Ranging Package

Ranging Package

Description:

This page discusses the *ranging_package* in detail. The main purpose of this package is to perform object vehicle ranging using YOLO object detection and monocular ranging methods.

The methods used for ranging are a simple pinhole camera model which assumes a known vehicle width and a virtual horizon model which generates a "vanishing point" on the image plane for ranging - the virtual horizon method avoids setting an assumed size for all vehicles but still uses an average vehicle width to calculate the vanishing point.

Resources on these methods can be found in these two papers:

[1] Park, Ki-Yeong, and Sun-Young Hwang. “Robust Range Estimation with a Monocular Camera for Vision-Based Forward Collision Warning System.” The Scientific World Journal 2014 (2014): 1–9. https://doi.org/10.1155/2014/923632.

[2] Flegel, T. (2023). Relative Position Vector Generation with Computer Vision for Vehicle Platooning Applications (thesis). Auburn University, Auburn.

Requirements:

- Ubuntu Focal Fossa
- ROS2 Foxy Fitzroy
- C++17 or higher

Configuration and Launch Files

Before using this package, make sure ALL paths in the configuration and launch files are set correctly. This will especially cause issues when cloning the repository to a new machine as the paths cloned from the remote repo are (most likely) not valid for the new local repo.

Note: All paths are set globally. Paths are also rarely set outside of configuration/launch files. This allows the user to point to different files in a more automated manner. Editing the configuration file DOES NOT require you to rebuild the package, but editing the launch file DOES!

***Configuration File*:**
`object_position_estimation_config.yaml` is shown below:

src > ranging_package > config > ! object_position_estimation.yaml

```yaml
/**:
  ros__parameters:
    conf_threshold: 0.5
    nms_threshold: 0.4
    inp_width: 416 #224
    inp_height: 416 #224
    device: "cpu"
    input_type: "image"
    input_path: "/home/tahnt/T3_Repos/post_process_packages/ros2_ws/src/ranging_package/data/images/frame410.jpg"
    write_output: true
    output_file: "/home/tahnt/T3_Repos/post_process_packages/ros2_ws/src/ranging_package/output/images/example_output.jpg"
    classes_file: "/home/tahnt/T3_Repos/post_process_packages/ros2_ws/src/ranging_package/classes/coco.names"
    model_configuration: "/home/tahnt/T3_Repos/post_process_packages/ros2_ws/src/ranging_package/cfg/yolov3.cfg"
    model_weights: "/home/tahnt/T3_Repos/post_process_packages/ros2_ws/src/ranging_package/weights/yolov3.weights"
    camera_matrix_vector: [2.8669527693619584e+03, 0., 6.40000000000000e+02, 0., 2.8669527693619584e+03, 3.600000000000000e+02, 0., 0., 1.]
    dist_coeffs:  [-8.1211078630524319e-02, -1.1874772506987743e+00, 0., 0., 0.]
    camera_height: 2.27
```

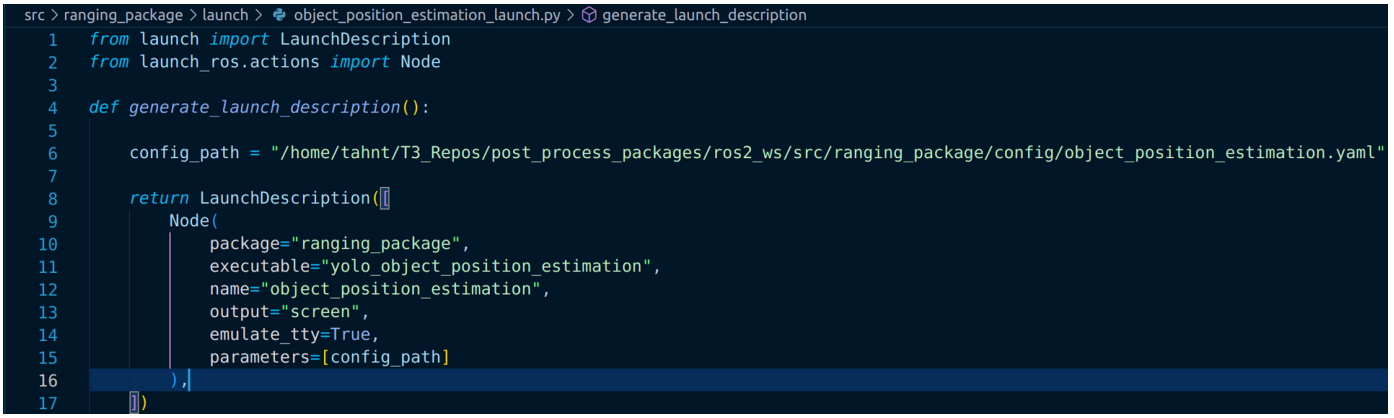
ROS2 Parameters Configured:

- **conf_threshold:** The maximum allowable confidence threshold for a YOLO detection. Any detection below the set value will not be processed.
- **nms_threshold:** Non-maximum suppression threshold for the NMSBoxes function
- **inp_width:** NOT input image width! Spatial width for the convolutional neural network. Common sizes are (224×224), (227×227), (299×299), and (416x416). Lower sizes are more efficient but less accurate and larger sizes are less efficient but more accurate.
- **inp_height:** NOT input image height! Spatial height for the convolutional neural network. Common sizes are (224×224), (227×227), (299×299), and (416x416). Lower sizes are more efficient but less accurate and larger sizes are less efficient but more accurate.
- **device:** "cpu" or "gpu" - the backend device to run the CNN.
  * **Note:** Running the CNN off GPU is obviously desired for higher computational speed however this requires a specific version of OpenCV built with with a specific version of CUDA!
- **input_type:** "image" or "video" - the input type for processing
- **input_path:** Path to the input file
- **write_output:** "true" or "false" - boolean to write/save a processed output file
- **output_file:** Path to the output file
- **classes_file:** Path to the YOLO class file
- **model_configuration:** Path to the YOLO model configuration. Original YOLOv3 and YOLOv3-tiny available in package
- **model_weights:** Path to the YOLO training weights. Original YOLOv3 and YOLOv3-tiny available in package
- **camera_matrix_vector:** Estimated camera matrix from the calibration process in **vector** form. Input is in vector form and sorted by row (11, 12, 13, 21, 22, 23, 31, 32, 33). A matrix is generated from the vector elements and used to undistort the image before processing - also used to extract focal length for range equations.
  - **Note:** There's probably a better way to do this but I ran into issues when inputing a matrix into YAML so this was a quick fix.
- **dist_coeffs:** Estimated distortion coefficients from the calibration process. Input is in vector form (same as calibration output). Used to undistort the image before processing.

- **camera_height:** Measured meter height of the camera from flat ground

***Launch File*:**

`object_position_estimation_launch.py` is shown below:



Typically, the launch file does not need to be edited often. Make sure **config_path** correctly points to `object_position_estimation_config.yaml` .

Using the Package

To run ranging follow the steps below (also included in `README.md`)

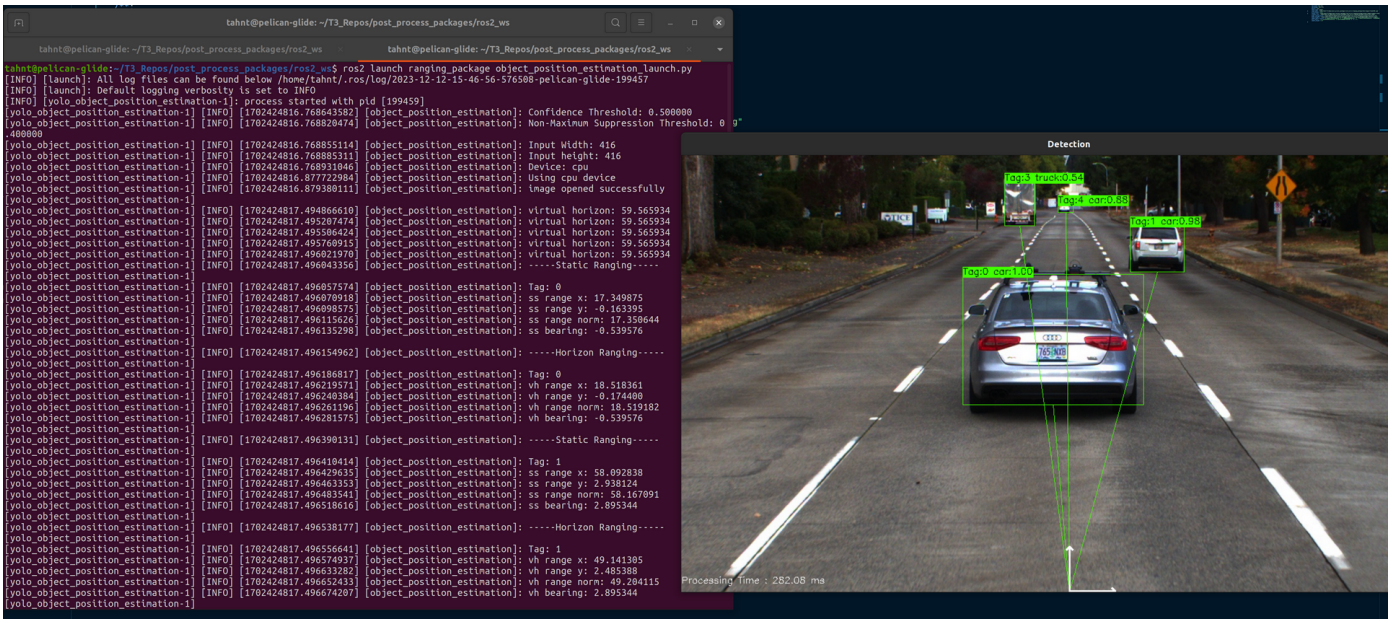
Before Use:

- **Make sure ALL PATHS ARE SET CORRECTLY in the launch and config files before use!**
- **These steps assume you have already created a workspace folder and a `/src` directory within it!**

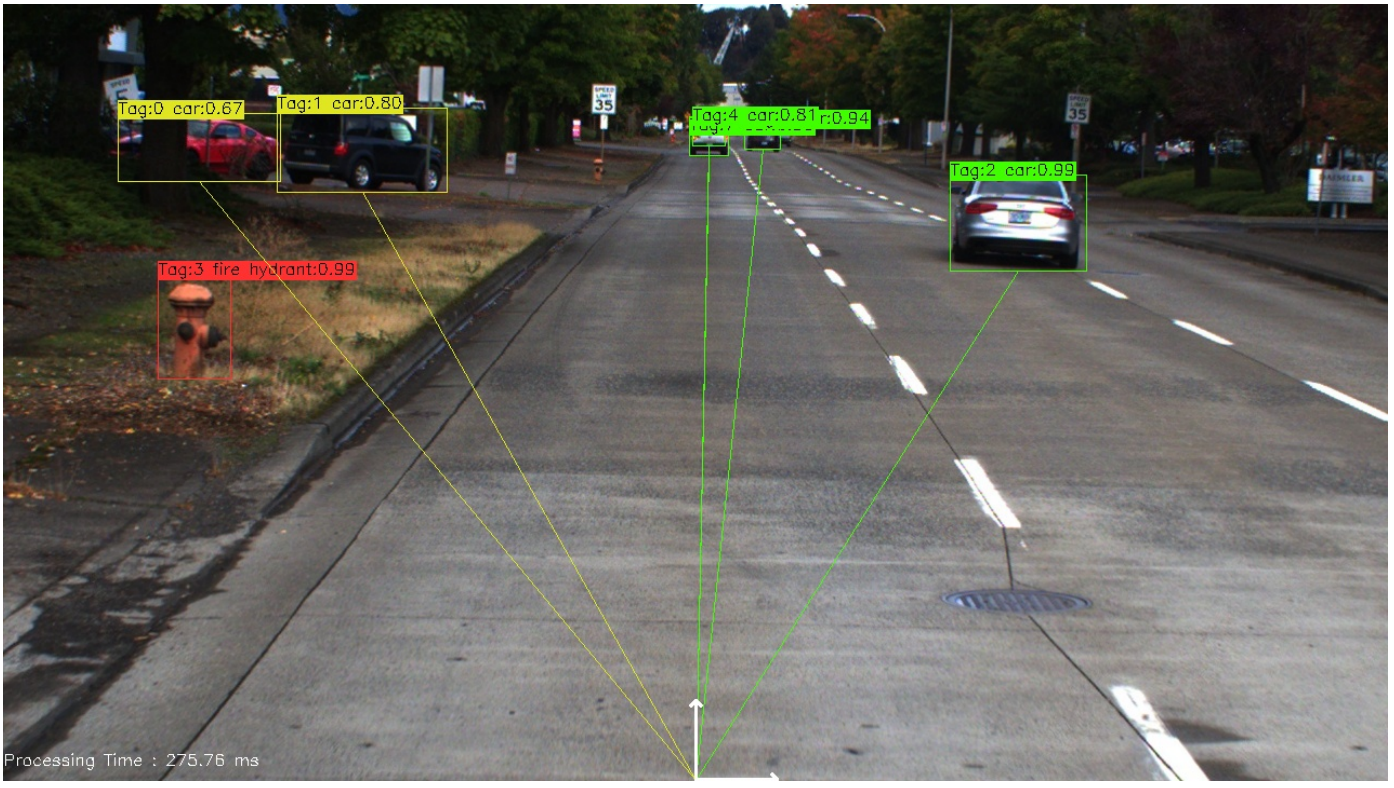
Steps:

1. Navigate into the `/src` directory of your workspace and clone the repo using `git clone`
2. Navigate back into the workspace directory and source `$ source /opt/ros/foxy/setup.bash`
3. Build package `$ colcon build` or `$ colcon build --packages-select <package_name>`
4. Open a new terminal and source it `$ . install/setup.bash`
5. Run launch file `$ ros2 launch <package_name> <launch_file_name>` in this case it is `$ ros2 launch ranging_package object_position_estimation_launch.py`

If executed correctly, the terminal should display the ranges from the static size method and the virtual horizon method for each vehicle object. Each vehicle object is indicated using tags. The output should be similar to the following:



Here is another output:



A red bounding box represents a non-vehicle object, yellow represents a filtered vehicle object, and green represents a valid vehicle detection. Ranging is performed on ALL vehicle detection **even if considered invalid**.

Note-worthy Considerations:

The main purpose of this package is to test and compare different monocular ranging techniques. Further testing is needed for validation as it is possible these methods are implemented incorrectly in code - there is no truth data so intuition is the only way to check these calculations. Controlled testing could be a good idea moving forward.

From observation, longitudinal ranging for either method is poor, however lateral ranging is often much more reliable.

The virtual horizon needs more testing and analyses as it seems to be the right approach but does not consistently produce reliable estimates.

This package is meant for testing and is currently not practical for real-time implementation. However, it has been set up to be able to transition into a real-time package after some development. These can include:

- Running YOLO off GPU
- Embed an image message subscriber
- Object tracking