Lane Detection Package

Lane Detection Package

Description:

This page discusses the *lane_detection_package* in detail. The main purpose of this package is to perform lane detection using a polynomial fitting model and various image processing techniques.

The resource for this detection algorithm can be found here:

Moataz Elmasry. Computer Vision for Lane Finding. en. Apr. 2018. url: https : //towardsdatascience.com/computer-vision-for-lane-finding-24ea77f25209
(visited on 07/05/2023).

Note: that this algorithm is NOT completed in C++ but is completed in the `/dev` folder in python code. Progress has been made in C++ up to performing an IPM on a binary image (so maybe about ~50% of the entire process).

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:
`sliding_box_ld.yaml` is shown below:

src > lane_detection_package > config > ! sliding_box_ld.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/lane_detection_package/data/images/frame1265.jpg"
    write_output: false
    output_file: "/home/tahnt/T3_Repos/post_process_packages/ros2_ws/src/yolo_package/output/images/test_image.jpg"
    classes_file: "/home/tahnt/T3_Repos/post_process_packages/ros2_ws/src/yolo_toolbox/classes/coco.names"
    model_configuration: "/home/tahnt/T3_Repos/post_process_packages/ros2_ws/src/yolo_toolbox/cfg/yolov3.cfg"
    model_weights: "/home/tahnt/T3_Repos/post_process_packages/ros2_ws/src/yolo_toolbox/weights/yolov3.weights"
    camera_matrix_vector: [2.8669527693619584e+03, 0., 6.4000000000000000e+02, 0., 2.8669527693619584e+03, 3.6000000000000000e+02, 0., 0., 1.]
    dist_coeffs: [ -1.6652509493273235e-01, 9.7738779215728377e-01, 0., 0., 0. ]
```

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.

Launch File:

`sliding_box_ld_launch.py` is shown below:

src > lane_detection_package > launch > sliding_box_ld_launch.py > ...

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

def generate_launch_description():

    config_path = "/home/tahnt/T3_Repos/post_process_packages/ros2_ws/src/lane_detection_package/config/sliding_box_ld.yaml"

    return LaunchDescription([
        Node(
            package="lane_detection_package",
            executable="sliding_box_ld",
            name="sliding_box_lane_detection",
            output="screen",
            emulate_tty=True,
            parameters=[config_path]
        ),
    ])
```

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

Using the Package

ROS2 package

To use the (currently uncompleted) ROS2 package, follow the following steps (also included in the `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 lane_detection_package sliding_box_ld_launch.py`

Python Development

Under the `/dev` folder are two python scripts (one for image data, and one for video data - both using the same functions) that has the completed algorithm. To change image/video paths, locate the input paths set in the run() function near the bottom of the script (examples shown below)

Input image path in `sliding_box_ld.py`

```python
def run():

    #Read image
    input_path = '/home/tahnt/T3_Repos/post_process_packages/ros2_ws/src/lane_detection_package/data/images/frame1265.jpg'
    frame = readImage(input_path)
    frame_cp = frame.copy()
```

Input video path in `sliding_box_ld_vid.py`

```python
    #Read video
    vid_path = '/home/tahnt/T3_Repos/post_process_packages/ros2_ws/src/lane_detection_package/data/videos/video2.mp4'
    cap = cv.VideoCapture(vid_path)
    fps = cap.get(cv.CAP_PROP_FPS)
    frame_width = int(cap.get(cv.CAP_PROP_FRAME_WIDTH))
    frame_height = int(cap.get(cv.CAP_PROP_FRAME_HEIGHT))
```

Additional Notes:

The algorithm (specifically the Inverse Perspective Mapping function) is designed for a 1280x720 image/video frame. There a couple tunable values to consider when switching to a different ROI:

- **offset:** Tunable value used when setting the "destination points"
- **tol/tolerance:** Tunable value used when setting the "source points" (determines lateral displacement from image center)
- **height_scale:** Tunable value used when setting the "source points" (determines vertical displacement image height/image bottom)

Here's how they are used in code:

In `sliding_box_ld.cpp`:

```cpp
cv::Mat SlidingBoxLaneDetection::inversePerspectiveMapping(const cv::Mat& frame)
{

    //Define source points (set same as mask points - tunable)
    int tolerance = 75; //tunable
    float height_scale = 4.5;
    cv::Point p1 = cv::Point(frame.size().width/2 - tolerance, frame.size().height/height_scale);        //Top-left point
    cv::Point p2 = cv::Point(0, frame.size().height);                                               //Bottom-left point
    cv::Point p3 = cv::Point(frame.size().width, frame.size().height);                              //Bottom-right point
    cv::Point p4 = cv::Point(frame.size().width/2 + tolerance, frame.size().height/height_scale);        //Top-right point

    //Source points
    cv::Point2f src_points[] = {p1, p2, p3, p4};

    //Define warping points
    int offset = 570; //tunable
    cv::Point pp1 = cv::Point(offset, 0);                                                     //Top-left corner
    cv::Point pp2 = cv::Point(offset, frame.size().height);                                   //Bottom-left corner
    cv::Point pp3 = cv::Point(frame.size().width - offset, frame.size().height);              //Bottom-right corner
    cv::Point pp4 = cv::Point(frame.size().width - offset, 0);                                //Top-right corner

    //Destination points
    cv::Point2f dst_points[] = {pp1, pp2, pp3, pp4};

    //Perspective Transform
    cv::Mat pers_trans = cv::getPerspectiveTransform(src_points, dst_points);

    //Warp raw image
    cv::Mat ipm_frame;
    cv::warpPerspective(frame, ipm_frame, pers_trans, cv::Size(frame.size().width, frame.size().height), cv::INTER_LINEAR);

    return ipm_frame;
}
```

In `sliding_box_ld.py`:

```python
#----- Inverse Perspective Mapping -----
def invPersTrans(masked_binary, frame):

    frame_size = frame.shape[::-1][1:] #width by height
    offset = 570 #tunable
    tol = 75 #tunable
    height_scale = 4.5 #tunable
    src_points = np.float32([
        (frame_size[0]/2 - tol, frame_size[1]/height_scale),     # Top-left corner
        (0, frame_size[1]),                                      # Bottom-left corner
        (frame_size[0], frame_size[1]),                          # Bottom-right corner
        (frame_size[0]/2 + tol, frame_size[1]/height_scale)      # Top-right corner
    ])

    dst_points = np.float32([
        [offset, 0],
        [offset, frame_size[1]],
        [frame_size[0]-offset, frame_size[1]],
        [frame_size[0]-offset, 0]
    ])

    trans_mat = cv.getPerspectiveTransform(src_points, dst_points)
    inv_trans_mat = cv.getPerspectiveTransform(dst_points, src_points)

    warped_frame = cv.warpPerspective(frame, trans_mat, frame_size, flags=cv.INTER_LINEAR)
    warped_binary = cv.warpPerspective(masked_binary, trans_mat, frame_size, flags=cv.INTER_LINEAR)

    #Display warped raw as a check
    cv.imshow("IPM Frame", warped_frame)
    cv.waitKey(0)

    return warped_binary, inv_trans_mat
```

Note: Similar values are used when setting the mask. These values DO NOT have to match but they can.

For the IPM, it's really a matter of playing with the values to capture the desired "bird's-eye view" perspective to replicate parallel lane lines (shown below):