lists the latency measured using a minimal OpenCV application similar to the disparity code sample provided in the SDK that makes a `GrabFrames` call to query Left, Right, Disparity & Depth panoramas.

The table below lists values measured in milliseconds

	RGB Color Space	
	Asynchronous mode	Synchronous Mode
Resolution: 5290x1819	229	644
Resolution: 3544x1218	114	344

Resolution: 1322x454	57	172
Resolution: 660x227	< 57*	114

Note: For values marked with * the latency observed was smaller than what could be measured by our setup.

5.5.1 Explorer

The rgb latency observed in the Explorer application in default mode is 229 milliseconds.

6. Known issues and troubleshooting

6.1 Unable to grab frames

This can happen when the USB cable is not properly inserted, or when the USB cable is pulled out and replaced in a different slot. You are required to keep the USB cable in the same port when the application is running. Once the cable is connected, allow the Linux OS 5-10 seconds to recognize the hardware changes, and then run the code.

6.2 Camera initialization failure

This can happen when the USB cable is inserted into the USB2.0 slot, instead of USB3.0 slot. USB2.0 is not fast enough to support the bandwidth required by PAL. You should make sure that the cable is inserted into the USB3.0 slot.

6.3 Running multiple PAL applications

Running multiple applications simultaneously is not supported by the SDK currently. When an application is communicating with PAL, and if another application tries to communicate with PAL, the behaviour is undefined. It is recommended to use a single application at a time.

6.4 Unable to find PAL .so libraries error

This error might happen when the .so file is not found in the expected path to the application. In order to resolve the issue please add the path of the lib folder of the SDK to the environment variable LD_LIBRARY_PATH. For eg:

```
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/path/to/lib/folder
```

6.5 Explorer crashes in Person detection mode when HFoV is changed

Running Explorer in person detection mode may result in a crash when a lower horizontal field of view is used.

6.6 Discontinuity in panorama when Person detection mode is

used with lower HFoV

Running person detection on a lower horizontal field of view will bring discontinuity at the end of the panorama.