Lab 5 - Laser Scanner

After implementing the tasks in the previous labs, you have the knowledge to conduct:

- creating and building ROS package,
- launching package with *rosrun* or *roslaunch* commands,
- using ROS tools to examine nodes and topics,
- sending velocity commands to the ROSbot, and
- generating the robot trajectory and velocity graphs.

In this lab, you will learn the following knowledge:

- how a laser sensor operates and how to collect the data from it.
- how to convert the collected laser data from the sensor frame to the global frame in order to build an environmental map.

There are some transformation equations being given to achieve such a conversion. In the end, you will see the generated map that is matching with the environmental model roughly. There are some deviations as the laser sensor data contains some noise.

5.1 Laser sensor and its mechanism

The laser used in this lab is RPLIDAR A3 laser scanner. Its specification can be found from the link below:

https://husarion.com/manuals/rosbot-manual/#hardware-guide

The RPLIDAR is based on laser triangulation ranging principle and adopts the high-speed vision acquisition and processing hardware developed by SLAMTEC. The system has the range readings at over 16000 times per second.

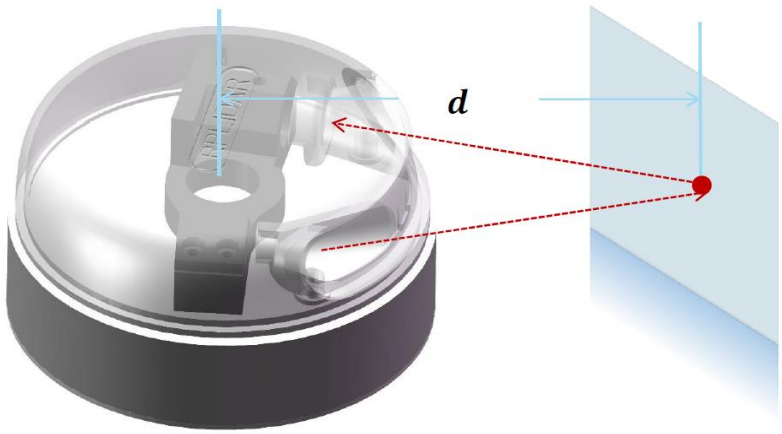


Figure 5.1 The RPLIDAR working schematic

The RPLIDAR A3M1 range scanner is driven by a DC motor and rotated clockwise to conduct the 360-degree scan for its environment. During every ranging process, the emitter of RPLIDAR sends modulated infrared laser signals, and its vision acquisition system detects the reflected signals from the object.

The returning signals are then processed by the DSP unit embedded in RPLIDAR, which outputs distance value and angle value between object and RPLIDAR via communication interface.

5.2 Laser commands and the selected five range readings

Here is the definition of the laser *msgs*:

```
Header header               # timestamp in header is the acquisition
                            # time of the first ray in the scan.
                            # in frame frame_id, angles are measured
                            # around the positive Z axis
                            # (counterclockwise, if Z is up) with zero
                            # angle being forward along the x axis
float32 angle_min           # start angle of the scan [rad]
float32 angle_max           # end angle of the scan [rad]
float32 angle_increment     # angular distance between
                            # measurements[rad]
float32 time_increment      # time between measurements [seconds]
                            # if your scanner is moving, this is used
                            # in interpolating position of 3d points.
float32 scan_time           # time between scans [seconds]
float32 range_min           # minimum range value [m]
float32 range_max           # maximum range value [m]
float32[] ranges            # range data [m] (Note: values < range_min
# or > range_max should be discarded)
float32[] intensities       # intensity data [device-specific units].
                            # If your device doesn't provide
                            # intensities, please leave the array
                            # empty.
```

When the laser subscriber returns a msg with the following features:

- msg.ranges[index] is an array of 720 range data at 0.5 degrees apart.

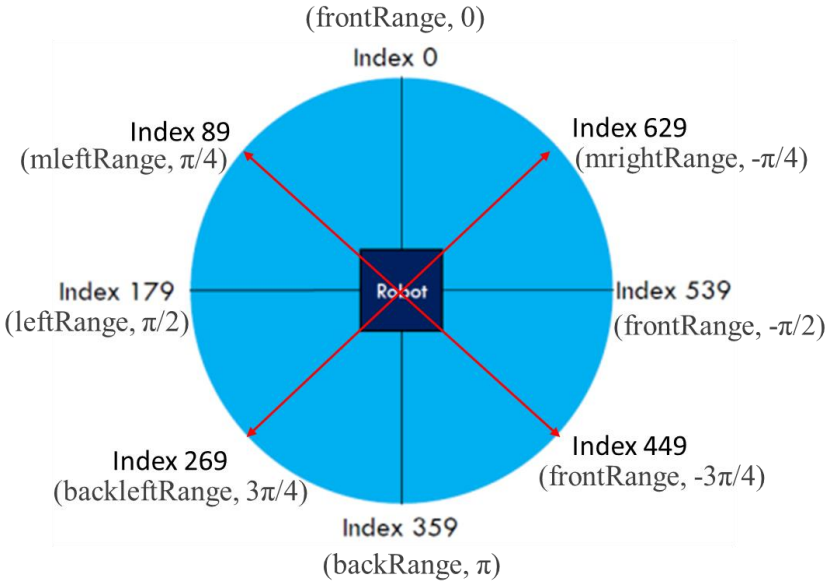


Fig. 5.2 The laser sensor has 720 range data in a full scan

At this stage, the eight laser range readings are selected for your learning process. You may use more laser range readings in your assignment in the next stage.

Task 5.1 Based on the file ***tutorial_pkg_node.cpp*** created in Lab 4, you should add the laser commands into your control code to collect laser range readings (*Range, theta*) at five angles and stored them into the .csv file for viewing and plotting.

Step 1: To include the head file '*LaserScan.h'* in your source code.

```
#include "sensor_msgs/LaserScan.h"
```

Step 2: You need to define *laserSub* in the private member of RobotMove class:

```
class RobotMove {
public:
    // Turnable parameters
    //…… some existing parameters.
    void  scanCallback(const sensor_msgs::LaserScan::ConstPtr& scan);
Private:
   ros::NodeHandle node;
   ros::Publisher commandPub;// Publisher to the robot's velocity command topic
   ros::Subscriber odomSub;     // Subscriber to robot's odometry topic
   ros::Subscriber laserSub;      // Subscriber to robot's laser topic

    // some existing code

    double frontRange=0,mleftRange=0,leftRange=0,rightRange=0,mrightRange=0;
    double backRange=0, backleftRange=0, backrightRange=0, laserData[36];
    double frontAngle=0, mleftAngle=M_PI/4, leftAngle=M_PI/2;
    double rightAngle=-M_PI/2, mrightAngle=-M_PI/4;
    double backAngle=M_PI, backleftAngle=3*M_PI/4, backrightAngle=-3*M_PI/4;
}
```

Step 3: You need to subscribe to the laser topic '/scan'.

```
RobotMove::RobotMove(){
   //Advertise a new publisher for the simulated robot's velocity command topic at 10Hz
   commandPub = node.advertise<geometry_msgs::Twist>("cmd_vel", 10);
   odomSub = node.subscribe("odom", 20, &RobotMove::odomCallback, this);
   laserSub = node.subscribe("scan", 1, &RobotMove::scanCallback, this);}
```

Step 4: to add the '*scanCallback*' function behand "odomCallback()" function.

```
void RobotMove::scanCallback(const sensor_msgs::LaserScan::ConstPtr& scan){
    // collect 36 laser readings every 360 degrees scan
    for(int i=0; i<36; i++)    // to get 36 laser readings over 360 degrees
        laserData[i] = scan->ranges[i*10]; // to get laser readings every 10 degrees
   // the following code for the control purpose
   frontRange = scan->ranges[0];    // get the range reading at 0 radians
   mleftRange = scan->ranges[89];   // get the range reading at -π/4 radians
   leftRange = scan->ranges[179];   // get the range reading at -π/2 radians
```

```
    rightRange = scan->ranges[539]; // get the range reading at π/2 radians
    mrightRange = scan->ranges[629]; // get the range reading at π/4 radians
    backRange = scan->ranges[359];   // get the range reading at π radians
    backleftRange = scan->ranges[269]; // get the range reading at π/2 radians
    backrightRange = scan->ranges[449]; // get the range reading at π/4 radians
}
```

Step 5: Finally, you need to add the following code into ***"void RobotMove::startMoving()".***

```
void RobotMove::startMoving(){

    // some existing code

    ofstream laserFile = openFile("laserData.csv");

    ros::Rate rate(20);   //Define rate for repeatable operations.
    ROS_INFO("Start moving");
    while (ros::ok()){ // keep spinning loop until user presses Ctrl+C
      switch(stage){
        case 1:
          // some exist code
          // .....
        case 6:
          moveStop();
          break;
      }
      // some existing code

      for(int i=0; i<36; i++)     // save laser data for view and check
          laserFile <<  i  << " " << laserData[i];
      laserFile <<  endl;
      ros::spinOnce();
      rate.sleep();
    }
    odomTrajFile.close();
    odomVelFile.close();
    laserFile.close();
}
```

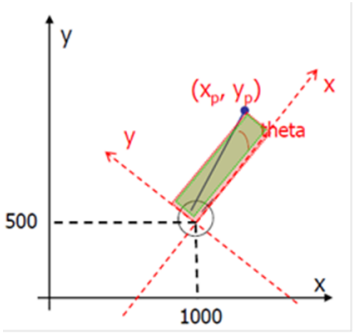
Step 6: To compile and run your code.

```
cd  ~/M-Drive/ros_workspace
catkin_make
source devel/setup.sh
roslaunch tutorial_pkg tutorial_rosbot.launch
```

Step 7: Using LibreOffice in Ubuntu or Excel to view the laser scanning data in the csv file.

5.3 Polar to Cartesian for building a dynamic environment map

As shown in Figure 5.3, the laser readings use Polar coordinates, (*Range*, *theta*), in a sensor frame that is centred at the robot local frame. On the other hand, and the robot location uses Cartesian coordinates, (*X*, *Y*) in a global frame.



Laser data:

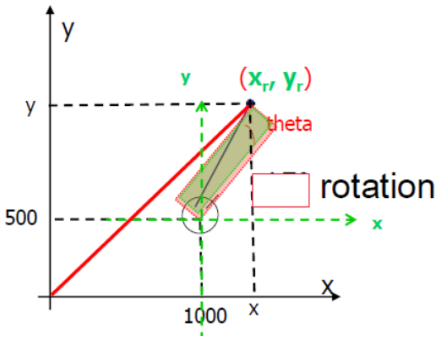
from (Range, Angle) → (Xp, Yp)

i.e. Polar → Cartesian coordinates

Xp = Range * cos(Angle)

Yp = Range * sin(Angle)

Figure 5.3 From laser Polar data to the robot coordinate system.



Rotation and transformation

i.e. from (Xp, Yp) → (Xr, Yr)

Xr = (Xp*cos(theta) – Yp*sin(theta))+RobotX

Yr = (Xp*sin(theta) + Yp *cos(theta))+RobotY

Figure 5.4 From the robot coordinates to Global coordinates

To build a dynamic environment map, you should translate your laser readings from Polar coordinates, (*Range*, *theta*), in the robot Cartesian frame (x_p, y_p), and then to the global Cartesian coordinates (x_r, y_r) for the purpose of robot navigation and obstacle avoidance.

Figure 5.3 shows how to convert the laser readings from Polar format to the robot coordinate system. Figure 5.4 shows two equations used for converting the laser data from the robot coordinates (x_p, y_p) to the global coordinates (x_r , y_r).

Note that *Theta* is the heading angle of the robot in the global coordinate system. *theta* is the angle between the laser range and the robot's heading direction.

It should be noticed that the heading angle of the robot, *Theta*, is not available from the Gazebo simulator. You need calculate it by using the orientation (*x*, *y*, *z*, *w*) of the robot that is available in the message published from the '/odom' topic.

After the head angle of the robot, *Theta*, is obtained, you can use it to do the calculations of equations in Figure 5.3 and Figure 5.4 to obtain the Cartesian coordinates of the laser range, (x_r , y_r), each time you have obtained a laser reading data.

Task 5.2: To add the new code into *tutorial_pkg_node.cpp* so that you can obtain the head angle of the robot, *Theta*, and then use it to do the calculations of equations in Figure 5.3 and Figure 5.4 to obtain the Global Cartesian coordinates of the laser range, (x_r , y_r) each time a

laser reading data is obtained. Finally, you should store the calculated the Cartesian coordinates of the laser readings into the csv file for viewing and plotting.

Step 1: To include the following code at the beginning of your code for the calculation of *Th*:

```
using namespace std;

struct EulerAngles{
    double roll, pitch, yaw;};     // yaw is what you want, i.e. Th

struct Quaternion{double w, x, y, z;};

EulerAngles ToEulerAngles(Quaternion q)
{
    EulerAngles angles;
    // roll (x-axis rotation)
    double sinr_cosp = +2.0 * (q.w * q.x + q.y * q.z);
    double cosr_cosp = +1.0 - 2.0 * (q.x * q.x + q.y * q.y);
    angles.roll = atan2(sinr_cosp, cosr_cosp);
    // pitch (y-axis rotation)
    double sinp = +2.0 * (q.w * q.y - q.z * q.x);
    if (fabs(sinp) >= 1)
        angles.pitch = copysign(M_PI/2, sinp); //use 90 degrees if out of range
    else
        angles.pitch = asin(sinp);
    // yaw (z-axis rotation)
    double siny_cosp = +2.0 * (q.w * q.z + q.x * q.y);
    double cosy_cosp = +1.0 - 2.0 * (q.y * q.y + q.z * q.z);
    angles.yaw = atan2(siny_cosp, cosy_cosp);
    return angles;
}
```

Step 2: To add the following code into ***'Class RobotMove{ }'***

```
class RobotMove {
public:
      // Turnable parameters
     // some existing code.
     void transformMapPoint(ofstream& fp, double laserRange, double laserTh)
private:
     // some existing code
      Quaternion robotQuat;
      EulerAngles robotAngles;
      double robotHeadAngle;
}
```

Step 3: To add the following function to transform the points to global coordinates.

```
void RobotMove::transformMapPoint(ofstream& fp, double laserRange, double laserTh)
{
    double localX, localY, globalX, globalY;
```

```
    localX = laserRange * cos(laserTh);
    localY = laserRange * sin(laserTh);
    globalX =(localX*cos(robotHeadAngle)–localY*sin(robotHeadAngle))+ PositionX;
    globalY = (localX*sin(robotHeadAngle)+localY*cos(robotHeadAngle))+ PositionY;
    if (globalX < 0) globalX = 0; else if (globalX > 2.5) globalX = 2.5;
    if (globalY < 0) globalY = 0; else if (globalY > 2.5) globalY = 2.5;
    fp << globalX << "  " << globalY << endl;
}
```

Step 4: to add the following code into the callback function '*odomCallback*':

```
void RobotMove::odomCallback(const nav_msgs::Odometry::ConstPtr& odomMsg)
{
    // some existing code .....
    robotAngles = ToEulerAngles(robotQuat);
    robotHeadAngle = robotAngles.yaw;
    robotQuat.x = odomMsg->pose.pose.orientation.x;
    robotQuat.y = odomMsg->pose.pose.orientation.y;
    robotQuat.z = odomMsg->pose.pose.orientation.z;
    robotQuat.w = odomMsg->pose.pose.orientation.w;
}
```

Step 5: Finally, you need to add the following code in Yellow to ***"void***

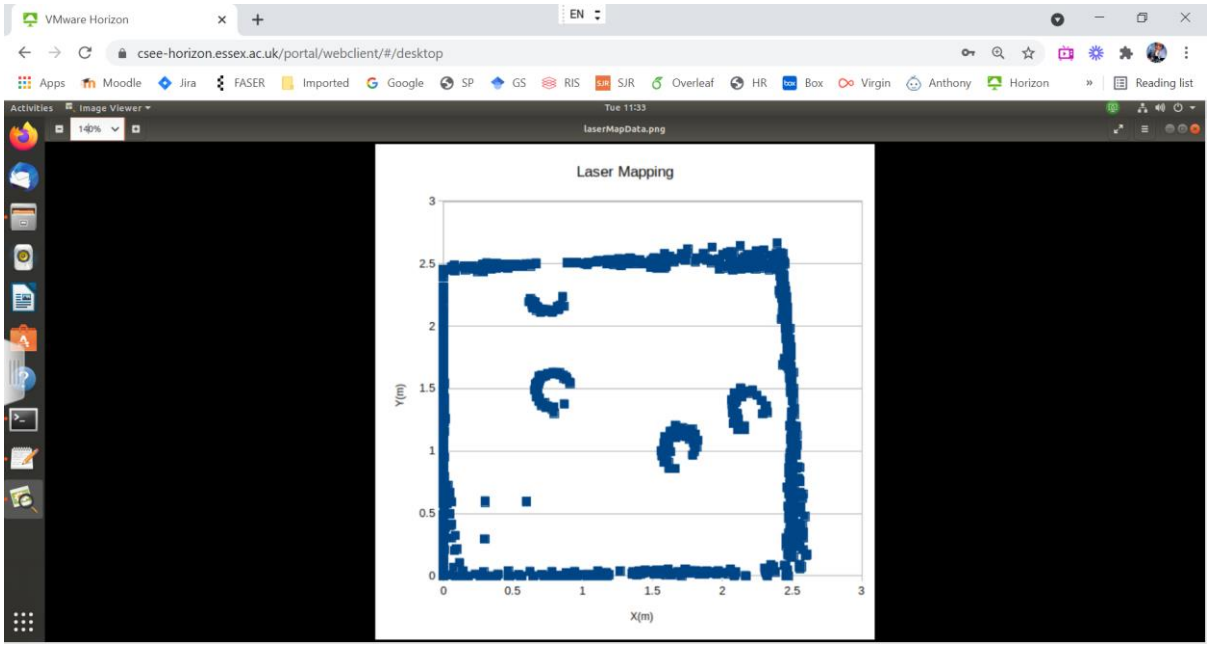
```
void RobotMove::startMoving(){
    // some existing code
    ofstream laserMapFile = openFile("laserMapData.csv");
    ros::Rate rate(20);   //Define rate for repeatable operations.
    ROS_INFO("Start moving");
    while (ros::ok()){ // keep spinning loop until user presses Ctrl+C
      switch(stage){
            case 1:
               // some exist code
               // ……
            case 6:
               moveStop();
              break;
    }
    // some existing code……
    transformMapPoint(laserMapFile, frontRange, frontAngle);
    transformMapPoint(laserMapFile, leftRange, lefttAngle);
    transformMapPoint(laserMapFile, rightRange, rightAngle);
    transformMapPoint(laserMapFile, mlefttRange, mleftAngle);
    transformMapPoint(laserMapFile, mrightRange, mrightAngle);
    ros::spinOnce();
    rate.sleep();
```

```
    }
    // some existing code
     laserMapFile.close();
}
```

Step 6: To compile and run your code.

```
cd  ~/M-Drive/ros_workspace
catkin_make
source devel/setup.sh
roslaunch tutorial_pkg tutorial_rosbot.launch
```

Step 7: Using LibreOffice in Ubuntu or Excel to view the laser map data in the csv file.



5.4 To handle infinity laser data

If you get infinity range problem with the laser in Gazebo, you should take the actions:

- Go to *~/M-Drive/ros_workspace/src/rosbot_description/src/rosbot_description/urdf*
- Using gedit to open the file: *rosbot.gazebo*
- Find *'<sensor type="gpu_ray" name="head_rplidar_sensor">'*,

  replace it with *' <sensor type="**ray**" name="head_rplidar_sensor">'.*
- Find *<plugin name="gazebo_ros_head_rplidar_controller" filename="libgazebo_ros_gpu_laser.so">*,

  replace it with *'<plugin name="gazebo_ros_head_rplidar_controller" filename="**libgazebo_ros_laser.so**">'*

Reference:

[1] Wikipedia, Conversion between quaternions & Euler angles. Accessed on 11/11/2020. https://en.wikipedia.org/wiki/Conversion_between_quaternions_and_Euler_angles.