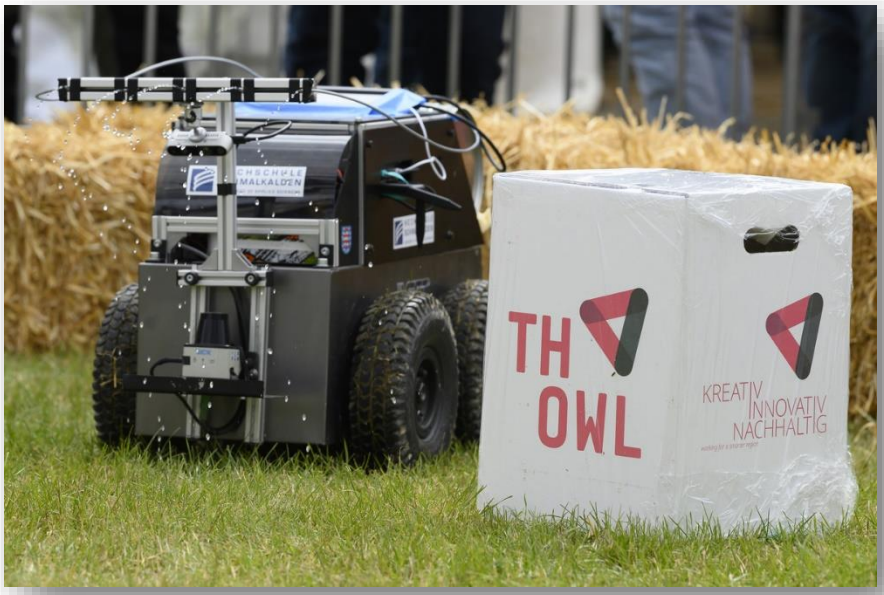
Field Robot Event 2024
Robot Documentation

Index:

1. Grasshopper Autonomous Mobile Robot:

- **Overview**
  Grasshopper is a modular autonomous mobile robot designed and developed for various outdoor applications, with a particular focus on agricultural tasks. The robot's capabilities were put to the test at the Field Robot Event 2024, where it demonstrated its effectiveness in handling complex agricultural environments.



- **Drive System**
  Grasshopper features a 4-wheel drive skid steering system, powered by four independent brushless in-runner motors. Each motor is coupled with a reduction gearbox to enhance torque while reducing speed. The gearbox employs a two-stage reduction mechanism:
  - First Stage: Helical spur gears.
  - Second Stage: Straight spur gears.
  - Total Reduction Ratio: 1:9.67 (one tire rotation equals 9.67 motor shaft rotations).

  The combined theoretical power output of the robot is **40 horsepower**.

- **Structural Design**
  - Base Plate and Top Plate: Constructed from 3mm mild steel.
  - Side Walls: Made of 1.5mm aluminum sheets.
  - Frame Extrusions: Utilize 30mm aluminum profiles for robust structural integrity.
  - Top Enclosure: Fabricated from 4mm plastic sheets, precisely cut to shape.

- **Motor Control**
  Each motor is controlled by its own VESC (Vedder Electronic Speed Controller), connected via serial ports. The VESC units are configured using the VESC Tool app, which also manages the motors' AS5047p encoders. These encoders are mounted on the back of each motor and connected to the sensor ports of the VESCs for precise control and feedback.

- **Odometry and Sensor Fusion**
  Grasshopper utilizes the inbuilt IMU (Inertial Measurement Unit) of the VESCs for odometry. The IMU is calibrated through the VESC Tool app. A Kalman filter-based sensor fusion algorithm integrates data from the encoders, IMU, and additional laser sensors to ensure accurate localization and navigation.

- **Perception Systems**
  To navigate and perceive its environment, Grasshopper is equipped with:
  - 2D Lidar: SICK Tim 561 for horizontal scanning and obstacle detection.
  - Depth Camera: Intel RealSense D435i for depth estimation and object detection.

- **Mobility**
  The robot moves on pneumatic tires with a diameter of 250mm, providing the necessary traction and shock absorption for outdoor terrains.
  Grasshopper's design and capabilities make it a versatile solution for various agricultural tasks, from precision farming to field monitoring, enhancing efficiency and productivity in outdoor environments.

2. Team Members and List of Task:

Mayank Yogesh Khandelwal	**Santosh Ashok Konduskar**
Swaraj Tendulkar	**Onkar Vilas Repe**
Rahul Eknath Kadam	**Jovan Oliveira**
Dhaval Vijaybhai Lad	**Prakash Dhirubhai Makvana**
Harsh Somani	**Shlok Khambhati**
Prashant Thommandra	

FRE 2024 Task:

- **Problem Statement**
- **Website**
- **Drive Link**

3. Sponsors (Contacts):

Company Name	**Contact Person**	**Email Id**
Applyo Jena gmbh	**Dr. Thanh Tu Hellmich-Duong** CTO · Managing Partner	t.hellmich-duong@applyo-jena.com
Delivery-Me	**G. Alexander Kolbai** CEO · Owner	akolbai@exxternity.eu
SICK AG	**Kai Schakau** Sales & Service · SICK Vertriebs	kai.schakau@sick.de
Trampa Boards	**Ted Orr** **Proprietor**	ted@trampaboards.com

4. Design:

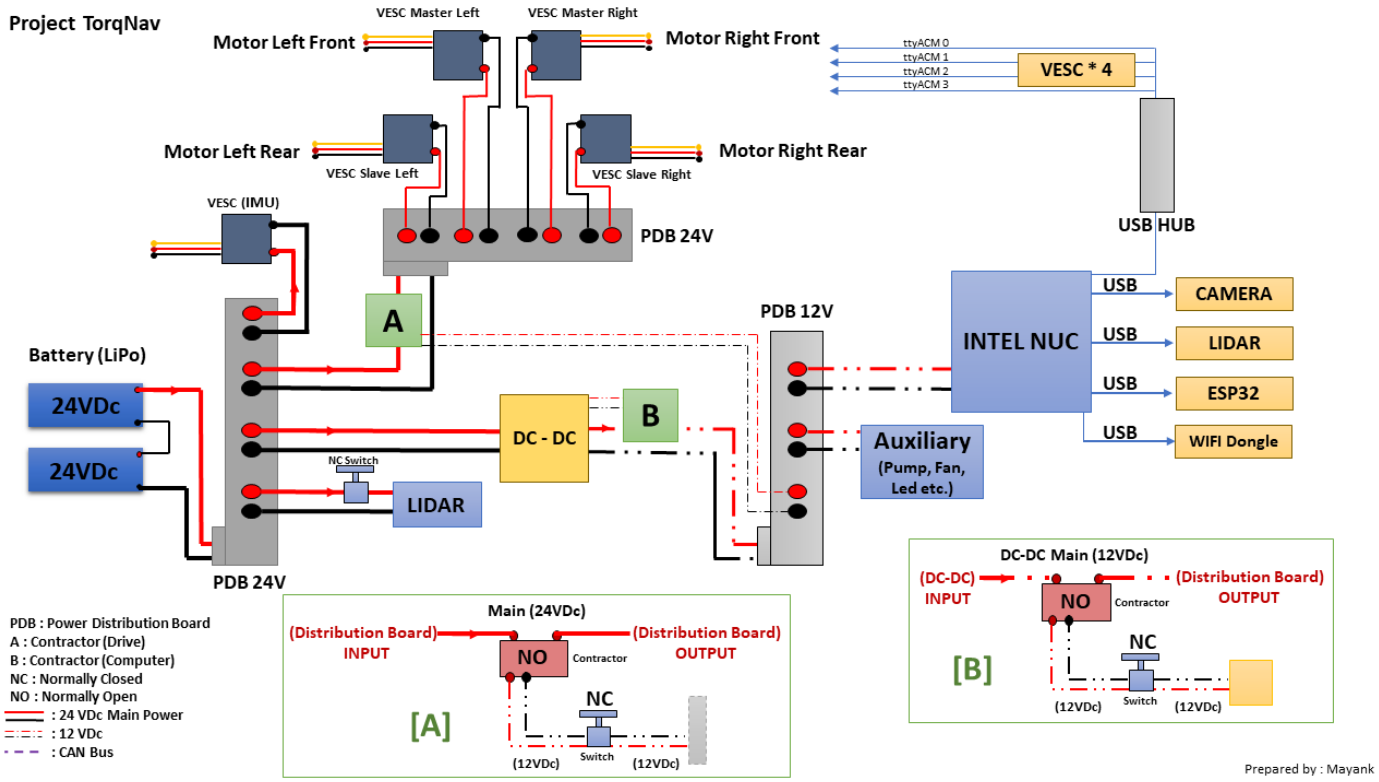
Grasshopper CAD Files:

CAD Files

5. Components:

Component Name	Use	Quantity	Buying Ref Link
INTEL NUC 11 Nuvo-9000 Series	Computer	1	
VESC 6 MKVI	Motor Drivers	4	Trampaboards
Innova Tractor Tread 10”	Pneumatic Wheels	4	
Primo 3 Spoke Comp. Hub	Wheel hubs	4	
AS5047P	Encoders	4	Amazon
SICK Tim 561	2D Laser Scanner	1	Sick
Intel Real Sense	RGB Depth Camera	1	
CNHL LIPO	Battery	2	Ebay
10 AWG Silicon Wires	Wire	-	
Rocker + Emergency Switch	Toggle Switch	3	Amazon
DC Pump	Spray Pumps	2	Amazon
ESP32	Micro-controller	1	
WIFI Donge	Wifi	1	Amazon
HDMI Dummy	Screen Share	1	Conrrad
Connectors (XT90 + 4mm Bullet + Lugs)	Connectors	-	Amazon
Fasteners (M8, M6)	Nut, Bolt, Washers	-	
Gigavac Minitactor P-Series	Solenoid Relay	2	
Gearbox + Accessories	Drive	4	
Brose BLDC Motors	Drive Motor	4	

6. Electrical Schematics:



7. Motor Configuration Using VESCTool

- Open the VESC Tool Application.
  (Download from https://vesc-project.com/vesc_tool for Windows & Linux)

  Some video tutorials :
  https://www.youtube.com/watch?v=QaVPISP8BF4&t=60s
  https://www.youtube.com/watch?v=QM5_vCy_uWk
  https://www.youtube.com/watch?v=vOkU4mDSdu4

- Power the VESC and connect it using the USB B.
  - The VESC will be available as shown:
  - Connect to the vesc which needs to be configured
    (Multiple vesc can be displayed, but only one can be configured at a time)

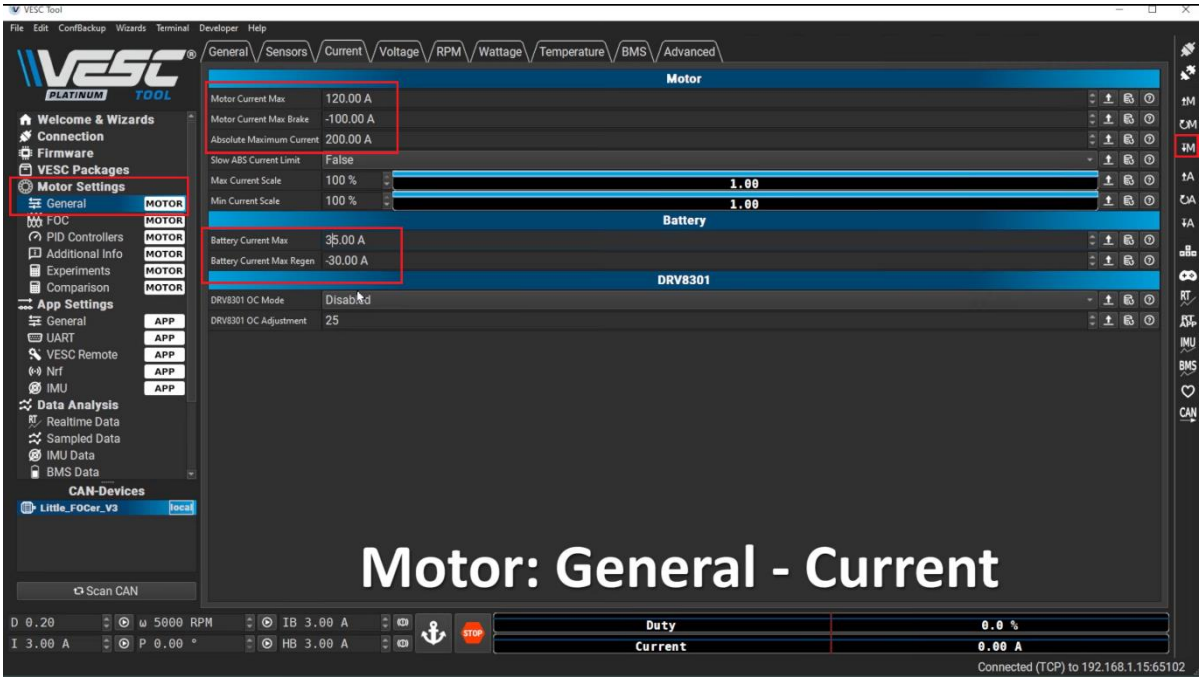
- Click on the Setup Motors FOC, this is a wizard that will guide you through the config process.
- Choose (Large Inrunner) when prompted with motor type selection
  The motor will rotate and check for sensors and give the output at the end if the config. Is completed successfully.

- Advance Settings :

  Under Motor Settings, A lot of parameters can be tuned. All of them are described in detail as well.Some of the Important ones which we config. are :

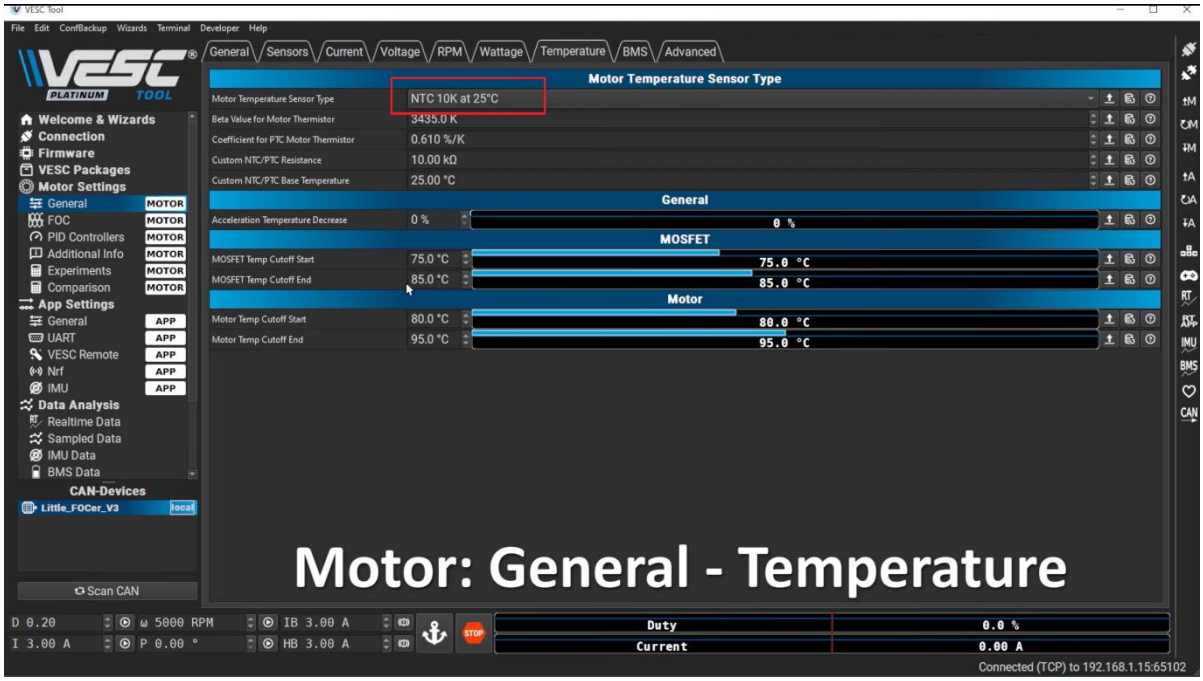
- CURRENT:
  - Set Motor Current Max to 80.00A
  - Set Motor Current Max Brake to -80.00A
  - Set Absolute Maximum Current to 100.00A
  - Set Battery Current Max Regen to not more than -30.00A

  Always Click M with Drop Down Arrow to override your settings on the vesc.

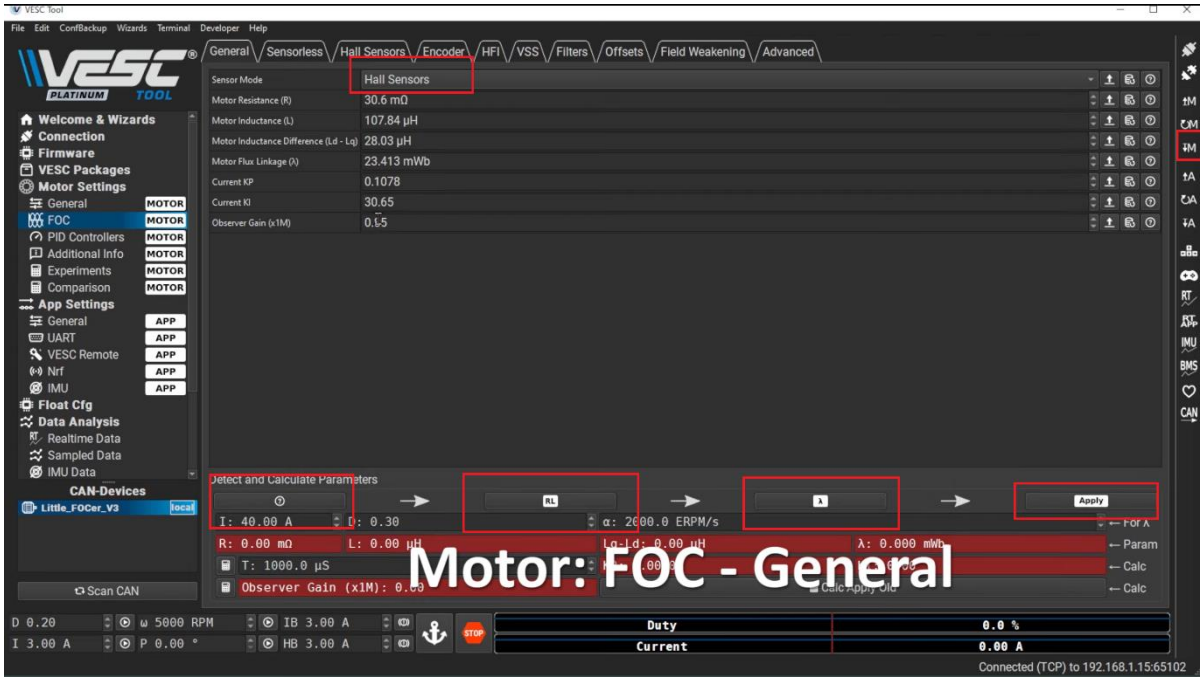


- **TEMPERATURE:**
  - Set Motor Temperature Sensor Type to None (Disable)

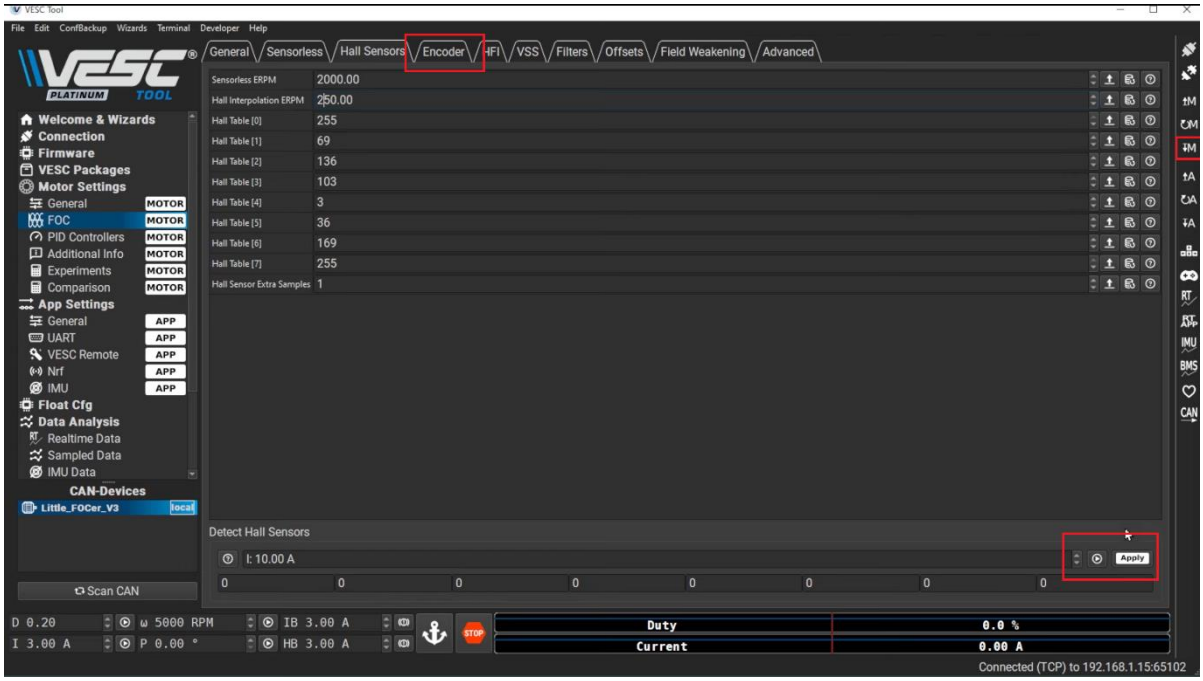
  Since we do not use any temperature sensor, we disable this setting. Otherwise, it will just give a garbage value which can cause uncertain behavior during control.



- **FOC:**
  - In the Sensor Mode select : Encoders
  - Check for Motor Parameters using the : ?,RL and Lambda
  - Apply the updated motor params. and override the settings using M



- **ENCODER:**
  - Encoders can also be detected and tuned from this Tab.
  - Click Play and the motor will start to detect the encoders.

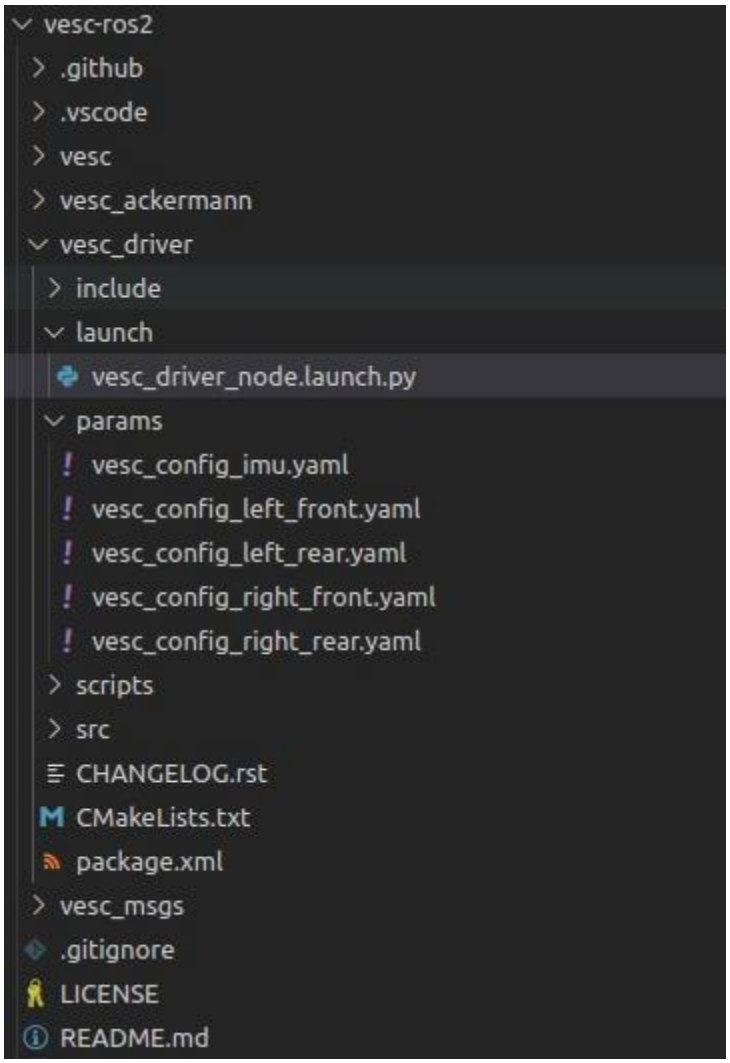


- **Additional Settings :**
  - Make sure the App Settings > General is always set to UART.
  - You can obtain the VESC ID from here.
  - If you have multiple devices over can, they can be found in CAN Devices.
  - Keyboard Control can be activated from the Tab in the right.

8. VESC (Motor Driver) Interface with ROS 2 & Control:

- STEP 1: Installation:

  - Create a workspace (Name: vesc_ws)
  - Git Clone: https://github.com/f1tenth/vesc/tree/ros2
  - Dependencies to be downloaded as well (transport Drivers)
  - Out of this only vesc_driver is Used for Connecting the vesc with the computer using ROS2



  - USE: Access all data from the VESC (imu data, speed, erpm, etc)
  - 4 VESC are used for drive, 1 is used for IMU

- 5 Different Param files are created and configured for vesc, USE: Configure port no, rotor position, IMU YES or No, etc.
- Example for Param file for left front motor-vesc. Similarly, we have 5

vesc-ros2 > vesc_driver > params > ! vesc_config_left_front.yaml

```yaml
/**:
  ros__parameters:
    port: "/dev/ttyACM0"
    rotor_position_source: "encoder"
    brake_max: 200000.0
    brake_min: -20000.0
    current_max: 100.0
    current_min: 0.0
    duty_cycle_max: 1.0
    duty_cycle_min: -1.0
    position_max: 0.0
    position_min: 0.0
    servo_max: 0.85
    servo_min: 0.15
    speed_max: 23250.0
    speed_min: -23250.0
```

- Port No is obtained using the VescTool and should be entered here
- USB Input Sequence as of now with the USB HUB:
  1. Left Front VESC – ttyACM0
  2. Right Front VESC – ttyACM1
  3. Left Rear VESC – ttyACM2
  4. Right Rear VESC – ttyACM3
  5. IMU VESC – ttyACM4

CAUTION: MAINTAIN THIS SEQUENCE ONLY

- Motor Configuration and IMU Configuration are done using VESC TOOL.

CAUTION: READ DOCUMENTATION BEFORE PROCEEDING.

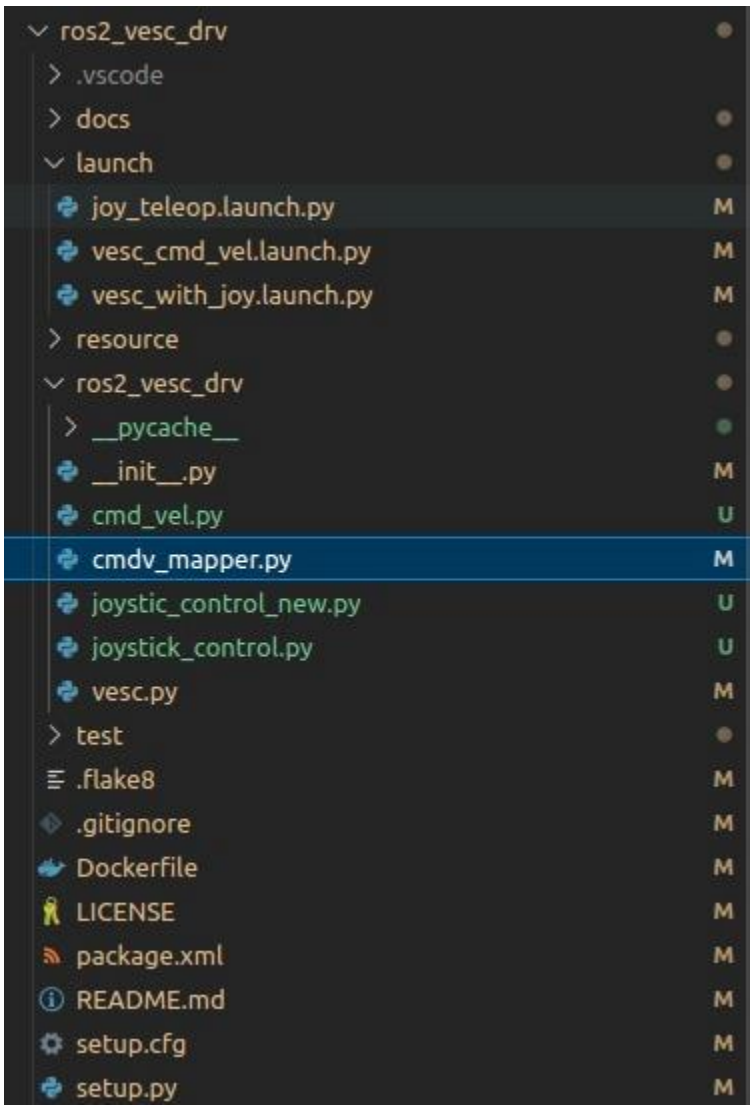
- Make changes in Launch File: vesc_driver_node.launch.py, for launching 5 vesc driver nodes. USE: Access all data from 5 different vesc on different topics.

- Git Clone: https://github.com/uos/sick_tim/tree/humble for connecting the sick tim 2d lidar

- Launch: sick_tim551_2050001_usb.launch.py

- Git Clone: https://github.com/AlexKaravaev/ros2_laser_scan_matcher for scan matching and calculating odometry from laser data

- Dependencies to be downloaded as well (csm)
- To know more: https://youtu.be/HGfJnU2p2YM?si=vZqxLC8_Zwo0cUd6
- Git Clone: https://github.com/vik378/ros2_vesc_drv



- By default it works for 2WD, but for 4WD it needs configuration
- Configure it for publishing 4 different velocities for 4 different vescs
- Make the following changes in cmdv_mapper.py (src):

```
TOPIC_CMD_VEL: Final = "/cmd_vel"
TOPIC_VESC_LEFT_FRONT: Final = "/commands/dutycycle/left_front"
TOPIC_VESC_LEFT_REAR: Final = "/commands/dutycycle/left_rear"
TOPIC_VESC_RIGHT_FRONT: Final = "/commands/dutycycle/right_front"
TOPIC_VESC_RIGHT_REAR: Final = "/commands/dutycycle/right_rear"
```

```python
class BasicRemapper(Node):
    def __init__(self):
        super().__init__("cmd_vel_remapper")
        self.subs_cmdv = self.create_subscription(
            Twist, TOPIC_CMD_VEL, self.remap_cmd_vel, 10
        )
        self.pub_vesc_l_f = self.create_publisher(Float64, TOPIC_VESC_LEFT_FRONT, 10)
        self.pub_vesc_r_f = self.create_publisher(Float64, TOPIC_VESC_RIGHT_FRONT, 10)
        self.pub_vesc_l_r = self.create_publisher(Float64,TOPIC_VESC_LEFT_REAR, 10)
        self.pub_vesc_r_r = self.create_publisher(Float64,TOPIC_VESC_RIGHT_REAR, 10)

    def remap_cmd_vel(self, msg: Twist):
        left, right = diamond_steer(msg.linear.x, msg.angular.z)
        msg = Float64()
        msg.data = left
        self.pub_vesc_l_f.publish(msg)
        self.pub_vesc_l_r.publish(msg)

        msg.data = right
        self.pub_vesc_r_f.publish(msg)
        self.pub_vesc_r_r.publish(msg)
```

- A new file joystick_control_new.py is created for: Start and Stop of odometry and to run the code for task 1 and 2

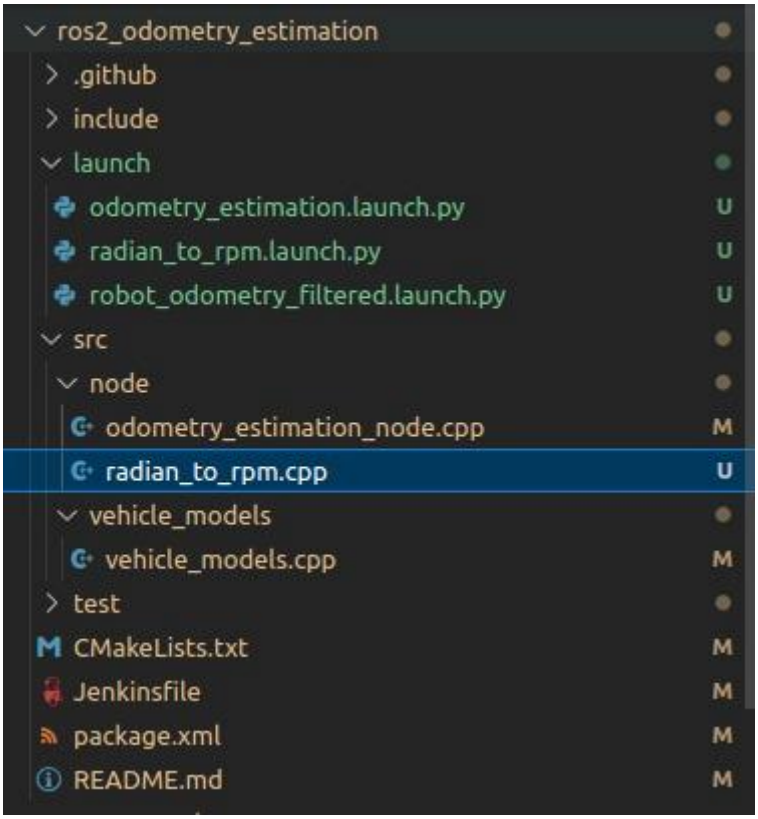
- Add this new file in setup.py:

```python
import os
from glob import glob
from setuptools import setup


package_name = "ros2_vesc_drv"

setup(
    name=package_name,
    version="0.0.0",
    packages=[package_name],
    data_files=[
        ("share/ament_index/resource_index/packages", ["resource/" + package_name]),
        ("share/" + package_name, ["package.xml"]),
        (os.path.join("share", package_name), glob("launch/*.launch.py")),
    ],
    install_requires=["setuptools"],
    zip_safe=True,
    maintainer="Vktork378",
    maintainer_email="viktor@vik.works",
    description="Provides basic VESC driver for differential drive app (2 VESCs)",
    license="Apache License 2.0",
    tests_require=["pytest"],
    entry_points={
        "console_scripts": [
            "vesc_diff_drv=ros2_vesc_drv.vesc:main",
            "cmdv_mapper=ros2_vesc_drv.cmdv_mapper:main",
            "cmd_vel=ros2_vesc_drv.cmd_vel:main",
            "joy_control = ros2_vesc_drv.joystick_control:main",
            "joystic_control_new = ros2_vesc_drv.joystic_control_new:main"

        ],
    },
)
```

- Now make the changes as described below in vesc_with_joy.launch.py
- Then launch: vesc_with_joy.launch.py
- USE: Start Joystick control with all 4 vesc

```python
from launch import LaunchDescription
from launch_ros.actions import Node


def generate_launch_description():
    return LaunchDescription([
        Node(package="joy", executable="joy_node", name="joy_node"),

        #logitech joy config.
        # Node(
        #     package="teleop_twist_joy",
        #     executable="teleop_node",
        #     parameters=[
        #         {"axis_angular": 2},              #right joy button
        #         {"axis_linear": {"x": 3}},
        #         {"require_enable_button": True},   #RB button in joy
        #         {"scale_linear": {"x": 0.4}},
        #         {"scale_angular": {"yaw": 0.4}},
        #     ],
        # ),

        # Ps4 controller config.
        Node(
            package="teleop_twist_joy",
            executable="teleop_node",
            parameters=[
                {"axis_angular": {"yaw":3}},         #right joy button
                {"axis_linear": {"x": 4}},
                {"scale_linear": {"x": 0.44}},
                {"scale_angular": {"yaw": 0.44}},
                {"enable_button": 5},  #R1 Button in joy
            ],
        ),
        Node(
            package="ros2_vesc_drv",
            executable="cmdv_mapper"
        ),
        Node(
            package="ros2_vesc_drv",
            executable="joy_control"
        )
    ])
```

- Git Clone : https://github.com/hiwad-aziz/ros2_odometry_estimation/tree/main/src

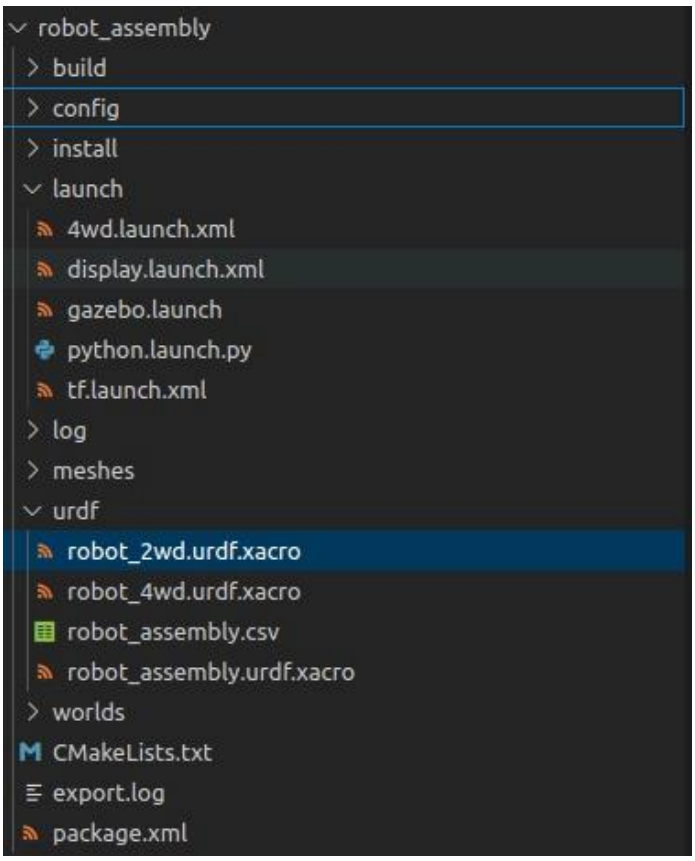


- Changes in odometry_estimation_node.cpp and odometry_estimation_node.h for calculating accurate odom, resolve errors, etc.
- Vehicle_models is not used as it is combined in odometry_estimation_node.cpp

- Create a new file to convert encoder data (Angle-Radians) into RPM of Motor
- Refer this code from file: radian_to_rpm.cpp from Computer/Drive.
- Now make this new launch file as shown below : radian_to_rpm.launch.py , to launch radian_to_rpm.cpp node for getting rpm from the front 2 motors (vesc)

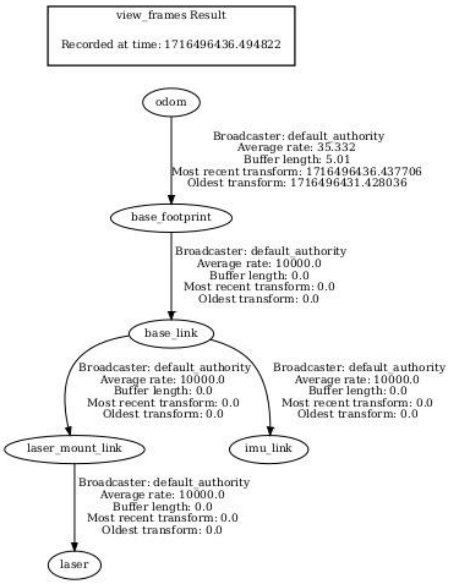
```python
import os

from ament_index_python.packages import get_package_share_directory
from launch import LaunchDescription
from launch.actions import DeclareLaunchArgument
from launch.substitutions import LaunchConfiguration
from launch_ros.actions import Node

def generate_launch_description():

    return LaunchDescription([

        Node(
            package='odometry_estimator',
            executable='radian_to_rpm',
            name='radian_to_rpm_node_left',
            remappings=[("motor_rpm","motor_rpm_left_front"),
                        ('sensors/rotor_position','sensors/rotor_position_left_front')]
        ),
        Node(
            package='odometry_estimator',
            executable='radian_to_rpm',
            name='radian_to_rpm_node_right',
            remappings=[("motor_rpm","motor_rpm_right_front"),
                        ('sensors/rotor_position','sensors/rotor_position_right_front')]
        ),

    ])
```

- File: odometry_estimation.launch.py launches odometry estimation node to calculate odometry from motor RPM
- 5th VESC is used for IMU
- A new package is created with the name: robot_assembly for tf2_tree.
- USE: To get transforms among the base to all the sensors and others.



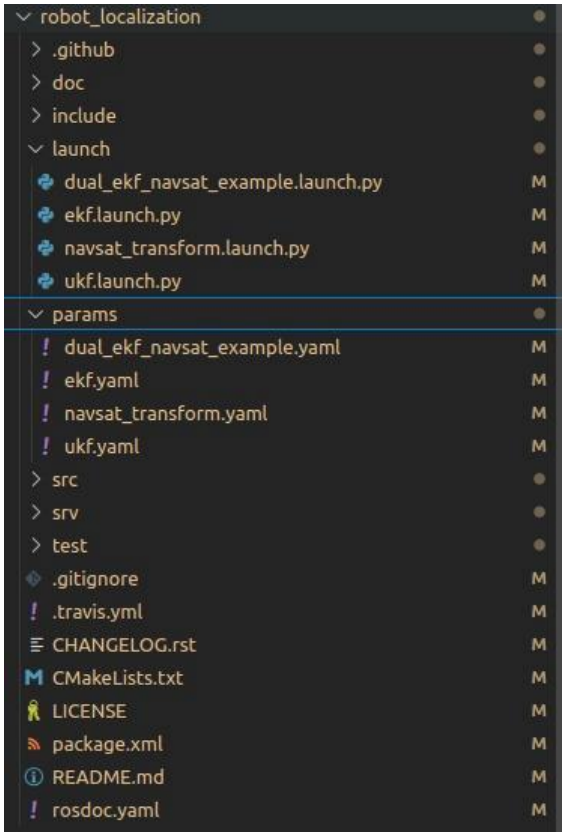
- TF is visualized as follows:

```
ros2 run tf2_tools view_frames
```



- A URDF file is created with name: robot_2wd.urdf.xacro and to visualize this tf tree we have created a launch file with name: 4wd.launch.xml

- Git Clone: https://github.com/cra-ros-pkg/robot_localization/tree/humble-devel
- Set param in ekf.yaml to fuse sensor readings from Encoder, IMU and Laser scan to get a refined odometry estimation.
- For this topic, names must be changed, for different config and to change the base link frame.
- In ekf.yaml, additional code is added to obtain odometry from laser scan data.
- Tuning of parameters is done in sensor config Matrix and Process Noise Covariance for better results. Refer to the file.
- Some necessary changes are made in the ekf.launch.py file to execute some static transform publishers and to remap some topics.



Final Launch Sequence:

- Step 1:

```
ros2 launch linorobot2gazebo gazebo.launch.py
ros2 launch vesc_driver vesc_driver_node.launch.py
```

  - Launch sick tim node

```
ros2 launch sick_tim sick_tim551_2050001_usb.launch.py
```

  - Launch 5 different vesc driver nodes (connecting 5 vesc)

- Step 2:

```
ros2 launch ros2_vesc_drv vesc_with_joy.launch.py
```

  - Enable Joystick (PS4 Blue)
  - Button O: Launch robot_odometry_filtered.launch.py from
    - odometry_estimation package. It will launch: Robot wheel
    - odomety, Tf, Laser Based odometry, EKF (from single file)
  - Button X: To Stop actions of button O
  - Button Triangle: Launch node.launch.py from local_controller package
  - Launch Lane1, Sequence1 and Sequence2 for Task 1 & 2
  - Button Square:To Stop actions of triangle

ROS 2 Navigation 2 (Nav2) is an advanced and robust navigation framework designed for mobile robots using the Robot Operating System (ROS) 2. It provides essential tools and capabilities for autonomous navigation, enabling robots to move from one location to another while avoiding obstacles and optimizing their paths.
Here's a brief overview of its key components and functionalities:

1. **Key Functionalities:**
   - Global Planner:
     - Function: Computes the optimal path from the robot's current position to the target destination using a global map.
     - Algorithms: Supports various path-planning algorithms, such as A* and Dijkstra's algorithm.
   - Local Planner:
     - Function: Generates a feasible path for the robot to follow in real-time, considering the robot's kinematic constraints and dynamic obstacles.
     - Algorithms: Includes methods like Dynamic Window Approach (DWA) and Timed Elastic Band (TEB).

   - Behavior Tree (BT) Executor:
     - Function: Manages and coordinates the robot's navigation tasks through a modular and flexible behavior tree structure.
     - Benefits: Improves reliability and maintainability of complex navigation tasks by breaking them down into simpler, manageable actions.

   - Costmap 2D:
     - Function: Maintains a 2D occupancy grid that represents the environment, combining sensor data to identify obstacles and free space.
     - Layers: Can be configured with multiple layers to integrate static maps, sensor data, and dynamic obstacles.

   - Recovery Behaviors:
     - Function: Provides mechanisms to recover from navigation failures, such as getting stuck or losing the global path.
     - Examples: Includes behaviors like clearing the local costmap, rotating in place, and re-planning the path.

   - Sensor Integration:
     - Types: Supports various sensors, including LiDAR, cameras, and IMUs, for environment perception and localization.
     - Fusion: Uses sensor fusion techniques to improve accuracy and robustness of obstacle detection and localization.

2. **Core Functionalities :**

- Path Planning: Determines the optimal path for the robot to reach its goal using both global and local planners.
- Obstacle Avoidance: Continuously monitors the environment and adjusts the robot's path to avoid collisions.
- Localization: Utilizes algorithms like AMCL (Adaptive Monte Carlo Localization) to estimate the robot's position within the map.
- Mapping: Supports simultaneous localization and mapping (SLAM) techniques to create and update maps of the environment.

3. **Applications:**

- Autonomous Mobile Robots (AMRs): Commonly used in warehouses, factories, and hospitals for tasks like material transport and delivery.
- Service Robots: Deployed in homes, offices, and public spaces for activities like cleaning, security, and customer service.
- Research and Development: Provides a flexible and extensible platform for developing and testing new navigation algorithms and robotic applications.

4. **Advantages:**
   - Modularity: Highly modular design allows for easy customization and integration with different robot platforms and sensors.
   - Scalability: Can be scaled to accommodate robots of various sizes and capabilities, from small indoor robots to large outdoor vehicles.
   - Community Support: Strong community and extensive documentation facilitate development, troubleshooting, and innovation.

Nav2 is a critical component of the ROS 2 ecosystem, providing robust and flexible navigation capabilities
essential for modern autonomous robots. Its advanced features and modular architecture make
it a preferred choice for developers and researchers working on mobile robotics applications.

Installation of Nav2 Package :

To install the Nav2 package in ROS 2, follow these steps. This guide assumes you have already installed ROS 2 on your system. The instructions provided are for ROS 2 humble Hawksbill, but similar steps can be adapted for other ROS 2 distributions.

1. Prerequisites
   - Ensure you have installed ROS 2 and sourced your ROS 2 installation. You can verify this by running:

```
source /opt/ros/humble/setup.bash
```

2. Install Dependencies
   - Install the required dependencies for Nav2:

```
sudo apt update
sudo apt install -y \
ros-humble-navigation2 \
ros-humble-nav2-bringup
```

3. Create a ROS 2 Workspace
   - If you don't already have a ROS 2 workspace, you can create one:

```
mkdir -p /ros2_ws/src
cd / ros2_ws
colcon build
source install/setup.bash
```

4. Clone the Nav2 Repository
   - Navigate to your workspace's src directory and clone the Nav2 repository:

```
cd / ros2ws / src
git clone https://github.com/ros-planning/snavigation2.git
```

5. Install Additional Dependencies
   - Use rosdep to install any additional dependencies required by Nav2:

```
cd / ros2_ws
rosdep install -y --from-paths src --ignore-src --ros distro humble
```

6. Build the Workspace
   - Build your workspace using colcon:

```
colcon build
```

7. Source the Workspace
   - After building, source the workspace:

```
source install/setup.bash
```

8. Launch Nav2
   - You can launch the Nav2 stack using one of the examples bringup launch files. For example:

```
ros2 launch linorobot2navigation navigation.launch.py
```

9. Running in Simulation
   - If you are using a Linorobot2 in Gazebo, you can use the following commands to start the simulation:

```
ros2 launch linorobot2gazebo gazebo.launch.py
```

10. Troubleshooting
    - Ensure all dependencies are correctly installed.
    - Verify that you have sourced your ROS 2 installation and your workspace setup file.
    - Check for any build errors and address missing dependencies or configuration issues.
    - Following these steps will set up and install the Nav2 package in your ROS 2 environment, allowing you to start developing and testing navigation functionalities for your robot.

(SLAM) Navigating While Mapping

1. Overview

This document explains how to use Nav2 with SLAM. The following steps show ROS 2 users how to generate occupancy grid maps and use Nav2 to move their robot around. This tutorial applies to both simulated and physical robots, but will be completed here on a physical robot. In this tutorial, we will be using the SLAM Toolbox.

2. Requirements
   - You must install Navigation2, Linorobot2, and SLAM Toolbox. If you don't have them installed, SLAM Toolbox can be installed via:

```
sudo apt install ros-<ros2-distro>-slam-toolbox
```

   or from built from source in your workspace with:

```
git clone -b <ros2-distro>-devel git@github.com: stevemacenski / slam toolbox.git
```

3. Tutorial Steps

- Launch Robot Interfaces
  For this tutorial, we will use the Linorobot2. If you have another robot, replace with your robot specific interfaces. Typically, this includes the robot state publisher of the URDF, simulated or physical robot interfaces, controllers, safety nodes, and the like. Run the following commands first whenever you open a new terminal during this tutorial.

```
source/opt/ros/<ros2-distro>/setup.bash
```

- Launch your robot's interface and robot state publisher,

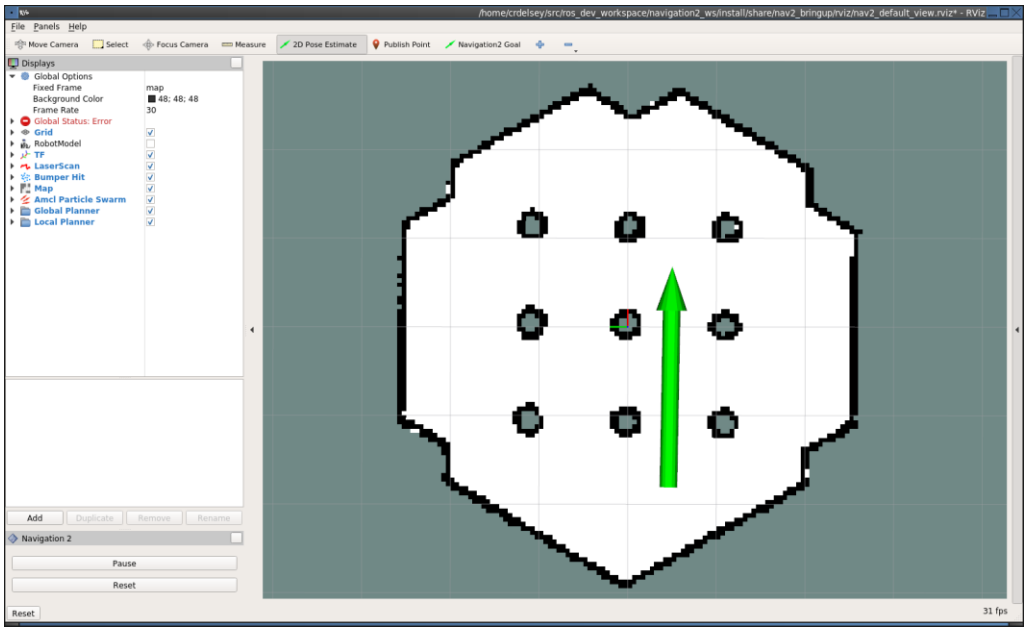
```
ros2 launch vesc_driver vesc_driver_node.launch.py
ros2 launch odometry_estimator robot_odometry_filtered.launch.py
```

  - Launch Navigation2
    Launch Navigation without nav2 amcl and nav2 map server. It is assumed that the SLAM node(s) will publish to /map topic and provide the map->odom transform.

```
ros2 launch nav2_bringup navigation.launch.py
```

  - Launch SLAM
    Bring up your choice of SLAM implementation. Make sure it provides the map->odom transform and/map topic. Run Rviz and add the topics you want to visualize such as /map, /tf, /laserscan, etc. For this tutorial, we will use SLAM Toolbox.

```
ros2 launch slam_toolbox online_async.launch.py
```



Working with SLAM

Move your robot by requesting a goal through RViz.

Getting Started Simplification

If you're only interested in running SLAM in the Linorobot2 getting started sandbox world, we also provide a simple way to enable SLAM as a launch configuration. Rather than individually launching the interfaces, navigation, and SLAM, you can continue to use the tb3 simulation launch.py with slam config set to true. We provide the instructions above with the assumption that you'd like to run SLAM on your own robot, which would have separated simulation / robot interfaces and navigation launch files that are combined in tb3 simulation launch.py for the purposes of easy testing.

Python Code for ROS 2 Navigator with Odometry

Code Folder Name:

```python
#!/usr/bin/env python3
import rclpy
from rclpy.node import Node
from nav2_simple_commander.robot_navigator import BasicNavigator
from geometry_msgs.msg import PoseStamped, Quaternion, TransformStamped
from nav_msgs.msg import Odometry
import tf_transformations

class NavigatorWithOdom(Node):
    def __init__(self):
        super().__init__('navigator_with_odom')
        self.nav = BasicNavigator()
        self.odom_subscriber = self.create_subscription(Odometry, 'odom', self.
            odom_callback, 10)
        self.current_pose = PoseStamped()
        self.initial_pose = PoseStamped()


    def odom_callback(self, msg):
        self.initial_pose.pose.position.x = 0.0
        self.initial_pose.pose.position.y = 0.0
        self.initial_pose.pose.position.z = 0.0
        self.initial_pose.pose.orientation.x = 0.0
        self.initial_pose.pose.orientation.y = 0.0
        self.initial_pose.pose.orientation.z = 0.0
        self.initial_pose.pose.orientation.w = 1.0
        self.current_pose.header = msg.header
        self.initial_pose.header = msg.header
        self.current_pose.pose = msg.pose.pose

    def create_pose_stamped(self, base_pose, position_x, position_y, rotation_z):
        q_x, q_y, q_z, q_w = tf_transformations.quaternion_from_euler(0.0, 0.0,
            rotation_z)
        goal_pose = PoseStamped()
        goal_pose.header.frame_id = 'map'
        goal_pose.header.stamp = self.get_clock().now().to_msg()
        goal_pose.pose.position.x = base_pose.pose.position.x + position_x
        goal_pose.pose.position.y = base_pose.pose.position.y + position_y
        goal_pose.pose.position.z = 0.0

        base_orientation_q = base_pose.pose.orientation
        base_orientation = [base_orientation_q.x, base_orientation_q.y,
            base_orientation_q.z, base_orientation_q.w]
        base_yaw = tf_transformations.euler_from_quaternion(base_orientation)[2]
        new_yaw = base_yaw + rotation_z

        new_orientation_q = tf_transformations.quaternion_from_euler(0.0, 0.0, new_yaw)
```

ROS 2 Navigation Configuration Parameters

Code Folder Name:

```yaml
amcl:
  ros__parameters:
    use_sim_time: True
    alpha1: 0.2
    alpha2: 0.2
    alpha3: 0.2
    alpha4: 0.2
    alpha5: 0.2
    base_frame_id: "base_footprint"
    beam_skip_distance: 0.5
    beam_skip_error_threshold: 0.9
    beam_skip_threshold: 0.3
    do_beamskip: false
    global_frame_id: "map"
    lambda_short: 0.1
    laser_likelihood_max_dist: 2.0
    laser_max_range: 100.0
    laser_min_range: -1.0
    laser_model_type: "likelihood_field"
    max_beams: 60
    max_particles: 2000
    min_particles: 500
    odom_frame_id: "odom"
    pf_err: 0.05
    pf_z: 0.99
    recovery_alpha_fast: 0.0
    recovery_alpha_slow: 0.0
    resample_interval: 1
    robot_model_type: "nav2_amcl::DifferentialMotionModel"
    save_pose_rate: 0.5
    sigma_hit: 0.2
    tf_broadcast: true
    transform_tolerance: 1.0
    update_min_a: 0.2
    update_min_d: 0.25
    z_hit: 0.5
    z_max: 0.05
    z_rand: 0.5
    z_short: 0.05
    scan_topic: scan

bt_navigator:
  ros__parameters:
    use_sim_time: True
    global_frame: map
    robot_base_frame: base_link
    odom_topic: /odometry/filtered
    bt_loop_duration: 10
    default_server_timeout: 20
    plugin_lib_names:
      - nav2_compute_path_to_pose_action_bt_node
      - nav2_compute_path_through_poses_action_bt_node
      - nav2_smooth_path_action_bt_node
      - nav2_follow_path_action_bt_node
      - nav2_spin_action_bt_node
      - nav2_wait_action_bt_node
      - nav2_assisted_teleop_action_bt_node
```

ROS 2 Navigation 2 (Nav2) Configuration Overview

Configuration Overview

1. odom topic: /odometry/filtered
   • Explanation: This specifies the topic name from which the robot will receive odometry data. In this case, it's /odometry/filtered, which suggests that the odometry data has been processed or filtered to provide more accurate information.

2. xy goal tolerance and yaw goal tolerance
   • xy goal tolerance: 0.4: Defines the tolerance (in meters) for how close the robot needs to get to the goal position in the XY plane before it considers the goal reached.
   • yaw goal tolerance: 0.4: Defines the tolerance (in radians) for how close the robot needs to be to the goal orientation (yaw angle) before it considers the goal reached.

3. inflation radius: 0.55
   • Explanation: This parameter defines the radius (in meters) around obstacles in the costmap that is inflated to ensure the robot avoids collisions. A larger inflation radius provides a wider safety margin around obstacles.

4. FollowPath Plugin Configuration
   • plugin:" dwb core::DWBLocalPlanner": Specifies the local planner plugin to use for following the path. DWBLocalPlanner is a local planner that uses the Dynamic Window Approach (DWA) for trajectory generation and control.
   • debug trajectory details: True: Enables detailed debugging information about trajectories, useful for troubleshooting and analysis.
   • min vel x: 0.0 and max vel x: 0.15: Defines the minimum and maximum linear velocities (in meters per second) for the robot along the X-axis.
   • min vel y: 0.0 and max vel y: 0.0: Defines the minimum and maximum linear velocities along the Y-axis, both set to 0, indicating no movement in the Y direction.
   • max vel theta: 0.0: Specifies the maximum angular velocity (in radians per second). Here, it is set to 0, indicating no rotational movement.
   • min speed xy: 0.0 and max speed xy: 0.09: Defines the minimum and maximum speed in the XY plane.
   • min speed theta: 0.0: Defines the minimum angular speed, which is set to 0.
   • acc lim x: 0.03: Defines the acceleration limit (in meters per second squared) for linear motion along the X-axis.
   • acc lim y: 0.0: Defines the acceleration limit for linear motion along the Y-axis, set to 0.

5. Critics
   • Explanation: This lists the types of critics used in the local planner to evaluate the quality of the generated trajectories. Critics are used to penalize certain undesirable behaviors.

      - RotateToGoal: Penalizes deviation from the goal orientation.
      - Oscillation: Penalizes unnecessary oscillations in the robot's motion.
      - BaseObstacle: Penalizes trajectories that come too close to obstacles.
      - GoalAlign: Penalizes poor alignment with the goal.
      - PathAlign: Penalizes poor alignment with the path.
      - PathDist: Penalizes being too far from the path.
      - GoalDist: Penalizes being too far from the goal.

6. **Critical Parameters**
   • BaseObstacle.scale: 0.02: Scale factor for the BaseObstacle critic, affecting how strongly it penalizes trajectories near obstacles.
   • PathAlign.scale: 12.0: Scale factor for the PathAlign critic, affecting the penalty for misalignment with the path.
   • PathAlign.forward point distance: 0.3: Distance (in meters) from the robot's front where path alignment is evaluated.
   • GoalAlign.scale: 12.0: Scale factor for the GoalAlign critic, affecting the penalty for misalignment with the goal.
   • GoalAlign.forward point distance: 0.3: Distance (in meters) from the robot's front where goal alignment is evaluated.
   • PathDist.scale: 2.0: Scale factor for the PathDist critic, affecting the penalty for being too far from the path.
   • GoalDist.scale: 14.0: Scale factor for the GoalDist critic, affecting the penalty for being too far from the goal.
   • RotateToGoal.scale: 12.0: Scale factor for the RotateToGoal critic, affecting the penalty for not aligning properly with the goal direction.
   • RotateToGoal.slowing factor: 10.0: Factor affecting how quickly the robot slows down as it approaches the goal direction.
   • RotateToGoal.lookahead time: 1.0: Time (in seconds) for looking ahead to predict the robot's alignment with the goal.

9. Task 1: Navigation in a Maize Field

Objective:

Navigate a robot autonomously through a maize field, starting at a designated location (S) and following adjacent rows from T1 to T10. The distance between rows is 0.75 meters, and the maize plants are approximately 0.3 - 0.4 meters tall. Note that some plants may be missing but not at the beginning or end of any row.

Methodology:

Algorithm Selection: To tackle this task, we employed a combination of the RANSAC algorithm and a Lane Keep Algorithm to traverse the distance and two Sequence nodes for turning of robot clockwise and anticlockwise respectively according to requirement.

1. **RANSAC Algorithm:**

- **Purpose:** Identify the positions of crop rows.
- **Process:**
    - Filter the LiDAR data to focus on the frontal 180 degrees.
    - Use RANSAC to detect points that likely represent maize plants.
    - Plot these points using MATHPLOT to visualize and confirm the area of interest using the bounding box.

2. **Lane Keep Algorithm:**

- **Purpose:** Maintain the robot's position within the crop rows.
- **Process:**
    - Implement a simplified lane-keeping mechanism.
    - When the robot approaches a crop row (left or right), it adjusts its course.
    - If the robot gets too close to a row, the system introduces an angular velocity in the opposite direction to steer away from the plants.
    - The robot maintains a set linear velocity when centered between rows and adjusts its angular velocity to correct its path as needed.

3. **Turning Sequence:**

- **Objective:** Navigate the robot to the next row after reaching the end of the current row.
- **Process:**
    - At the end of a row, based on the odometry data (/odometry/filtered topic), the robot initiates a turning sequence.

- **Turning Sequence Steps:**

  - Move straight to exit the current row.
  - Execute a left turn.
  - Move straight for a certain distance to align with the next row.
  - Execute another left turn.
  - Move straight again to enter the second row.

```
def control_loop(self):
    if self.active:
        # self.get_logger().info('Sequence started')
        if self.initial_position is None or self.initial_orientation is None:
            return

        # if self.state == 'set_desired_orientation':
        #     self.set_desired_orientation(3, self.desired_orientation)
        if self.state == 'move_forward':
            self.move_forward()
        elif self.state == 'turn':
            self.turn()
        elif self.state == 'move_forward_again':
            self.move_forward_again()
        elif self.state == 'turn_again':
            self.turn_again()
        elif self.state == 'move_forward_final':
            self.move_forward_final()
        elif self.state == 'done':
            self.active = False
            self.state = 'move_forward'
            self.get_logger().info(f'self.active={self.active}')
            self.get_logger().info(f'self.state={self.state}')
```

- **Direction Decision:**

  - When the robot reaches the end of a row, it checks its y-coordinate and value of the count variable. Count variable indicates row number in maize field. Initially, a count variable is set to one because of first row.
  - If the y-coordinate is greater than equal to right_x_lower_limit and count variable is odd, then the robot makes a left turn.
  - If the y-coordinate is less than and equal to left_x_lower_limit and count variable is even, then the robot makes a right turn.

- **<u>Integration:</u>**

  - The robot starts at location S and follows the designated tracks (T1 for example).
  - The combined use of RANSAC and Lane Keep algorithms ensures the robot can accurately detect and navigate between the rows, even if some plants are missing.
  - The system continuously monitors the distance to the rows and makes real-time adjustments to keep the robot centered and avoid collisions.

- Once end of row is detected by checking odometry and count variable, it will either run sequence 1 or sequence 2 program accordingly.

- **Parameter Description:**
  - right_x_lower_limit = defines the maximum length to be traversed in forward direction and trigger sequence one turning.
  - left_x_lower_limit = defines the limit after which lane1 code start working. And also trigger limit for sequence two turning.
  - bp = limits for rectangular bounding box for lidar point visualization.

```
right_x_lower_limit = 13.5
left_x_higher_limit = 0.5
laser_ranges = []
laser_angles = []
Count = 1

bp = [[0.3, 0.100], [0.75, 0.31], [0.3, -0.31], [0.75, -0.100]]  # , [0.15, 0.1], [0.5, 0.2], [0.15, -0.2], [0.5, -0.1]]
width1 = bp[1][0] - bp[0][0]
height1 = bp[1][1] - bp[0][1]
width2 = bp[3][0] - bp[2][0]
height2 = bp[3][1] - bp[2][1]
```

  - twist_msg.linear.x =defines maximum linear velocity with which robot start traversing the maize field. Values of this parameter need to fine tuned in forward velocity function and avoid obstacle function to get correct results according to actual site conditions.
  - twist_msg.angular.z =defines maximum angular velocity with which robot start traversing the maize field. Values of this parameter need to fine tuned in forward velocity function and avoid obstacle function to get correct results according to actual site conditions.

Repository:

Link:

Lane1:

- clbk_laser: Segmentation of lidar data for visualization and detecting crops in particular regions.

```
def clbk_laser(self, msg):
    global regions, right_dist, right_dist_avg, left_dist, left_dist_avg, front_dist, front_dist_avg, Count
    ranges = msg.ranges
    ranges = [x for x in ranges if not math.isnan(x)]
    custom_range = ranges[270:360] + ranges[0:90]

    right_dist = [x for x in custom_range[0:70] if x < 1.1]
    left_dist = [x for x in custom_range[110:180] if x < 1.1]

    if len(right_dist) == 0 and len(left_dist) == 0:
        print('no data points for distance found')
        right_dist_avg = 0
        left_dist_avg = 0
    elif len(right_dist) == 0 and len(left_dist) > 0:
        left_dist_avg = sum(left_dist) / len(left_dist)
        right_dist_avg = 0

    elif len(right_dist) > 0 and len(left_dist) == 0:
        right_dist_avg = sum(right_dist) / len(right_dist)
        left_dist_avg = 0

    elif len(right_dist) > 0 and len(left_dist) > 0:
        right_dist_avg = sum(right_dist) / len(right_dist)
        left_dist_avg = sum(left_dist) / len(left_dist)

    regions = {
        'right': min(min(custom_range[0:70]), 1.0),
        'right-1': min(min(custom_range[30:70]), 0.75),
        'fright': min(min(custom_range[70:85]), 1),
        'front': min(min(custom_range[70:110]), 1),
        'forfront': min(min(custom_range[85:95]), 1),
        'fleft': min(min(custom_range[95:110]), 1),
        'left': min(min(custom_range[110:180]), 1.0),
        'left-1': min(min(custom_range[110:150]), 0.75),
    }
```

- clbk_odom: Process the odometry data and check the robot position, which is key parameter in deciding running of sequence 1 or sequence 2 program.

```
def clbk_odom(self, msg):
    global position_
    global yaw_, active_, n, right_x_lower_limit, left_x_higher_limit, Count, regions

    # position
    position_ = msg.pose.pose.position
    if n == 0:
        if position_.x < right_x_lower_limit:
            active_ = True
        elif position_.x >= right_x_lower_limit:
            active_ = False
            n = None
        # elif (max(regions['left'])==1.0 and max(regions['right']==1.0)):
        #     active_ = False
        #     n = None


  if n is None:
      if left_x_higher_limit < position_.x < right_x_lower_limit:
          active_ = True
      elif position_.x <= left_x_higher_limit or position_.x >= right_x_lower_limit:
          active_ = False
      # elif (max(regions['left'])==1.0 and max(regions['right']==1.0)):
      #     active_ = False

  if position_.x >= (right_x_lower_limit - 0.1) and (Count % 2) != 0:
      self.state_pub.publish(Int32(data=1))
      # print("sequence 1")
      msg = String()
      msg.data='stop+1'
      self.stop_pub.publish(msg)
      Count+=1
      self.get_logger().info(f'Count: {Count}')
```

```
if position_.x <= (left_x_higher_limit + 0.1) and (Count % 2) == 0:
    self.state_pub.publish(Int32(data=2))
    # print("sequence 2")
    self.get_logger().info(f'Count: {Count}')

# if (max(regions['left'])==1.0 and max(regions['right']==1.0)):
#     if (Count % 2) == 0:
#         self.state_pub.publish(Int32(data=2))
#         # print("sequence 2")
#         Count+=1
#         self.get_logger().info(f'Count: {Count}')

# yaw
quaternion = (
    msg.pose.pose.orientation.x,
    msg.pose.pose.orientation.y,
    msg.pose.pose.orientation.z,
    msg.pose.pose.orientation.w)
euler = euler_from_quaternion(quaternion)
yaw_ = euler[2]
```

- forward_velocity: If the robot position is between left_x_lower_limit and right_x_lower_limit, then controller will publish a constant linear velocity. If any crop detected in left or right bounding box then it will go into change_state function.

```
def forward_velocity(self):
    global pub, position_, X_left, X_right
    twist_msg = Twist()
    if position_.x>.5 and position_.x<5.5:
        twist_msg.linear.x = 0.1 #0.1
    else:
        twist_msg.linear.x = 0.07
        print('forward else')

    pub.publish(twist_msg)

    if X_left.shape[0] > 0:
        self.change_state(1)

    if X_right.shape[0] > 0:
        self.change_state(1)
```

- change_state: This function will switch the operation of robot from forward_velocity to avoid_obstacle or vice-versa.

- avoid_obstacle: If any crop is detected in any bounding box(right/left) then controller will publish velocities which will move robot away from the detected plants. If there is nothing in any bounding box then it will again go into forward_velocity function.

```
def avoid_obstacle(self):
    global pub
    global yaw_, pub, yaw_precision_, state_, regions, X_left, X_right, X_left_1, X_right_1
    twist_msg = Twist()
    # rate = self.create_rate(20)

    if X_left.shape[0] > 0:
        #while X_left.shape[0] > 0:
        twist_msg.linear.x = 0.055 #0.1
        twist_msg.angular.z = 0.06
        # self.get_logger().info("points found on Left" )

        pub.publish(twist_msg)
        # self.get_logger().info('spinning in left')
        # rate.sleep()
        time.sleep(0.1)
        self.change_state(0)
```

```
    elif X_right.shape[0] > 0:
        #while X_right.shape[0] > 0:
        twist_msg.linear.x = 0.055 #0.1
        twist_msg.angular.z = -0.06
        # self.get_logger().info('points found on Right')
        pub.publish(twist_msg)
        # self.get_logger().info('spining in right')
        # rate.sleep()
        time.sleep(0.1)
        self.change_state(0)
```

Running Sequence:

1. **Simulation:**

- Open the Gazebo Simulator. (Load the maize environment).

```
ros2 launch linorobot2_gazebo gazebo.launch.py
```

  - *Pre-requirements:*
  - Check the lidar data points in Rviz.
  - Check the odometry topic (it should be /odom)

- Run the following program files:

```
ros2 run local_controller lane1
ros2 run local_controller sequence1
ros2 run local_controller sequence2
```

2. **Running Sequence in a real environment:**
   - *Checkpoints:*
   - Lidar data to be visualized in rviz
   - Check the odometry topic. (it should be odometry/filtered)
   - Check the odometry data and it should be reflected as (0,0)

- Start the odometry estimator program.

- Step 1:

```
ros2 launch linorobot2gazebo gazebo.launch.py
ros2 launch vesc_driver vesc_driver_node.launch.py
```

  - Launch sick tim node

```
ros2 launch sick_tim sick_tim551_2050001_usb.launch.py
```

  - Launch 5 different vesc driver nodes (connecting 5 vesc)

- Step 2:

```
ros2 launch ros2_vesc_drv vesc_with_joy.launch.py
```

- Enable Joystick (PS4 Blue)
- Button O: Launch robot_odometry_filtered.launch.py from odometry_estimation package. It will launch: Robot wheel odomety, Tf, Laser Based odometry, EKF (from single file)
- Button X: To Stop actions of button O
- Button Triangle: Launch node.launch.py from local_controller package
- Launch Lane1, Sequence1 and Sequence2 for Task 1 & 2
- Button Square:To Stop actions of triangle

- Run the main Python file. (lane1).

```
ros2 run local_controller lane1
ros2 run local_controller sequence1
ros2 run local_controller sequence2
```

10. Task 2: Counting of maize crops

Objective:

Task 2 is essentially an extension of Task 1, with additional requirements for the field robots. Besides navigating through the field as specified in Task 1, the robots must now also count the number of plants in each row and provide this data row by row using a REST API. The commands provided in the next section are to be used with the navigation stack. The task is to count maize crops so the explanation of the yolo model training and further implementation of the counting operation is provided.

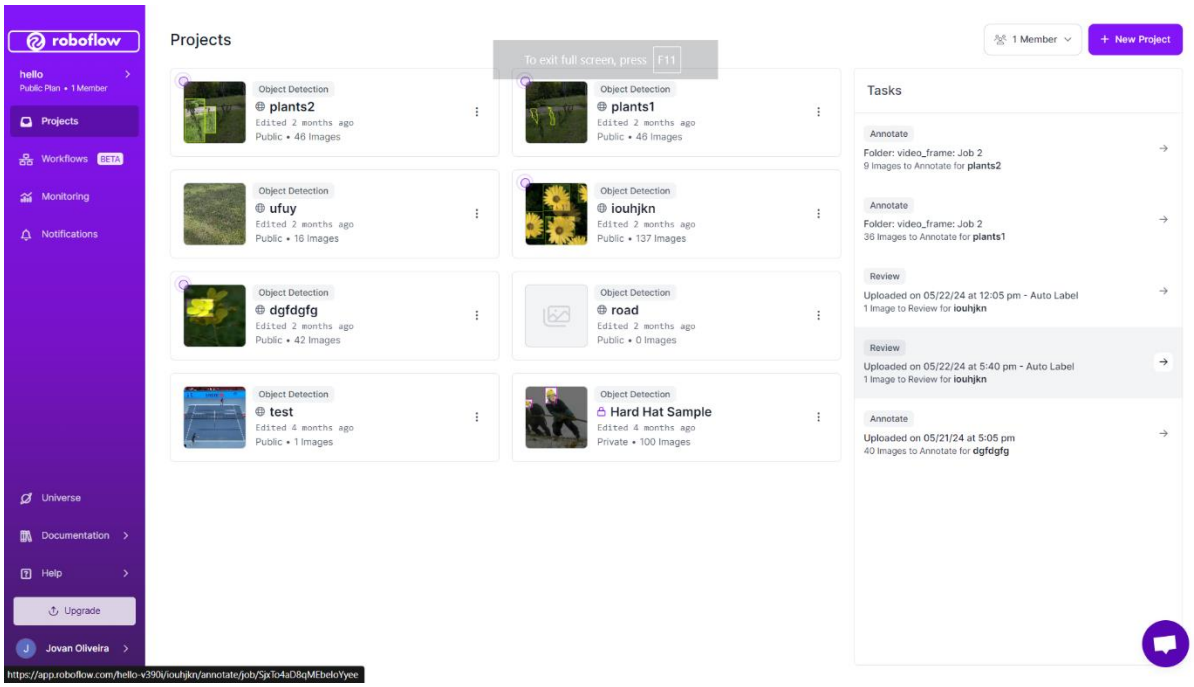
Additionally, at the end of the task, a JSON logfile must be generated and provided via a USB stick.

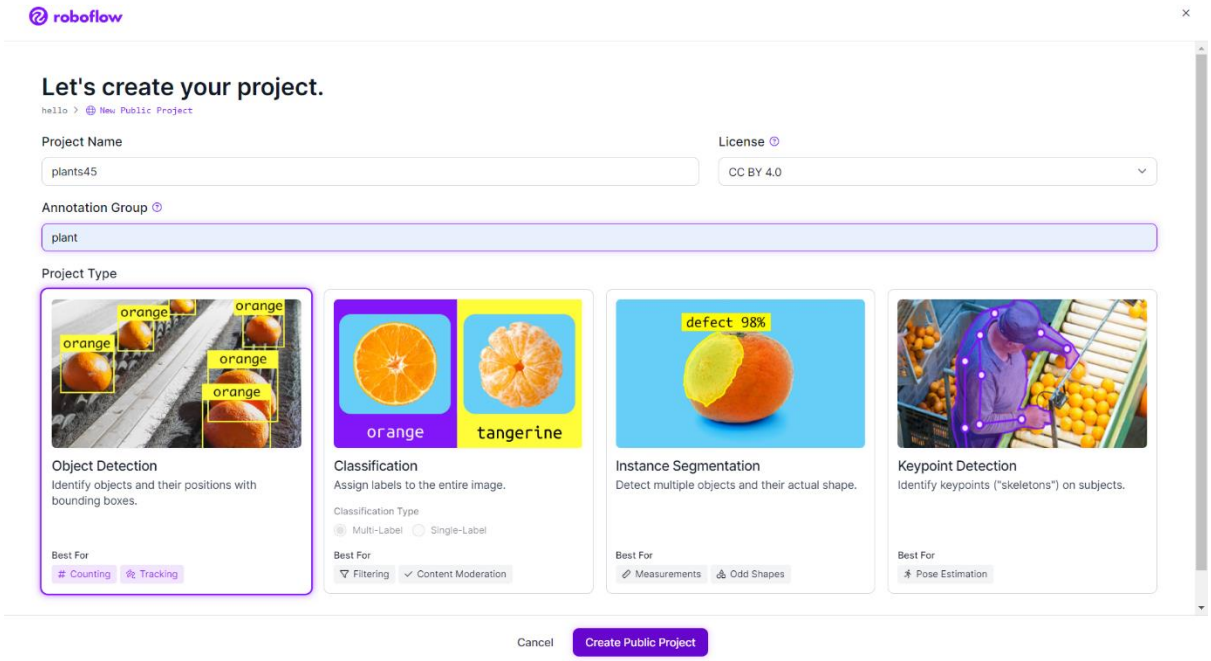
Methodology:

All navigation-related implementations and configurations applied in Task 1 will remain unchanged and are fully applicable to Task 2. The commands provided in the next section are to be used with the navigation stack.

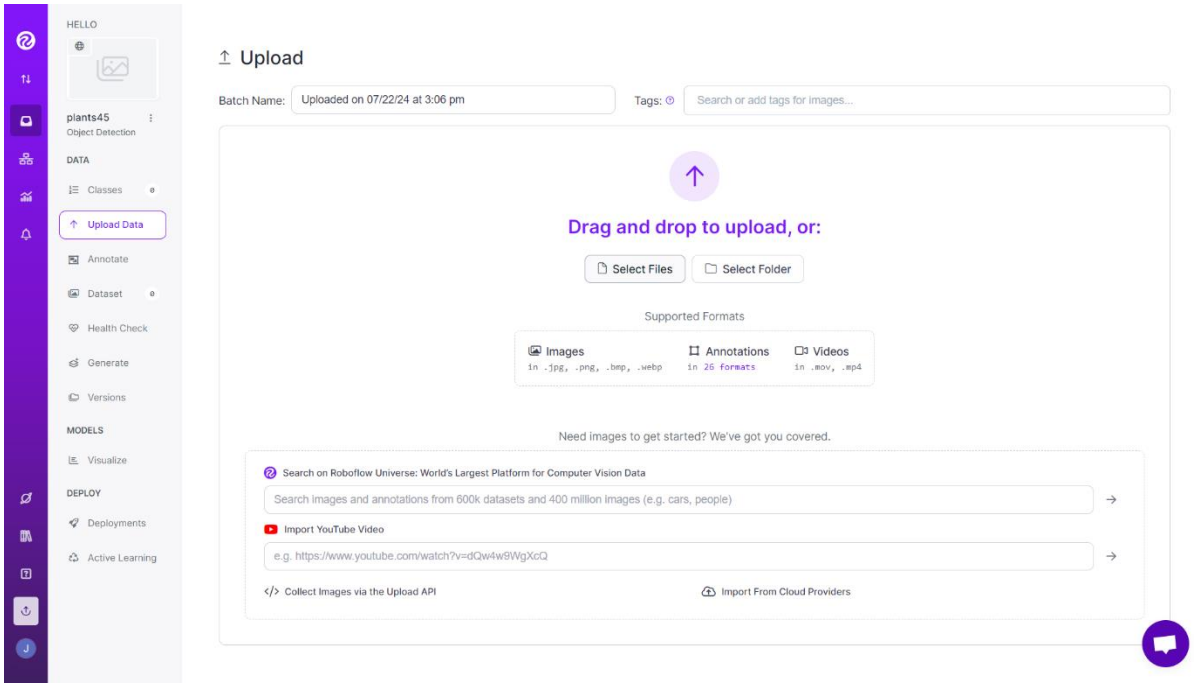
Task2: Plant tracking and counting

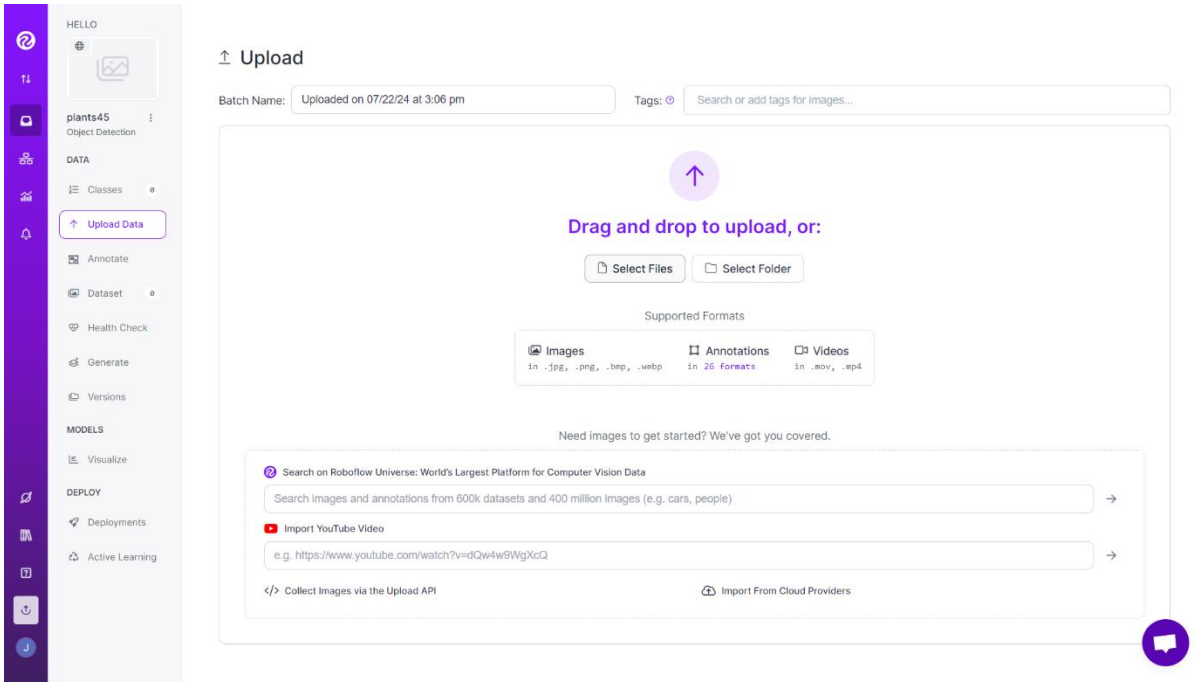
- Step1: Images collection (HD quality images will be better)
- Step2: Image annotation and dataset creation (use Roboflow)
  - Create a new project



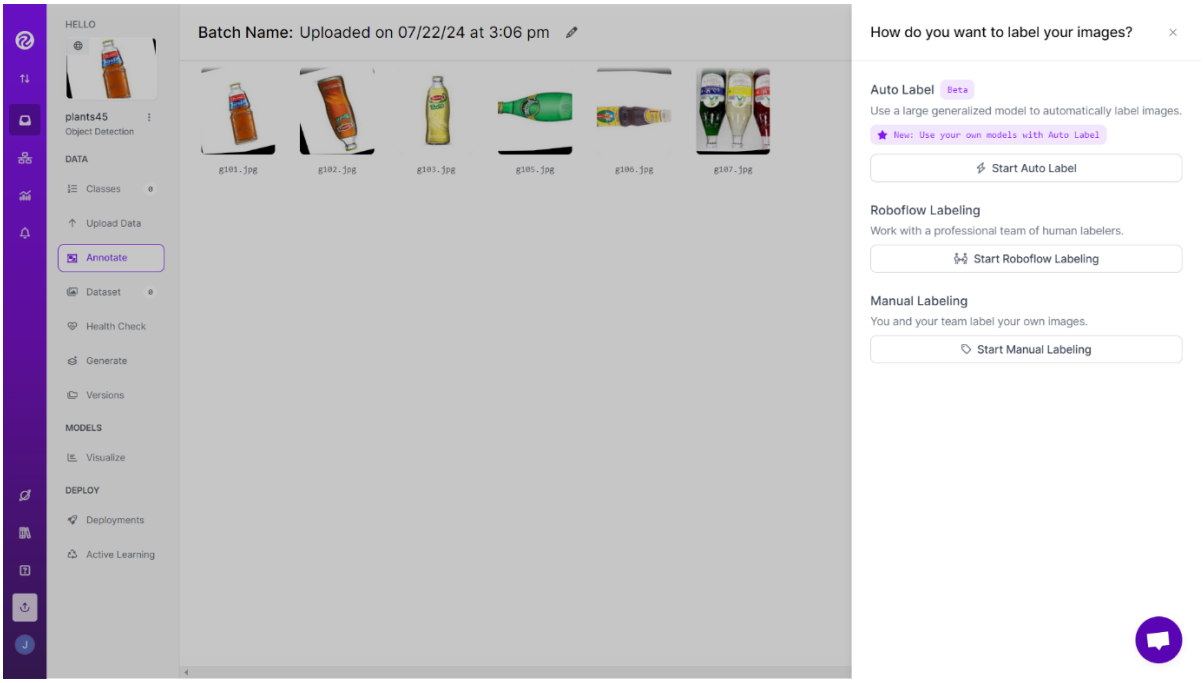


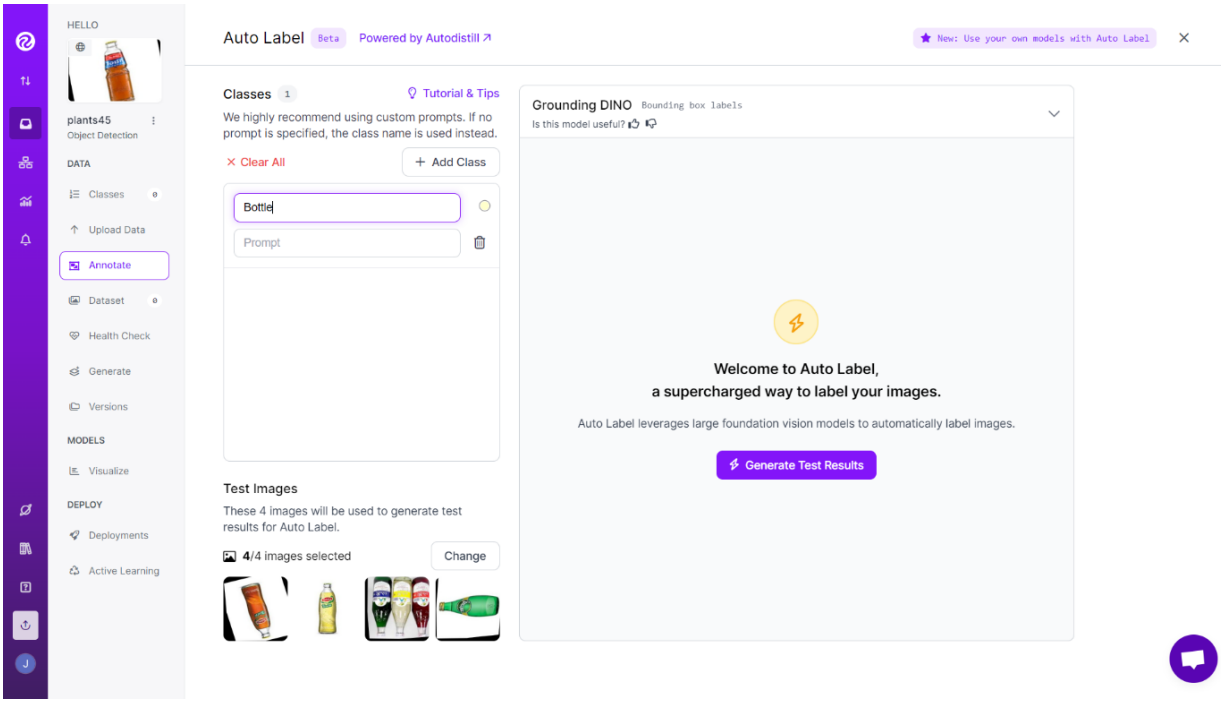
- Import images

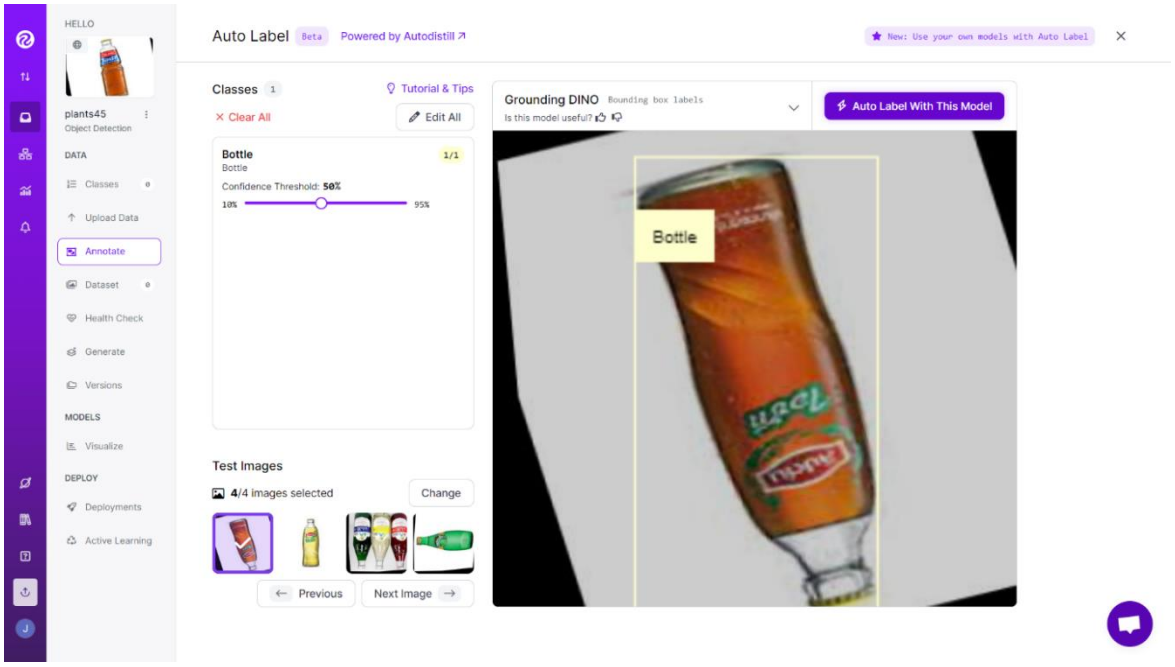


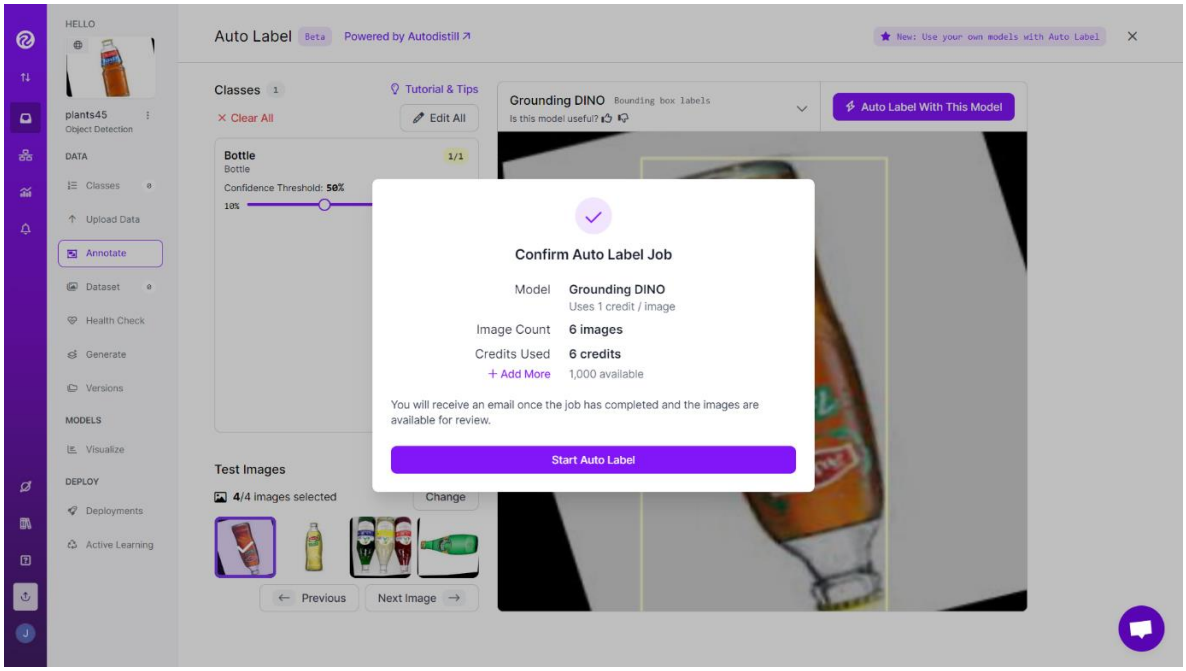


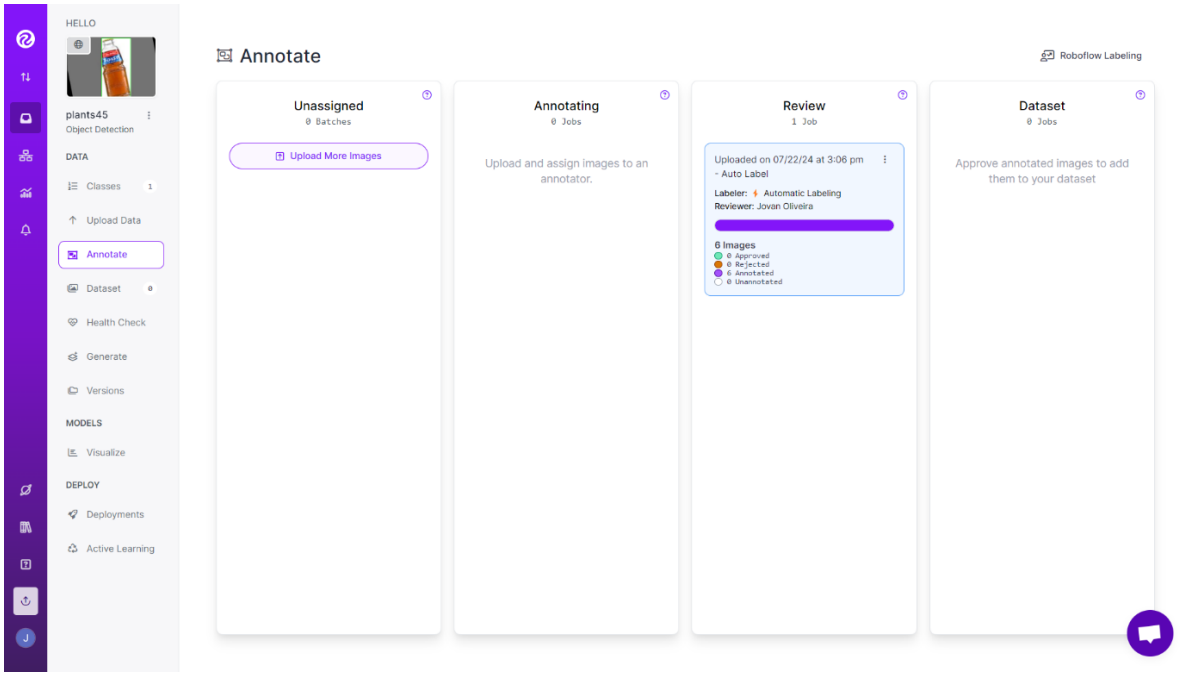
- Do annotation using smart annotation provided by Roboflow

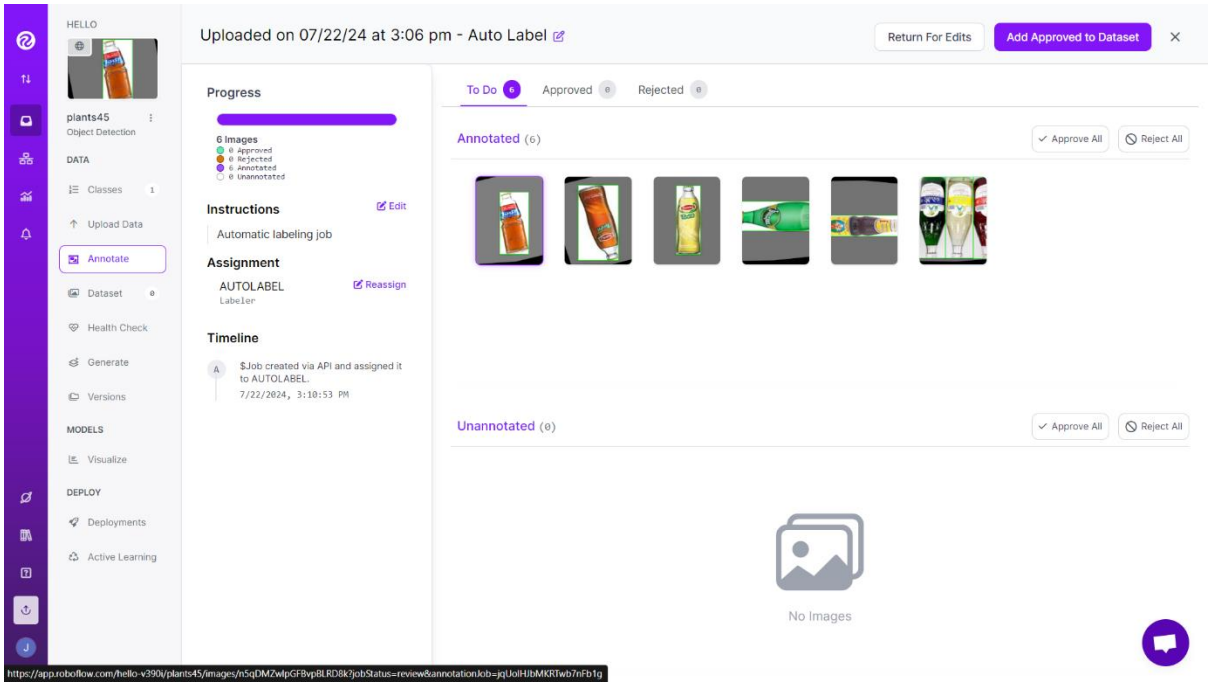


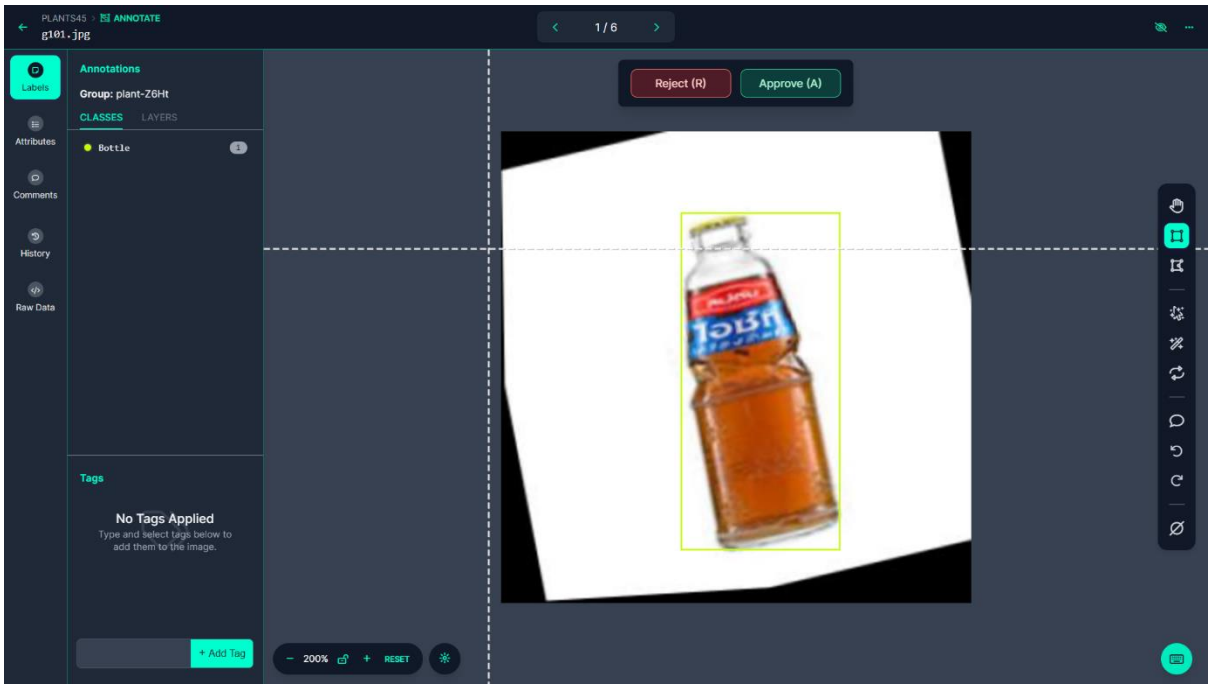

HELLO
plants45
Object Detection
DATA
Classes
Upload Data
Annotate
Dataset
Health Check
Generate
Versions
MODELS
Visualize
DEPLOY
Deployments
Active Learning
Auto Label
Beta
Powered by Autodistill
New: Use your own models with Auto Label
Classes
Tutorial & Tips
We highly recommend using custom prompts. If no prompt is specified, the class name is used instead.
Clear All
Add Class
Bottle
Prompt
Test Images
These 4 images will be used to generate test results for Auto Label.
4/4 images selected
Change
Grounding DINO
Bounding box labels
Is this model useful?
Welcome to Auto Label,
a supercharged way to label your images.
Auto Label leverages large foundation vision models to automatically label images.
Generate Test Results

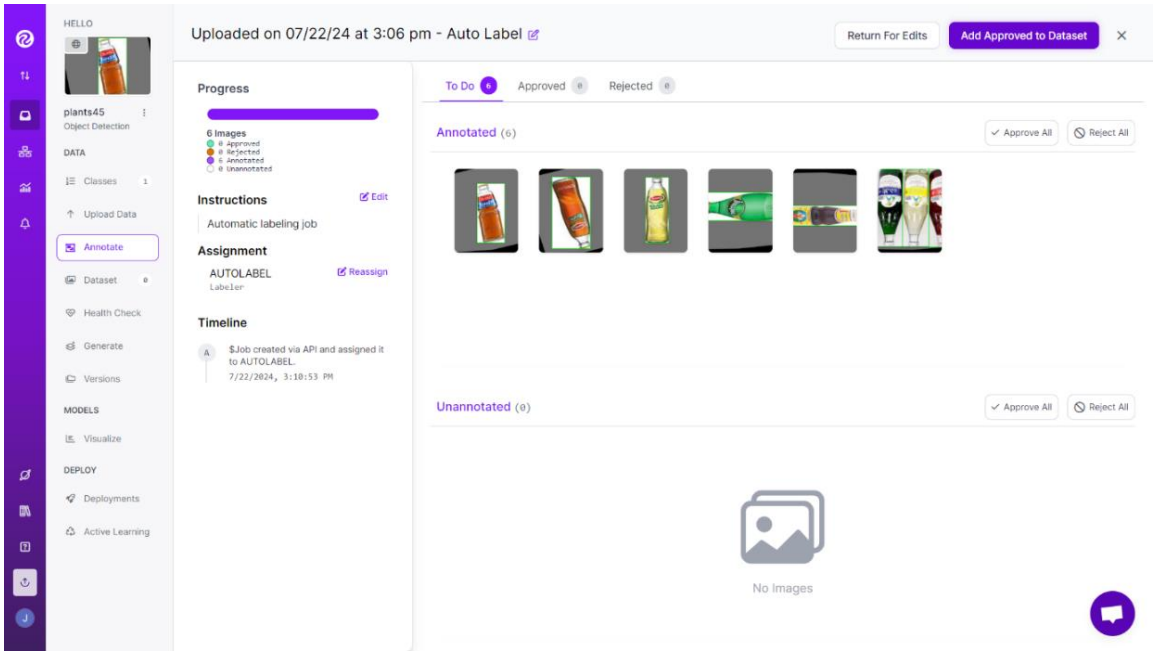

HELLO
plants45
Object Detection
DATA
Classes
Upload Data
Annotate
Dataset
Health Check
Generate
Versions
MODELS
Visualize
DEPLOY
Deployments
Active Learning
Auto Label
Beta
Powered by Autodistill
New: Use your own models with Auto Label
Classes
Tutorial & Tips
Clear All
Edit All
Bottle
Bottle
1/1
Confidence Threshold: 50%
10%
95%
Test Images
4/4 images selected
Change
Previous
Next Image
Grounding DINO
Bounding box labels
Is this model useful?
Auto Label With This Model
Bottle

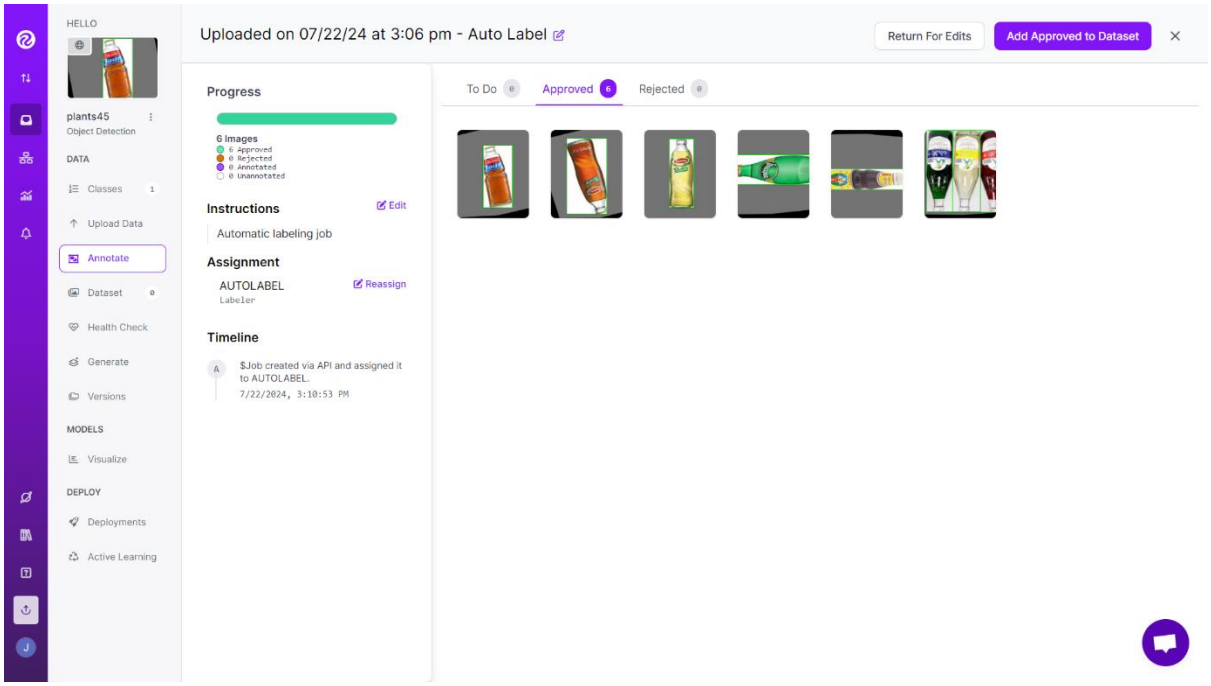

HELLO
plants45
Object Detection
DATA
Classes
Upload Data
Annotate
Dataset
Health Check
Generate
Versions
MODELS
Visualize
DEPLOY
Deployments
Active Learning
Auto Label
Beta
Powered by Autodistill
New: Use your own models with Auto Label
Classes
Tutorial & Tips
Clear All
Edit All
Bottle
Bottle
Confidence Threshold: 50%
Grounding DINO
Bounding box labels
Is this model useful?
Auto Label With This Model
Confirm Auto Label Job
Model
Grounding DINO
Uses 1 credit / image
Image Count
6 images
Credits Used
6 credits
+ Add More
1,000 available
You will receive an email once the job has completed and the images are available for review.
Start Auto Label
Test Images
4/4 images selected
Change
Previous
Next Image


HELLO
plants45
Object Detection
DATA
Classes
Upload Data
Annotate
Dataset
Health Check
Generate
Versions
MODELS
Visualize
DEPLOY
Deployments
Active Learning
Annotate
Roboflow Labeling
Unassigned
0 Batches
Upload More Images
Annotating
0 Jobs
Upload and assign images to an annotator.
Review
1 Job
Uploaded on 07/22/24 at 3:06 pm
- Auto Label
Labeler: Automatic Labeling
Reviewer: Jovan Oliveira
6 Images
0 Approved
0 Rejected
6 Annotated
0 Unannotated
Dataset
0 Jobs
Approve annotated images to add them to your dataset

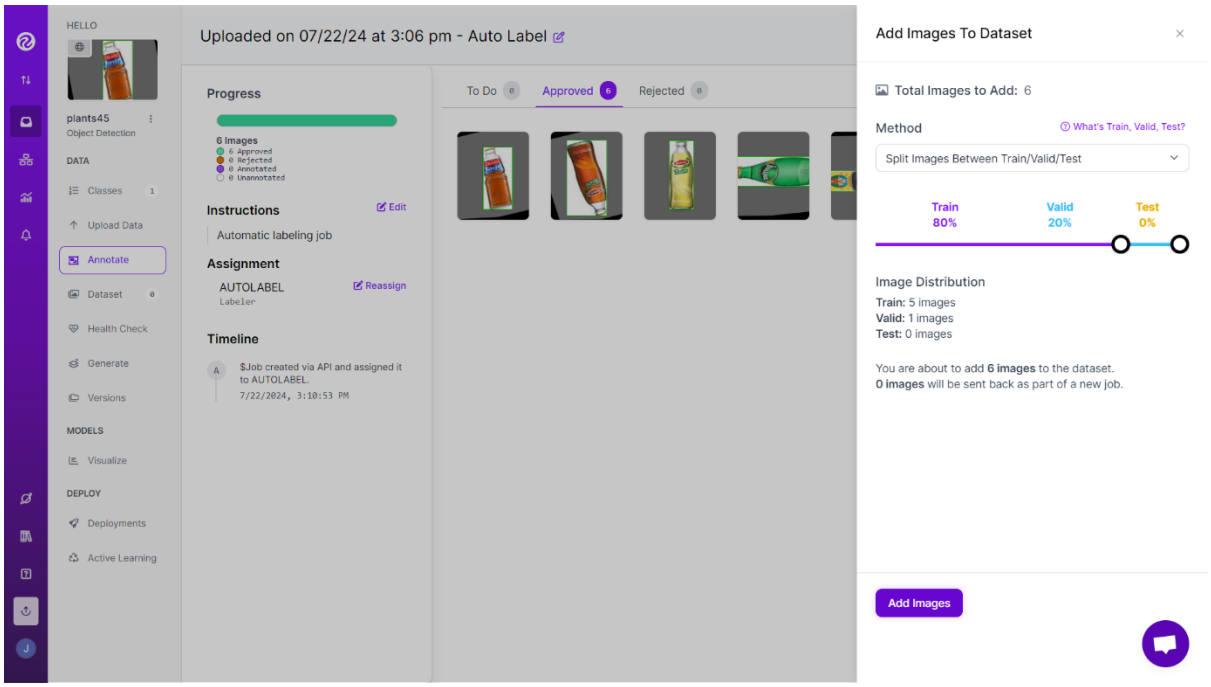

HELLO
plants45
Object Detection
DATA
Classes
Upload Data
Annotate
Dataset
Health Check
Generate
Versions
MODELS
Visualize
DEPLOY
Deployments
Active Learning
Uploaded on 07/22/24 at 3:06 pm - Auto Label
Return For Edits
Add Approved to Dataset
Progress
6 Images
0 Approved
0 Rejected
6 Annotated
0 Unannotated
Instructions
Edit
Automatic labeling job
Assignment
AUTOLABEL
Labeler
Reassign
Timeline
$Job created via API and assigned it to AUTOLABEL.
7/22/2024, 3:10:53 PM
To Do 6
Approved 0
Rejected 0
Annotated (6)
Approve All
Reject All
Unannotated (0)
Approve All
Reject All
No Images
https://app.roboflow.com/hello-v390i/plants45/images/n5qDMZwlpGF8vp8LRD8k?jobStatus=review&annotationJob=jqUolHJbMKRTwb7nFb1g


PLANTS45 ANNOTATE
g101.jpg
1 / 6
Labels
Attributes
Comments
History
Raw Data
Annotations
Group: plant-Z6Ht
CLASSES
LAYERS
Bottle 1
Tags
No Tags Applied
Type and select tags below to add them to the image.
+ Add Tag
Reject (R)
Approve (A)
200%
RESET

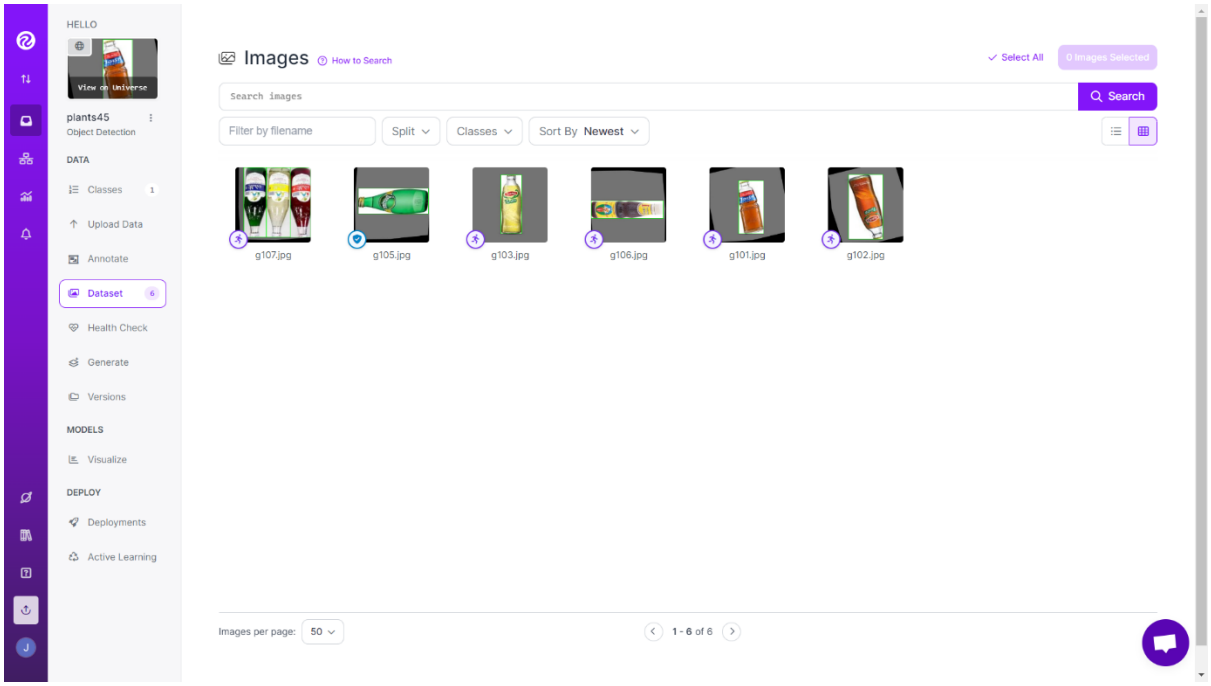

HELLO
plants45
Object Detection
DATA
Classes 1
Upload Data
Annotate
Dataset 0
Health Check
Generate
Versions
MODELS
Visualize
DEPLOY
Deployments
Active Learning
Uploaded on 07/22/24 at 3:06 pm - Auto Label
Return For Edits
Add Approved to Dataset
Progress
6 Images
6 Approved
0 Rejected
6 Annotated
0 Unannotated
Instructions
Edit
Automatic labeling job
Assignment
AUTOLABEL
Labeler
Reassign
Timeline
$Job created via API and assigned it to AUTOLABEL.
7/22/2024, 3:10:53 PM
To Do 6
Approved 0
Rejected 0
Annotated (6)
Approve All
Reject All
Unannotated (0)
Approve All
Reject All
No Images

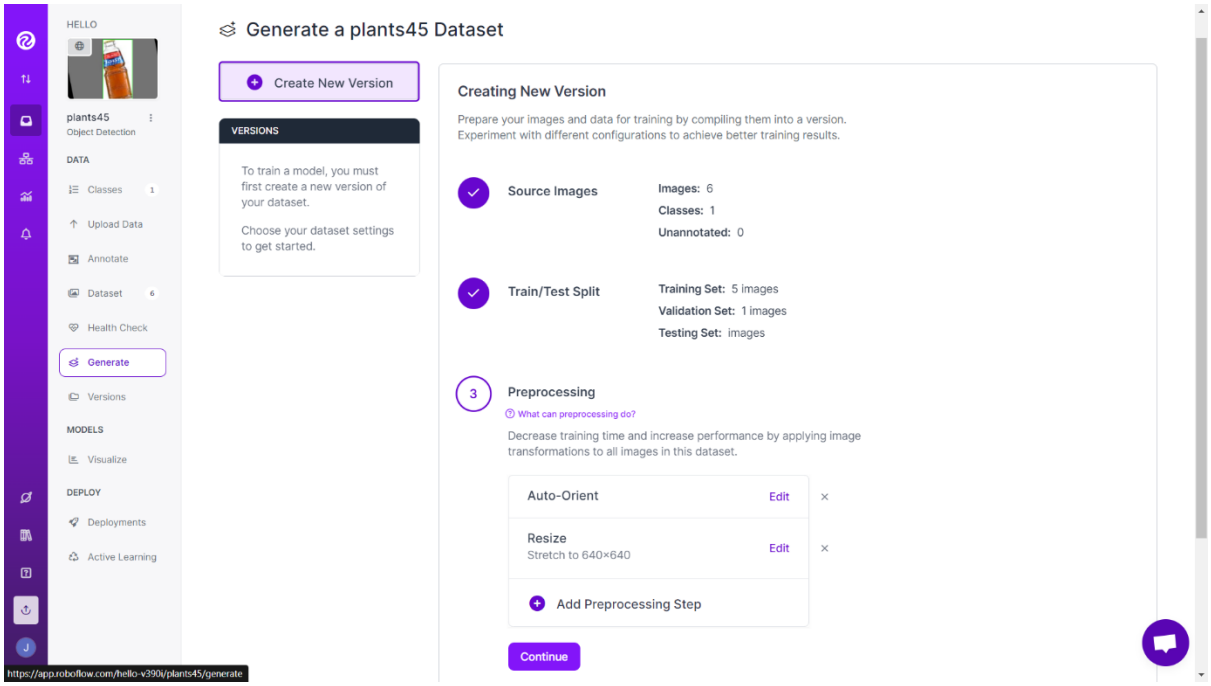

HELLO
plants45
Object Detection
DATA
Classes 1
Upload Data
Annotate
Dataset 0
Health Check
Generate
Versions
MODELS
Visualize
DEPLOY
Deployments
Active Learning
Uploaded on 07/22/24 at 3:06 pm - Auto Label
Return For Edits
Add Approved to Dataset
Progress
6 Images
6 Approved
0 Rejected
0 Annotated
0 Unannotated
Instructions
Edit
Automatic labeling job
Assignment
AUTOLABEL
Labeler
Reassign
Timeline
$Job created via API and assigned it to AUTOLABEL.
7/22/2024, 3:10:53 PM
To Do 0
Approved 6
Rejected 0

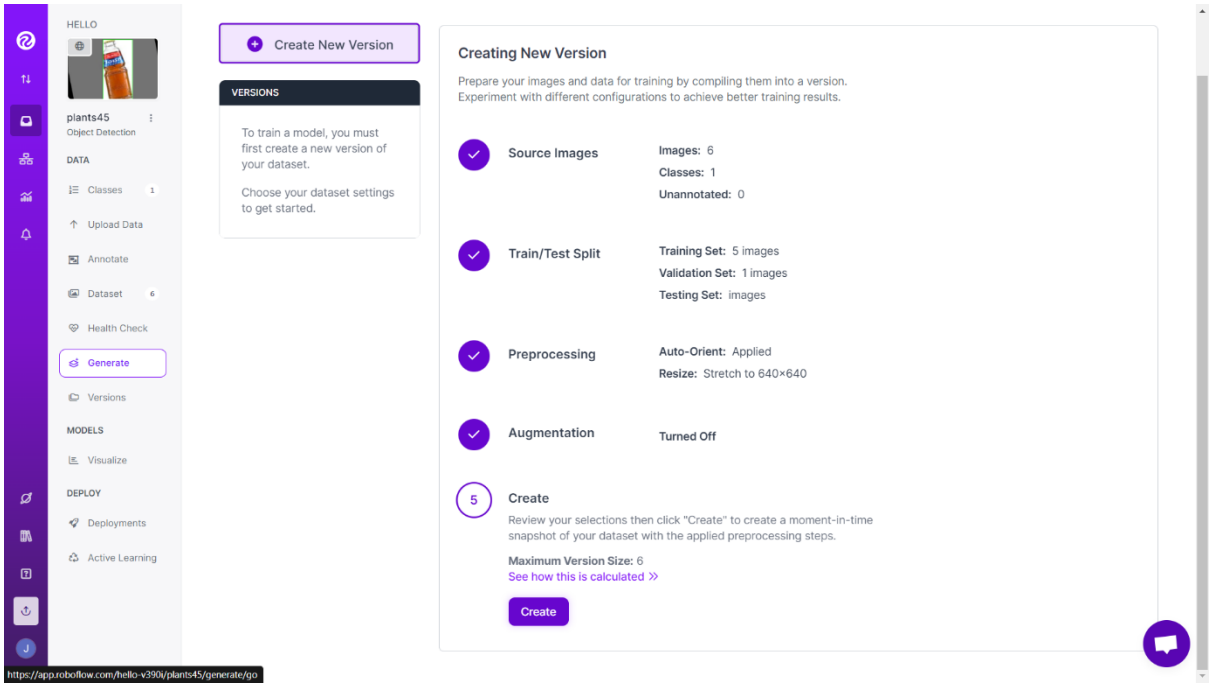
- Add annotated images to dataset and divide the dataset into 2: train (80%) and test (20%)

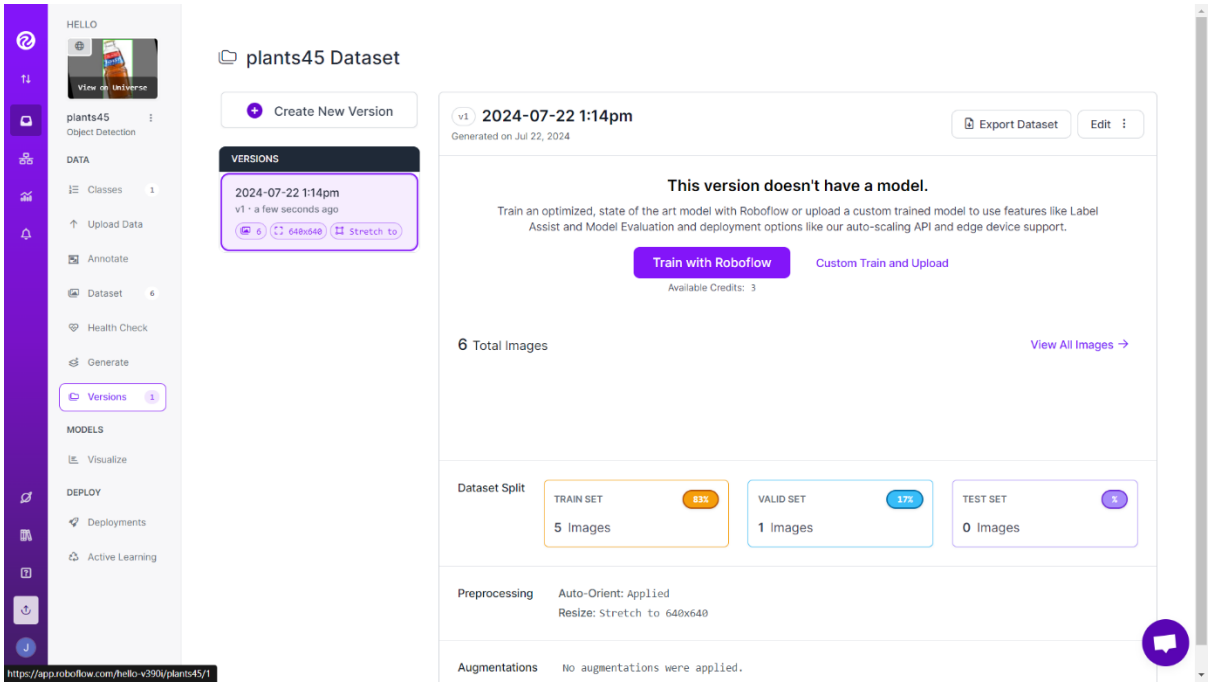


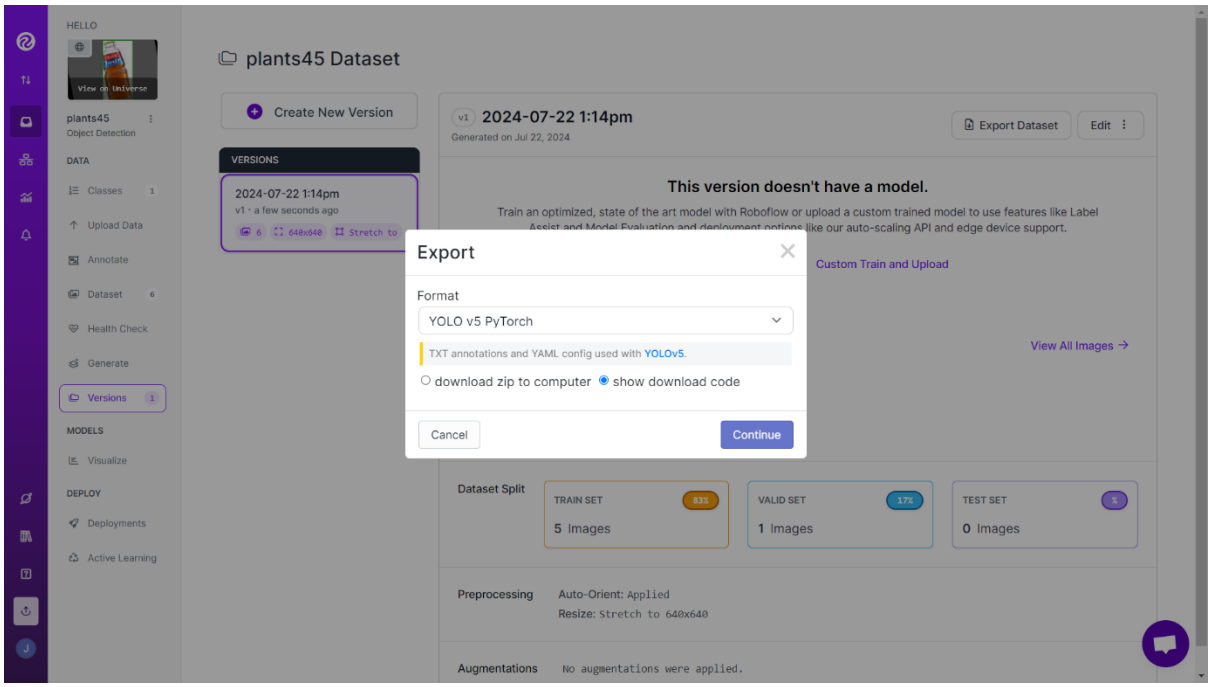
- Click on generate and create. Use the generated dataset to train a custom model.

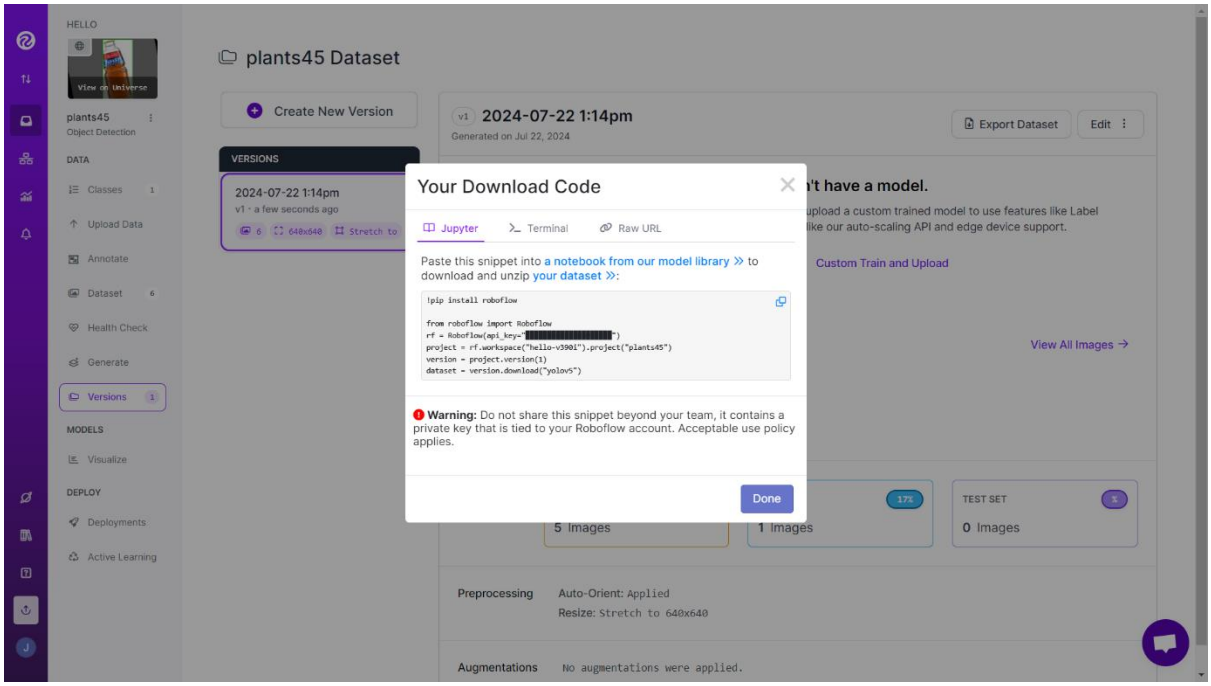



HELLO
plants45
Object Detection
DATA
Classes 1
Upload Data
Annotate
Dataset 6
Health Check
Generate
Versions
MODELS
Visualize
DEPLOY
Deployments
Active Learning
Generate a plants45 Dataset
Create New Version
VERSIONS
To train a model, you must first create a new version of your dataset.
Choose your dataset settings to get started.
Creating New Version
Prepare your images and data for training by compiling them into a version. Experiment with different configurations to achieve better training results.
Source Images
Images: 6
Classes: 1
Unannotated: 0
Train/Test Split
Training Set: 5 images
Validation Set: 1 images
Testing Set: images
3
Preprocessing
What can preprocessing do?
Decrease training time and increase performance by applying image transformations to all images in this dataset.
Auto-Orient Edit
Resize Edit
Stretch to 640×640
Add Preprocessing Step
Continue
https://app.roboflow.com/hello-v390i/plants45/generate

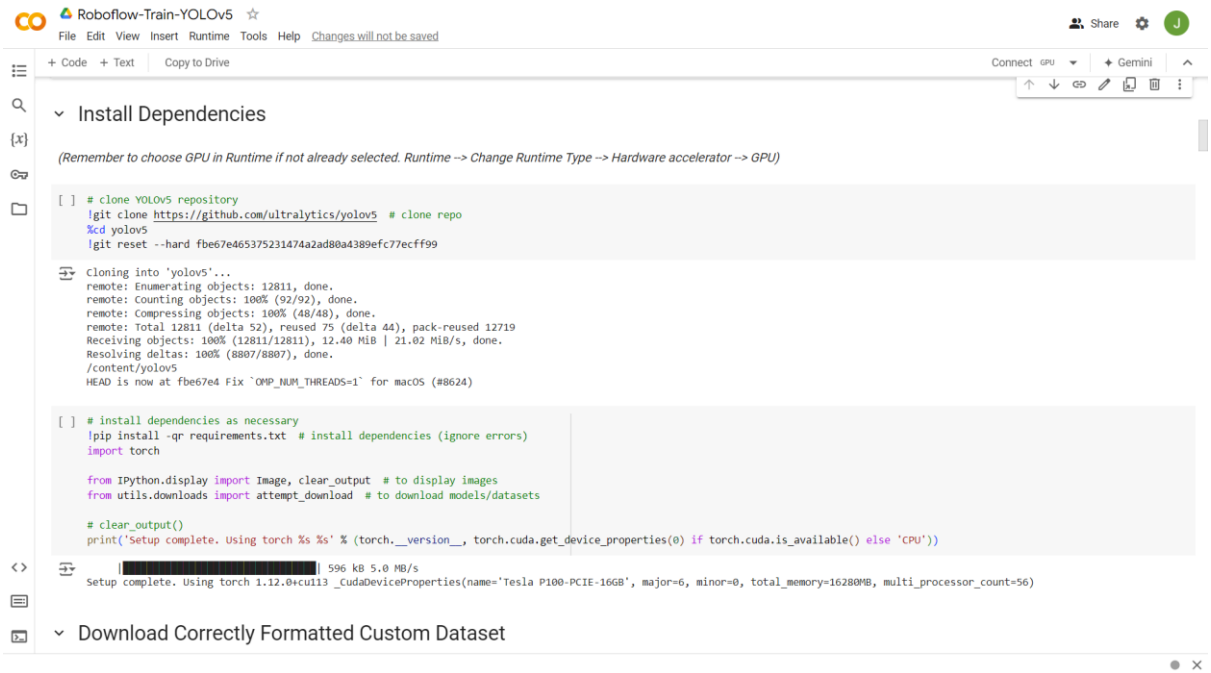

HELLO
plants45
Object Detection
DATA
Classes 1
Upload Data
Annotate
Dataset 6
Health Check
Generate
Versions
MODELS
Visualize
DEPLOY
Deployments
Active Learning
Create New Version
VERSIONS
To train a model, you must first create a new version of your dataset.
Choose your dataset settings to get started.
Creating New Version
Prepare your images and data for training by compiling them into a version. Experiment with different configurations to achieve better training results.
Source Images
Images: 6
Classes: 1
Unannotated: 0
Train/Test Split
Training Set: 5 images
Validation Set: 1 images
Testing Set: images
Preprocessing
Auto-Orient: Applied
Resize: Stretch to 640×640
Augmentation
Turned Off
5
Create
Review your selections then click "Create" to create a moment-in-time snapshot of your dataset with the applied preprocessing steps.
Maximum Version Size: 6
See how this is calculated »
Create
https://app.roboflow.com/hello-v390i/plants45/generate/go


HELLO
View on Universe
plants45
Object Detection
DATA
Classes 1
Upload Data
Annotate
Dataset 6
Health Check
Generate
Versions 1
MODELS
Visualize
DEPLOY
Deployments
Active Learning
https://app.roboflow.com/hello-v390i/plants45/1
plants45 Dataset
Create New Version
VERSIONS
2024-07-22 1:14pm
v1 · a few seconds ago
6 640x640 Stretch to
v1 2024-07-22 1:14pm
Generated on Jul 22, 2024
Export Dataset
Edit
This version doesn't have a model.
Train an optimized, state of the art model with Roboflow or upload a custom trained model to use features like Label Assist and Model Evaluation and deployment options like our auto-scaling API and edge device support.
Train with Roboflow
Custom Train and Upload
Available Credits: 3
6 Total Images
View All Images →
Dataset Split
TRAIN SET 83%
5 Images
VALID SET 17%
1 Images
TEST SET %
0 Images
Preprocessing
Auto-Orient: Applied
Resize: Stretch to 640x640
Augmentations
No augmentations were applied.

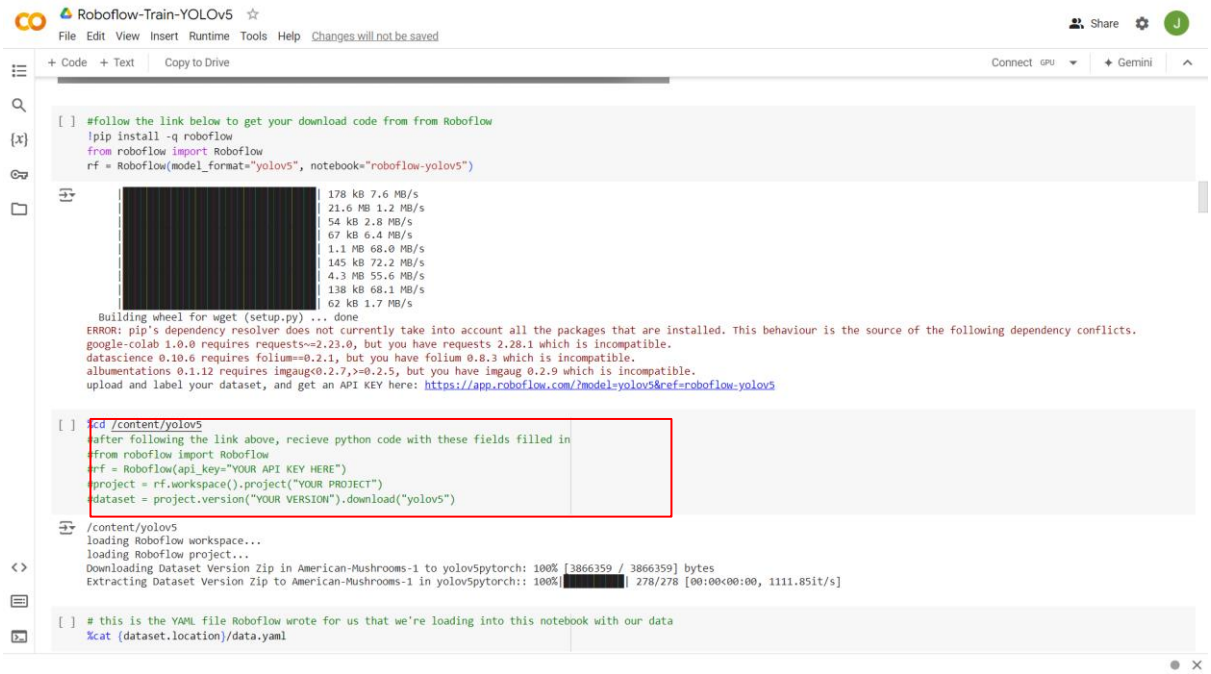

HELLO
View on Universe
plants45
Object Detection
DATA
Classes 1
Upload Data
Annotate
Dataset 6
Health Check
Generate
Versions 1
MODELS
Visualize
DEPLOY
Deployments
Active Learning
plants45 Dataset
Create New Version
VERSIONS
2024-07-22 1:14pm
v1 · a few seconds ago
v1 2024-07-22 1:14pm
Generated on Jul 22, 2024
Export Dataset
Edit
This version doesn't have a model.
Train an optimized, state of the art model with Roboflow or upload a custom trained model to use features like Label Assist and Model Evaluation and deployment options like our auto-scaling API and edge device support.
Custom Train and Upload
View All Images →
Export
Format
YOLO v5 PyTorch
TXT annotations and YAML config used with YOLOv5.
download zip to computer
show download code
Cancel
Continue
Dataset Split
TRAIN SET 83%
5 Images
VALID SET 17%
1 Images
TEST SET %
0 Images
Preprocessing
Auto-Orient: Applied
Resize: Stretch to 640x640
Augmentations
No augmentations were applied.



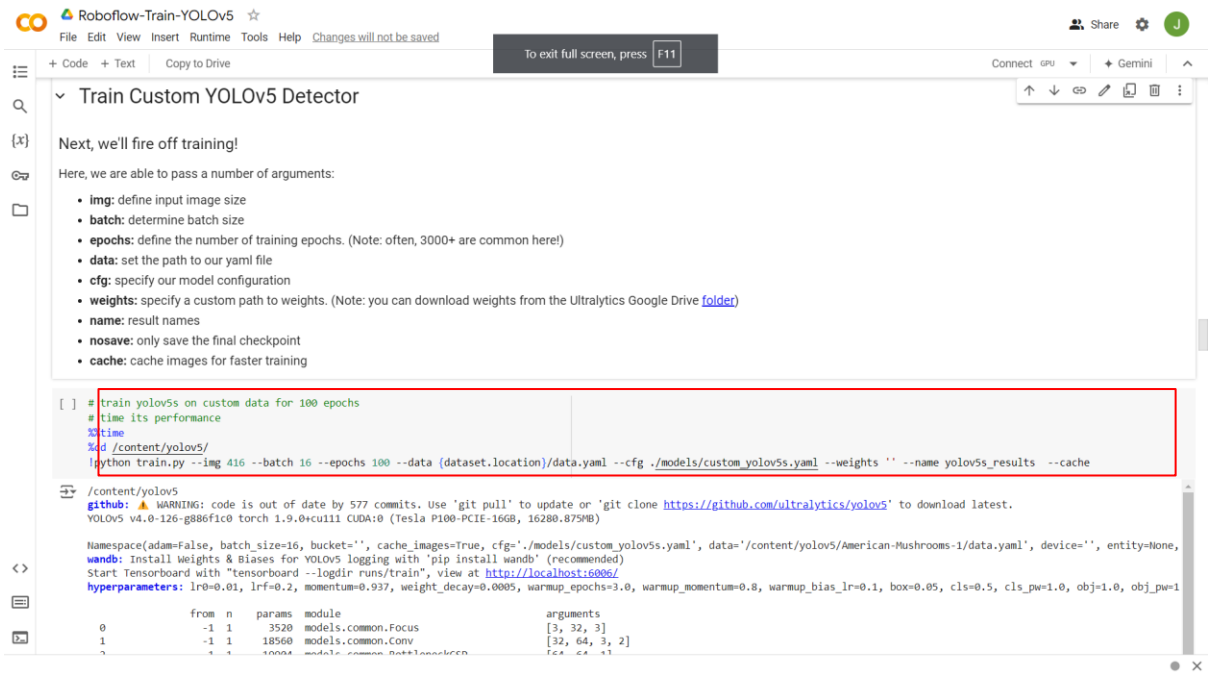
- Step3: training the model
  - Use the provided collab notebook
  - https://colab.research.google.com/drive/1gDZ2xcTOgR39tGGs-EZ6i3RTs16wmzZQ



- Enter the generated API KEY below.



- Before running the highlighted box,
  - write –weights yolov5s.pt, in place of –weights ‘ ‘
  - go to yolov5/utils/dataloaders.py, on the lines 476 and 518, make (np.int) → (int), save the file and then run the highlighted block



- The generated weight (best.pt) file is saved in the yolov5/run folder.

- Step4: Execution
  - Git clone the FRE perception repository into the **src** folder of your workspace

```
git clone
```

  - Build the workspace

```
colcon build
```

  - Navigate in the **src/obj_track/src/Tracking/**
    - The main node is **object_tracker_new.py**
    - The main consideration is to add the generated weight file (best.pt) path in line no. 14 of the node.
  - Save this file and build the workspace again

```
colcon build
```

- Step5: Running the node
  - Source the workspace

```
cd ~/your_workspace
source install/setup.bash
```

  - Run the node

```
ros2 run obj_track object_tracker_new.py
```

Description: We have to connect the realsense camera before Step5. Furthermore, the code splits the image frame from the center to take the count of the plants on the left and right separately. Here self.row input is given as default which allows it to count right and left row plants of the first lane. Furthermore, at the end of row 1, the input self.row will be 2, given by the navigation stack while the robot will go in the third lane where it counts the third and fourth rows. By default it uses self.start as start. When the robot reaches the end of lane 1, the navigation stack will provide stop which will stop the counting. Furthermore, entering in the third lane will trigger start again which will start counting the third and fourth row. The output data is published in topic *count_publisher* where the count is published in format *"row_number$_x$ – count$_x$/row_number$_{x+1}$ – count$_{x+1}$"*

11. Task 3: Flower Detection and Coordinate Estimate

- Step1: images collection (HD quality images will be better)
- Step2: Image Annotation and Dataset Creation (use Roboflow)
  - Create new project
  - Import images
  - Do annotation using smart annotation provided by Roboflow
  - Add annotated images to dataset and divide the dataset into 2: train (80%) and test (20%)
  - Click on generate and create. Use the generated dataset to train a custom model.
- Step3: Training the Model
  - Use the provided colab notebook
  - The generated weight (best.pt) file is saved in the run folder.
- Step4: Execution
  - Git clone the FRE perception repository into the src folder of your workspace.

```
git clone
```

  - Build the workspace

```
colcon build
```

  - Navigate in the **src/obj_track/src/**
    - The main node for this task is **mainrevised3.py**
    - The main consideration is to add the generated weight file (best.pt) path in line no. 17 of the node.
  - Save this file and build the workspace again

```
colcon build
```

- Step5: Running the node
  - Source the workspace

```
cd ~/your_workspace
source install/setup.bash
```

  - Run the node

```
ros2 run obj_track mainrevised3.py
```

Description: We have to connect the realsense camera before Step 5. The convention for coordinate transform of the flowers is referenced in the diagram below. The node subscribes to /odometry/transformed from the Robot. This topic publishes the x,y and yaw from the robot odometry(origin). Yaw is converted into radians from quaternion. The camera is in the positive y direction from the robot. Therefore, half of robot length(distance between camera and IMU) (0.34m) was added to the depth. The coordinates are published in topic detections_publisher in float in the form “x-coordinate + y-coordinate”

*All the images for training and validation are added in the github repository

Autonomous Navigation: This document explains how to use Nav2 with SLAM. The following steps show ROS 2 users how to generate occupancy grid maps and use Nav2 to move their robot around. This tutorial applies to both simulated and physical robots, but will be completed here on a physical robot.In this tutorial, we will be using the SLAM Toolbox.

GitHub Links:

Nav2 Navigation: https://github.com/ros-navigation/navigation2/tree/humble

Slam Toolbox: https://github.com/SteveMacenski/slam_toolbox/tree/humble

FRE_2024: https://github.com/thommandra65/FRE_2024

- Step1:
  - Launch your robot's interface and robot state publisher,

```
ros2 launch vesc_driver vesc_driver_node.launch.py
ros2 launch odometry_estimator robot_odometry_filtered.launch.py
```

- Step2:

```
ros2 launch nav2_bringup navigation.launch.py
```

- Step3:

```
ros2 launch slam_toolbox online_async.launch.py
```

- Step4:

```
ros2 launch FRE_2024 robot_move.py
```

12. Task 4: Precise Flower Detection and Position Estimate for Spray

Autonomous Navigation: This document explains how to use Nav2 with SLAM. The following steps show ROS 2 users how to generate occupancy grid maps and use Nav2 to move their robot around. This tutorial applies to both simulated and physical robots, but will be completed here on a physical robot. In this tutorial, we will be using the SLAM Toolbox.

GitHub Links:

Nav2 Navigation: https://github.com/ros-navigation/navigation2/tree/humble

Slam Toolbox: https://github.com/SteveMacenski/slam_toolbox/tree/humble

FRE_2024: https://github.com/thommandra65/FRE_2024

- Step1:
  - Launch your robot's interface and robot state publisher,

```
ros2 launch vesc_driver vesc_driver_node.launch.py
ros2 launch odometry_estimator robot_odometry_filtered.launch.py
```

- Step2:

```
ros2 launch nav2_bringup navigation.launch.py
```

- Step3:

```
ros2 launch slam_toolbox online_async.launch.py
```

- Step4:

```
ros2 launch FRE_2024 robot_move.py
```

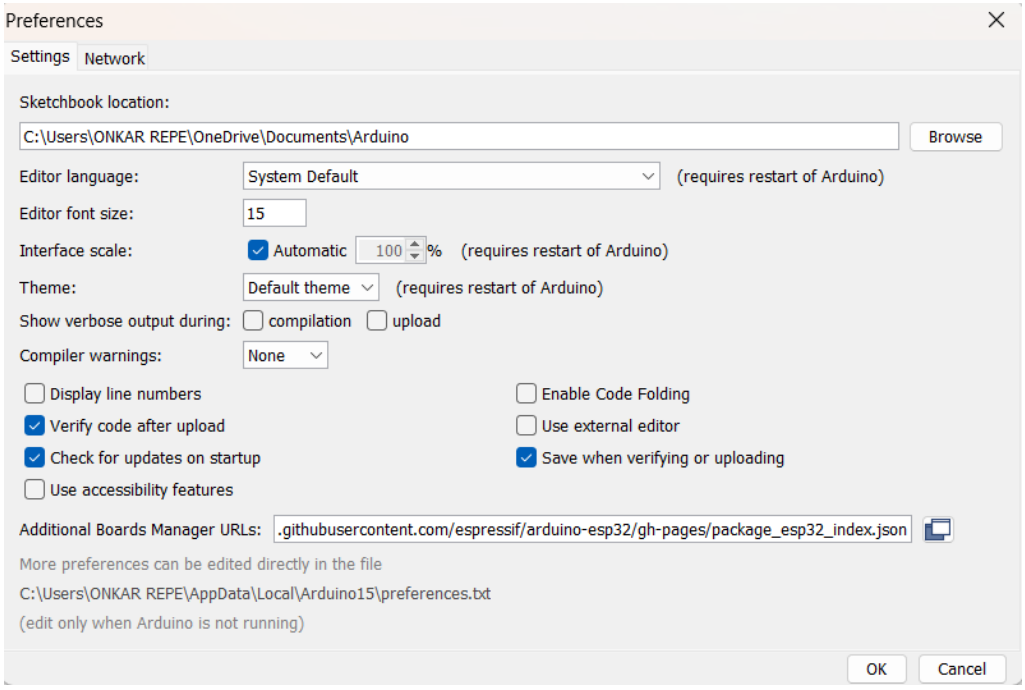
Setup Micro-ROS with Esp32

YouTube Guide Link: https://youtu.be/qtVFsgTG3AA?si=frLgHxoE3mRzL-se

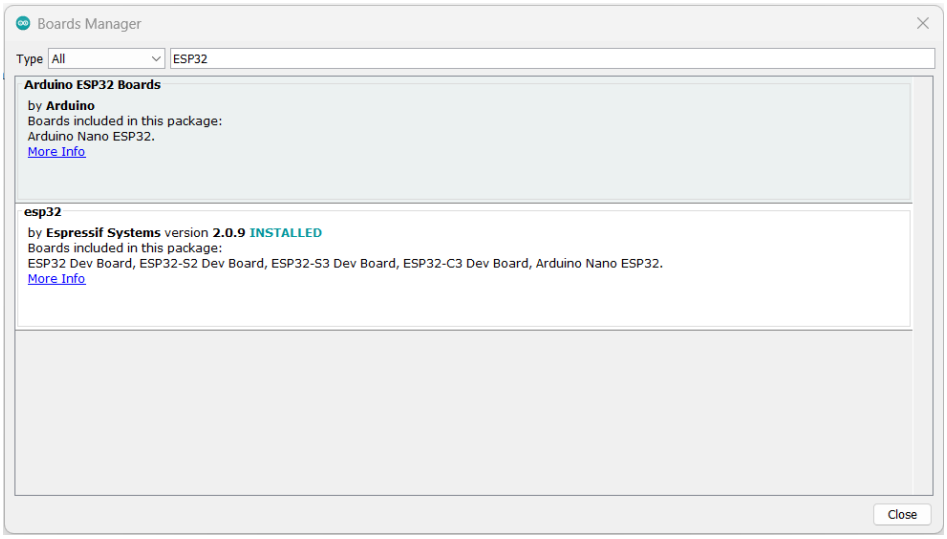
- Step 1: Install Arduino

```
sudo snap install arduino
```

- Step 2: To install ESP32 Board in your Arduino IDE, follow the below instructions.
  - In your Arduino IDE, go to File → Preferences
  - Enter the following into the "Additional Board Manager URLs" field: https://raw.githubusercontent.com/espressif/arduino-esp32/gh-pages/package_esp32_index.json
    Then, click the "OK" button.
    Note: if you already have the ESP8266 boards URL, you can separate the URLs with a comma as follows:
    https://raw.githubusercontent.com/espressif/arduino-esp32/gh-pages/package_esp32_index.json , http://arduino.esp8266.com/stable/package_esp8266com_index.json



- Open the Boards Manager. Go to Tools → Board → Boards Manager
- Search for ESP32 and check for the “ESP32 by Espressif Systems”



- Check and match the version of ESP32 DevModule at micro_ros_arduino repository: https://github.com/micro-ROS/micro_ros_arduino
- Version 2.0.2 used in our case.

Board	Min version	State	Details	.meta file
Arduino Portenta H7 M7 Core	v1.8.5	Supported	Official Arduino support	`colcon.meta`
Arduino Nano RP2040 Connect	v1.8.5	Supported	Official Arduino support	`colcon_verylowmem.meta`
OpenCR	v1.4.16	Supported	Based on custom board	`colcon.meta`
Teensy 4.0	v1.8.5	Not tested	Based on Teensyduino (1.58.x)	`colcon.meta`
Teensy 4.1	v1.8.5	Supported	Based on Teensyduino (1.58.x)	`colcon.meta`
Teensy 3.2/3.1	v1.8.5	Supported	Based on Teensyduino (1.58.x)	`colcon_lowmem.meta`
Teensy 3.5	v1.8.5	Not tested	Based on Teensyduino (1.58.x)	`colcon_lowmem.meta`
Teensy 3.6	v1.8.5	Supported	Based on Teensyduino (1.58.x)	`colcon_lowmem.meta`
ESP32 Dev Module	v1.8.5	Supported	Arduino core for the ESP32 (v2.0.2)	`colcon.meta`

- Click the Install button. It should be installed after a few seconds.

- Step 3: Micro ROS library addition
  - Go to micro_ros_arduino GitHub repository and select 'humble' branch
    - https://github.com/micro-ROS/micro_ros_arduino/tree/humble
  - Download it as a zip file
  - In Arduino IDE go to Sketch → Include Library → Add ZIP Library
  - Select the downloaded zip file and click "Open"

- Step 4: In Arduino IDE, go to Tools → Board: "Arduino Uno" → ESP32 Arduino → ESP32 Dev Module

- Step 5: Check Installation
  - Connect the ESP32 to USB 3.0
  - Select the port, Go to Tools → Port → /dev/ttyUSB0
  - Go to File → Examples → micro_ros_arduino → micro_ros_publisher
  - Upload the program to ESP32.
  - If the LED is turned on the installation is successful

- Step 6: Micro-ROS agent, it acts as a bridge between ESP32 and ROS2 that will show all the topics that are inside the ESP32 and allow the communication.
  - Create a workspace named task4_ws
  - Go to task4_ws → src and open in terminal
  - Git clone the following repository:
    - https://github.com/micro-ROS/micro-ROS-Agent

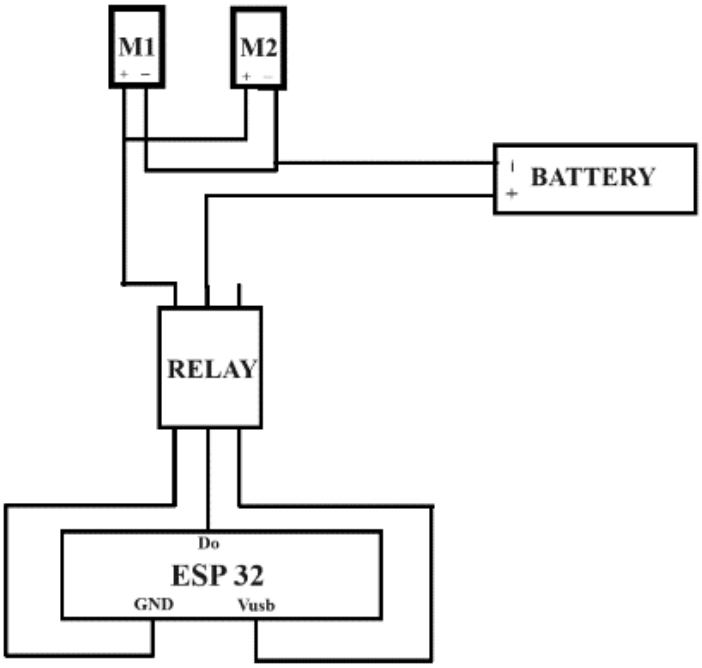
  - Install all the dependencies using the following command:

```
rosdep install --from-paths src --ignore-src -r -y
```

  - Build the workspace using colcon build

Connections of Pump

We are using 2 DC motors (M1, M2), relay module and external power source of 12V and ESP32 for the spraying application.
Do the connections as shown in the image below:



- Step 1: Copy the code provided in a file in Arduino IDE and compile and upload it on ESP32

```
#include <micro_ros_arduino.h>

#include <stdio.h>
#include <rcl/rcl.h>
#include <rcl/error_handling.h>
#include <rclc/rclc.h>
#include <rclc/executor.h>

#include <std_msgs/msg/int32.h>

rcl_subscription_t subscriber;
std_msgs__msg__Int32 msg;
rclc_executor_t executor;
rclc_support_t support;
rcl_allocator_t allocator;
rcl_node_t node;

#define LED_PIN 13

#define RCCHECK(fn) { rcl_ret_t temp_rc = fn; if((temp_rc != RCL_RET_OK)){error_loop();}}
#define RCSOFTCHECK(fn) { rcl_ret_t temp_rc = fn; if((temp_rc != RCL_RET_OK)){}}

void error_loop(){
  while(1){
    digitalWrite(LED_PIN, !digitalRead(LED_PIN));
    delay(100);
  }
}

void subscription_callback(const void * msgin)
{
  const std_msgs__msg__Int32 * msg = (const std_msgs__msg__Int32 *)msgin;
  if (msg->data == 1) {
    digitalWrite(LED_PIN, HIGH); // Turn LED on
  } else if (msg->data == 0) {
    digitalWrite(LED_PIN, LOW); // Turn LED off
  }
}

void setup() {
  set_microros_transports();

  pinMode(LED_PIN, OUTPUT);
  digitalWrite(LED_PIN, LOW); // Ensure the LED is off initially

  delay(2000);

  allocator = rcl_get_default_allocator();

  // Create init_options
  RCCHECK(rclc_support_init(&support, 0, NULL, &allocator));

  // Create node
  RCCHECK(rclc_node_init_default(&node, "micro_ros_arduino_node", "", &support));

  // Create subscriber
  RCCHECK(rclc_subscription_init_default(
    &subscriber,
    &node,
    ROSIDL_GET_MSG_TYPE_SUPPORT(std_msgs, msg, Int32),
    "micro_ros_arduino_subscriber"));

  // Create executor
  RCCHECK(rclc_executor_init(&executor, &support.context, 1, &allocator));
  RCCHECK(rclc_executor_add_subscription(&executor, &subscriber, &msg, &subscription_callback, ON_NEW_DATA));
}

void loop() {
  delay(100);
  RCCHECK(rclc_executor_spin_some(&executor, RCL_MS_TO_NS(100)));
}
```

- Step 2: Go to task4_ws/spraying/src and open pump_publisher.py
- Step 3: Add the path of the weight file for flower detection as done in task 3

- Step 4: Build and source the workspace. Run the following micro-ros agent command on the terminal.

```
ros2 run micro_ros_agent micro_ros_agent udp4 -port 8888
```

  - Once this command is run, press reset button on ESP32

- Step 5: In another terminal run the pump_publisher node.

  Description: Do the connections as shown above. The pump_subscriber on ESP32 subscribes to pump_publisher through micro-ros agent. Once the flower is detected pump_publisher publishes a flag value. Once this flag value is subscribed, the pump is turned on for 2 seconds to do the necessary spraying application. The pump remains off until next flower is detected.