robot_alog contains all the necessary packages to function Alog bot and they are mentioned below with brief details :

- alog_control :
    1. Include
    2. launch :
        1) *odom_custom.launch :* Launches necessary nodes required to publish odom data on **/odom** topic.Like **holonomic_odometry_node** and **imu_node .**
        2) *traj_trac.launch :* Launches nodes required for trajectory generation and tracking like **mecanum_waypoints** , **mecanum_controller** and also launches necessary launch files for **/odom** topic**.**
        3) *joystick.launch :* Launches launch files like **motor_drivers.launch** and **static_tf.launch** necessary for making bot move. Along with it launches nodes like **joy_node** from joy pkg (std pkg) with name of port as argument and also launches **holonomic_joystick_driver_node .**
        4) *startup.launch :* Launches launch file **joystick.launch** alongwith nodes like **light_controller** ,**ultrasonic_node** ,**wheel_light_controller** and **alog_launch_controller** also passes some parameter files like **robot_params.yaml** ,**embedded_params.yaml** and **wheel_light_config.yaml** from *config_package/config* .
        This launch file is launched by default at the system start using **ros_launch.sh** located in *home/startup/ .*

    3. Rviz
    4. scripts
        1) *base_move.py*
        2) *go_to_goal.py* : Consist of go-to goal controller based on PID controller . Publishesh velocity commands on **/cmd_vel** topic of Twist type. Commands of velocity is being calculated by minimizing error in position, which is obtained with the help of feedback from **/odom** topic data. All calculations of vx , vy and vw from 3 independent PID equations are done in Subscriber callback function **odometry_data_callback()** ,of /odom topic.Hence we publish velocity commands at the rate of 7-8 Hz (Hardware dependent).We can specify single Goal position or Multiple Goal position one after the other.
        3) *launch_control.py*
        4) *vel_pub.py*
        5) *vel_r_data.csv*
        6) *vel_y_data.csv*
        7) *waypoints_controller.py :* Executable creates a node that is responsible for discretizing trajectories specified by equations in terms of time or called parametric equations like x=A*cos(wt) and y=A*sin(wt). It publishes local waypoints sampled from these equations on topic **/waypoints_to_follow** of type Vector3 at the rate 50 Hz(optimal for the

Hardware used).Also publishes points and velocities sampled from the equations used on topic **/calc_traj_plots** of type Odometry for data Plotting purposes .Sample Time used here is 0.1 sec(optimal for velocity constrained 0.5 or tuned gains of PID).Elapsed value of time (**t**) is calculated at the rate of 100 Hz.

8) *waypoints_controller_traj_gen.py* ***:***
9) *waypoints_traj_controller.py :* A controller similar to go to goal ,based on PID . But it is written for tracking continuous streams of local goal points which are sampled from Trajectory equations. Here we have two subscriber topics namely **/waypoints_to_follow** (of type Vector3)and **/odom** (of type Odometry) .And publishes velocity commands on topic **/cmd_vel** (of type Twist) in body frame at the rate same as the data on topic /waypoints_to_follow gets publishes.We use two files for publishing local waypoints on topic /waypoints_to_follow for different purposes.**waypoints_controller.py** and **waypoints_controller_traj_gen.py** .

5. Src
6. CMakeLists.txt
7. package.xml

- config_package :
- embedded_communication :
- holonomic_controller :
    1. Include
        1) *holonomic_controller_node.h* : contains definition of publisher object **speed_pub** and and String object msg to publish on **/distributed_speed**.Along with definition of constants related to physical dimensions of bot and angular wheel speed **w1,w2,w3,w4**.
        2) *holonomic_odometry.h* : contains definition of some constant specific for pose and speed calculation using encoder data and encoder specific constants. And also definition of **odom_pub** a publisher object to publish odom data.

    2. Scripts
        1) *holonomic_joystick.py :* A script commands velocity msg to topic **/cmd_vel** of type Twist according to subscriber callback function of topic **/joy** which has msg type Joy.Just converts button inputs to velocity cmds. Joy is defined as (from **sensor_msgs.msg**)

            std_msgs/Header header
            Float32[ ] axes
            Int32[ ] buttons

            It also publishes Bool msg True on topic /navigation_button_pressed if navigation mode is active.
        2) *motor_driver_node.py* : It contains all hardware communication code which is serial communication with **Leadshine Servo Motor Driver .**It

subscribes to **/distributed_speed** topic and extracts velocity according to the motor ID or name then passes the motor driver.It publishes encoder data on topic **/motor_+motor_name+/encoder** of Int64 type and also publishes alarming msg on topic **/motor_+motor_name+/alarm** related to particular motor of String type,for example alerts for high current or voltage applied !! ,Encoder error !! .

This script has some parameters motor specific given in **config_package/config/motor_driver_param.yaml** (max_speed_p, acceleration etc) . This script is written in a way if launched multiple times using name of motor as parameter it will work for others motor drivers too. We use timer to continuously update encoder readings in intervals of 0.13 sec and check for alarms in interval of 1.1 sec . And commands velocity to motor driver every instant the **/distributed_speed** gets updated.

It also subscribes to **/emergency** topic for emergency conditions and interrupting motor driver in between.It has some services also ,like **/motor_+motor_name + /speed ,/motor_+motor_name+/voltage ,/motor_+motor_name+/write_register** and **/motor_+motor_name+/read_register .**

It uses some custom defined msg or service formates like Speed , Voltage ,VoltageResponse ,Buttons ,Readregister etc located in **embedded_communication/msg** and **motor_driver/srv** .

3. Src
    1) *holonomic_controller_node.cpp* : It consist a node **holonomic_controller** which subscribes to topic **cmd_vel** which gives velocity in bots vx,vy and angular speed about z axis and calculates angular velocity **w1,w2,w3** and **w4** using inverse kinematic model of mecanum drive.It includes header file **holonomic_controller_node.h** .It publishes calculated velocities to **/distributed_speed** topic as a msg of string type.
    2) *holonomic_odometry.cpp* : It subscribes to all four encoder topics **/motor_+motor_name+/encoder (motor_name={front_left,front_right,back_left,back_right})** and **/orientation** topic for absolute theta .They all have their callback functions to update respective data.We have timer function to recurse with interval of 0.1 second and calculate a odom data and publishes msg to **/odom** topic of Odometry type. It calculates vx,vy,w in bots frame using forward kinematic model and then it transform their frame in order to perform global x ,y, theta. It also calculates and broadcasts transform of odom data to base_link data.

    Everytime you launches this node along other its dependencies odom frame changes i.e it initializes origin at that point only.

    We will have an actual lateral traveling distance is shorter than actual forward traveling distance. It means that the wheel slippage affects very much when the robot moves laterally.So to compensate that we use multiplication factor with vy while calculating it using encoder data.

It includes **holonomic_odometry.h** header file located in **holonomic_controller/include** .

3) *lateral_diff_calculation.py* : The calculation of the Multiplication factor is given here and we can calculate that using this script for different coefficient of friction.

4. CMakeLists.tx
5. package.xml

- navigation_layers :
- rplidar_ros :
- sensors_package :
    1. launch :
        1) *hardware.launch :* Launches multiple launch files or all hardware configuring node files like **lidar.launch** ,**realsense.launch**,**laser_filter.launch**.**odometry_node** and **imu_node .**
        2) *laser_filter.launch :* Launches **laser_filter** node from **laser_filters** package with config file **laser_filters.yaml** located in config_package/config/laser_filters.yaml also with remapping arguments to remap topics.
        laser_filter node takes **/scan** topic data in and publishes filtered scan on **/scan_filtered** topic.
        3) *lidar.launch :* A launch file launches **rplidarNode** with all required parameters for establishing serial communication and reading data from it. Parameters are :
        serial_port,serial_baudrate,frame_id,inverted,angle_compensate,scan_mode
        4) *motor_drivers.launch :* A launch file launches **back_left_driver_node,back_right_driver_node,front_left_driver_node,front_left_driver_node** and **holonomic_controller_node** with considering parameters from config_package/config/motor_driver_param.yaml also some acceleration and deceleration arguments are provided explicitly.
        5) *pcl_to_laser.launch :*
        6) *realsense.launch*
        7) *static_tf.launch* : Launches **static_transform_publisher** node from **tf** package for different frames like btw base_link and camera_laser ,base_link and imu_link and other combinations with their location as arguments.

    2. scripts :
        1) *imu.py :* Contains hardware interfacing code ,includes serial communication with port for receiving sensor data. Publishes yaw values which are Float32 type msg on **/orientation** topic and also publishes IMU sensor data of type IMU (msg) on **/sensors/imu** topic .IMU msg consists :
        std_msgs/Header header

geometry_msgs/Quaternion orientation
float64[9] orientation_covariance
geometry_msgs/Vector3 angular_velocity
float64[9] angular_velocity_covariance
geometry_msgs/Vector3 linear_acceleration
float64[9] linear_acceleration_covariance

        2) *realsense_dynamic_configure.py*
    3. CMakeLists.txt
    4. package.xml
- step_back_recovery :
- stop_recovery :