3EY4 Bonus Lab Report

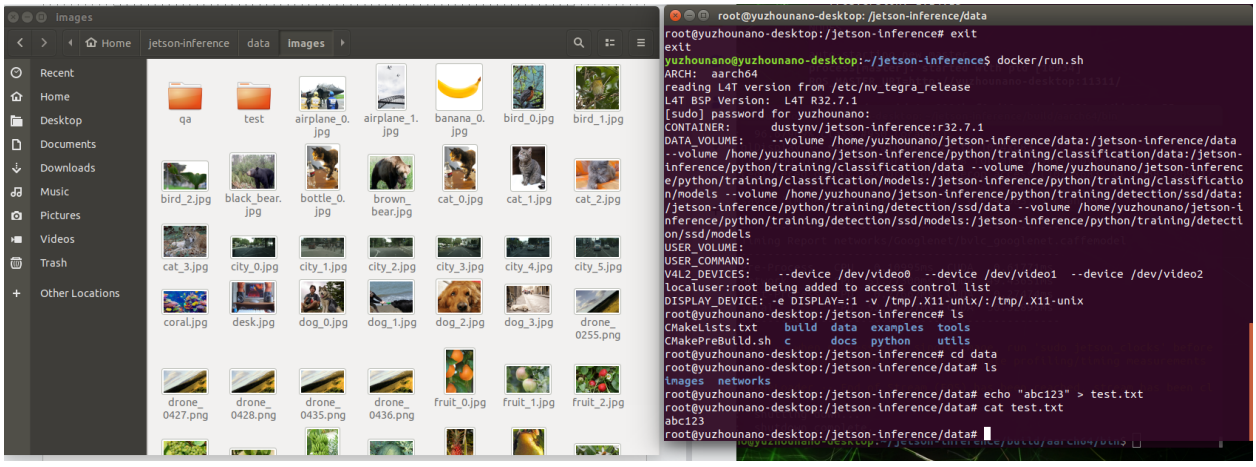
Zhaobo Wang 400188525
Xiaodi Xu 400239739
Yuzhou Jiang 400312539

Abstract
Throughout the whole 3EY4 course, we have been utilizing the lidar sensor to build the algorithm of our McMaster EV to achieve autonomous driving. But We haven't integrated the camera into our system yet. So in this bonus activity, we want to step further to find a way to utilize the camera into our AEV. Our objective is that the vehicle will respond to the detection of a person.

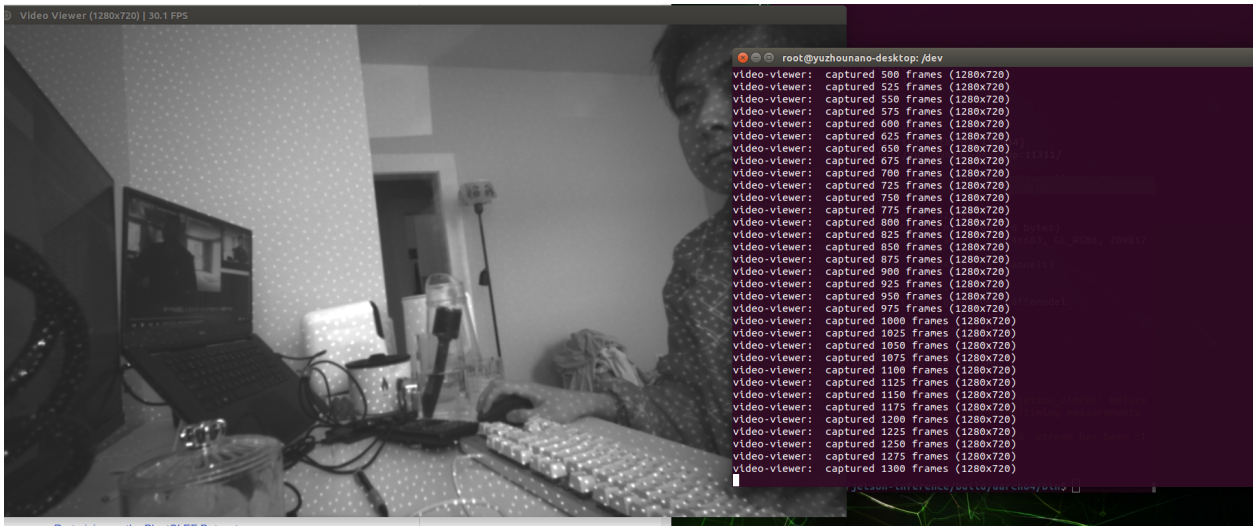
Problem Statement
The Group-AV#21 will try to develop a way to utilize the camera that when it detects a person, it will start to move and if no person is detected. Then the EV should not move.

Solution Approach
The Jetson Nano DNN Inference Library was used to detect the item in front of the EV. Jetson-inference. We modified the navigation.py file in Lab 9 so that we can achieve the new function based on what we have achieved in Lab 9. The most basic principle is the publish and subscribe in ROS. (The workflow diagram is pasted below). We created a topic called detect_person/confidence for detect_person node to publish confidence. We only want this EV only respond to a person, so we only keep the confidence level for the person and then use it to set the velocity. The process of detection is based on the DNN Inference Library.

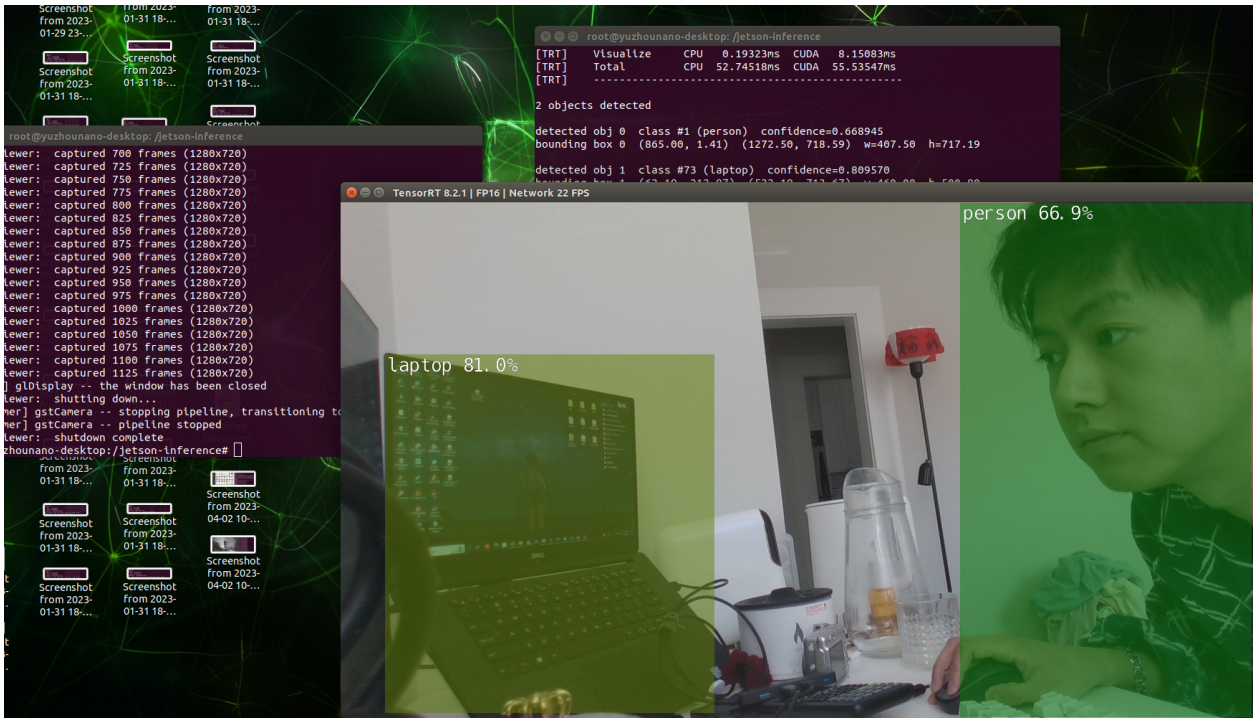
Run docker
$ git clone --recursive https://github.com/dusty-nv/jetson-inference
$ cd jetson-inference
$ docker/run.sh



$ video-viewer /dev/video1

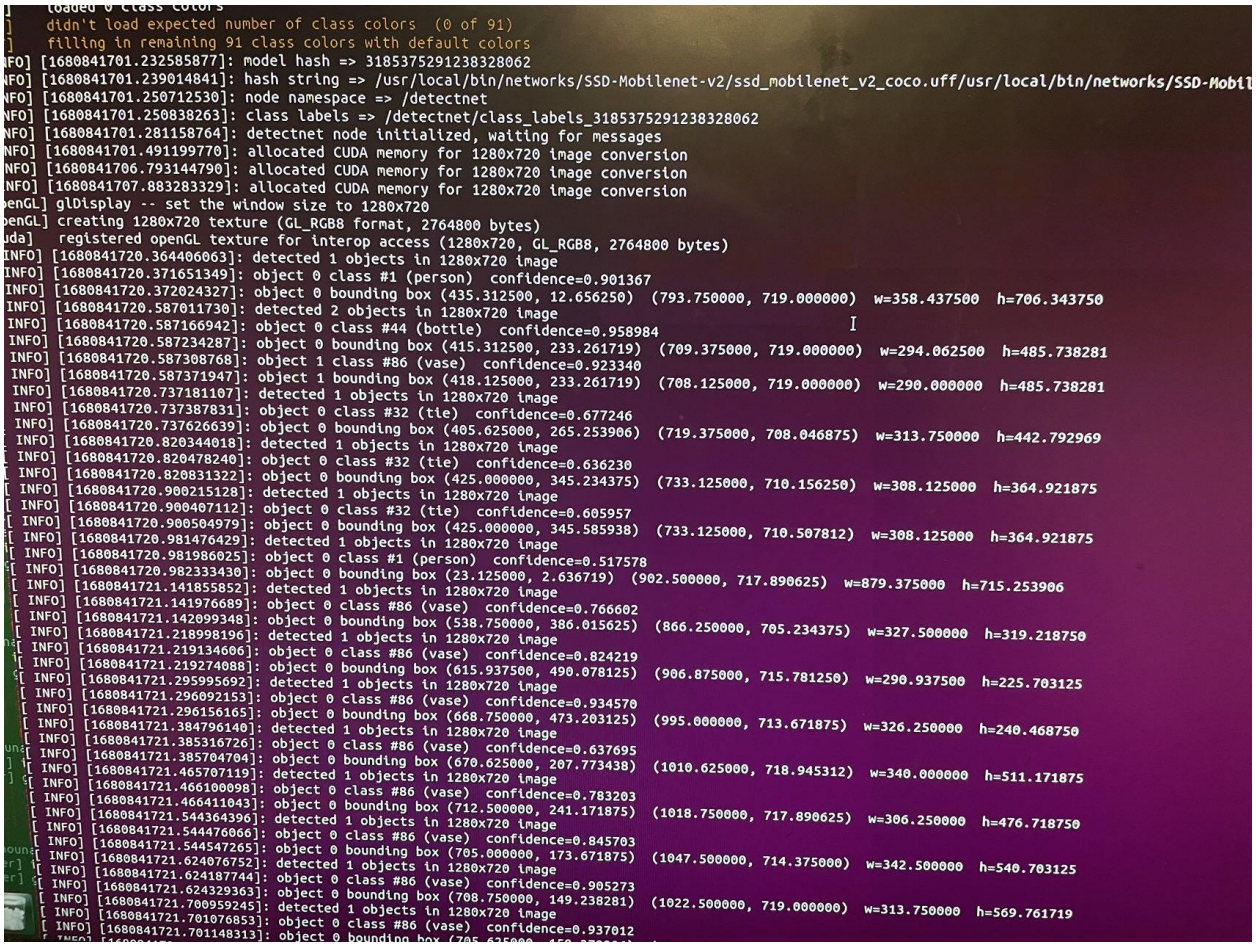


$ detectnet /dev/video2



$roscore
$cd catkin_ws/src/ros_deep_learning

$roslaunch ros_deep_learning detectnet.ros1.launch input:=v4l2:///dev/video2 output:=display://0



Class IDs / bounding boxes/ confidences

Below are the message topics and parameters that each node implements.

imagenet Node

Topic Name	I/O	Message Type	Description
image_in	Input	`sensor_msgs/Image`	Raw input image
classification	Output	`vision_msgs/Classification2D`	Classification results (class ID + confidence)
vision_info	Output	`vision_msgs/VisionInfo`	Vision metadata (class labels parameter list name)
overlay	Output	`sensor_msgs/Image`	Input image overlayed with the classification results

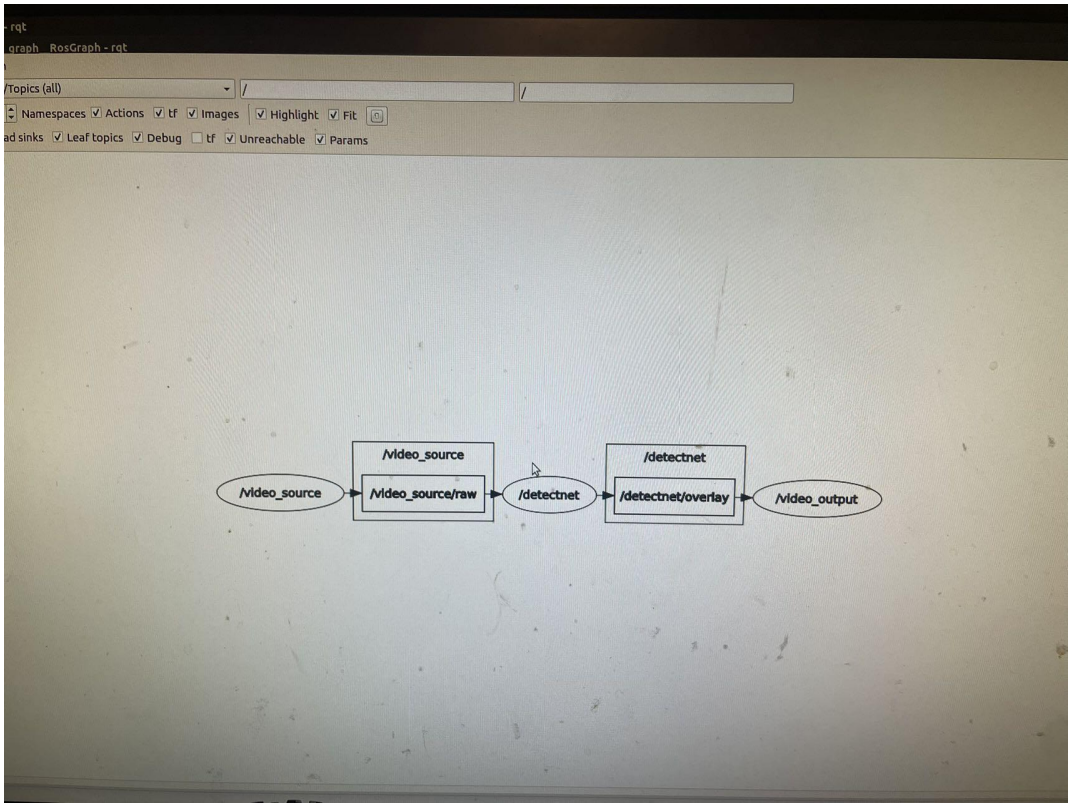
Parameter Name	Type	Default	Description
model_name	`string`	`"googlenet"`	Built-in model name (see here for valid values)
model_path	`string`	`""`	Path to custom caffe or ONNX model
prototxt_path	`string`	`""`	Path to custom caffe prototxt file
input_blob	`string`	`"data"`	Name of DNN input layer
output_blob	`string`	`"prob"`	Name of DNN output layer
class_labels_path	`string`	`""`	Path to custom class labels file
class_labels_HASH	`vector<string>`	class names	List of class labels, where HASH is model-specific (actual name of parameter is found via the `vision_info` topic)

detectnet Node

Topic Name	I/O	Message Type	Description
image_in	Input	`sensor_msgs/Image`	Raw input image
detections	Output	`vision_msgs/Detection2DArray`	Detection results (bounding boxes, class IDs, confidences)
vision_info	Output	`vision_msgs/VisionInfo`	Vision metadata (class labels parameter list name)
overlay	Output	`sensor_msgs/Image`	Input image overlayed with the detection results

Parameter Name	Type	Default	Description
model_name	`string`	`"ssd-mobilenet-v2"`	Built-in model name (see here for valid values)
model_path	`string`	`""`	Path to custom caffe or ONNX model
prototxt_path	`string`	`""`	Path to custom caffe prototxt file
input_blob	`string`	`"data"`	Name of DNN input layer
output_cvg	`string`	`"coverage"`	Name of DNN output layer (coverage/scores)
output_bbox	`string`	`"bboxes"`	Name of DNN output layer (bounding boxes)
class_labels_path	`string`	`""`	Path to custom class labels file
class_labels_HASH	`vector<string>`	class names	List of class labels, where HASH is model-specific (actual name of parameter is found via the `vision_info` topic)
overlay_flags	`string`	`"box,labels,conf"`	Flags used to generate the overlay (some combination of `none,box,labels,conf`)
mean_pixel_value	`float`	0.0	Mean pixel subtraction value to be applied to input (normally 0)
threshold	`float`	0.5	Minimum confidence value for positive detections (0.0 - 1.0)

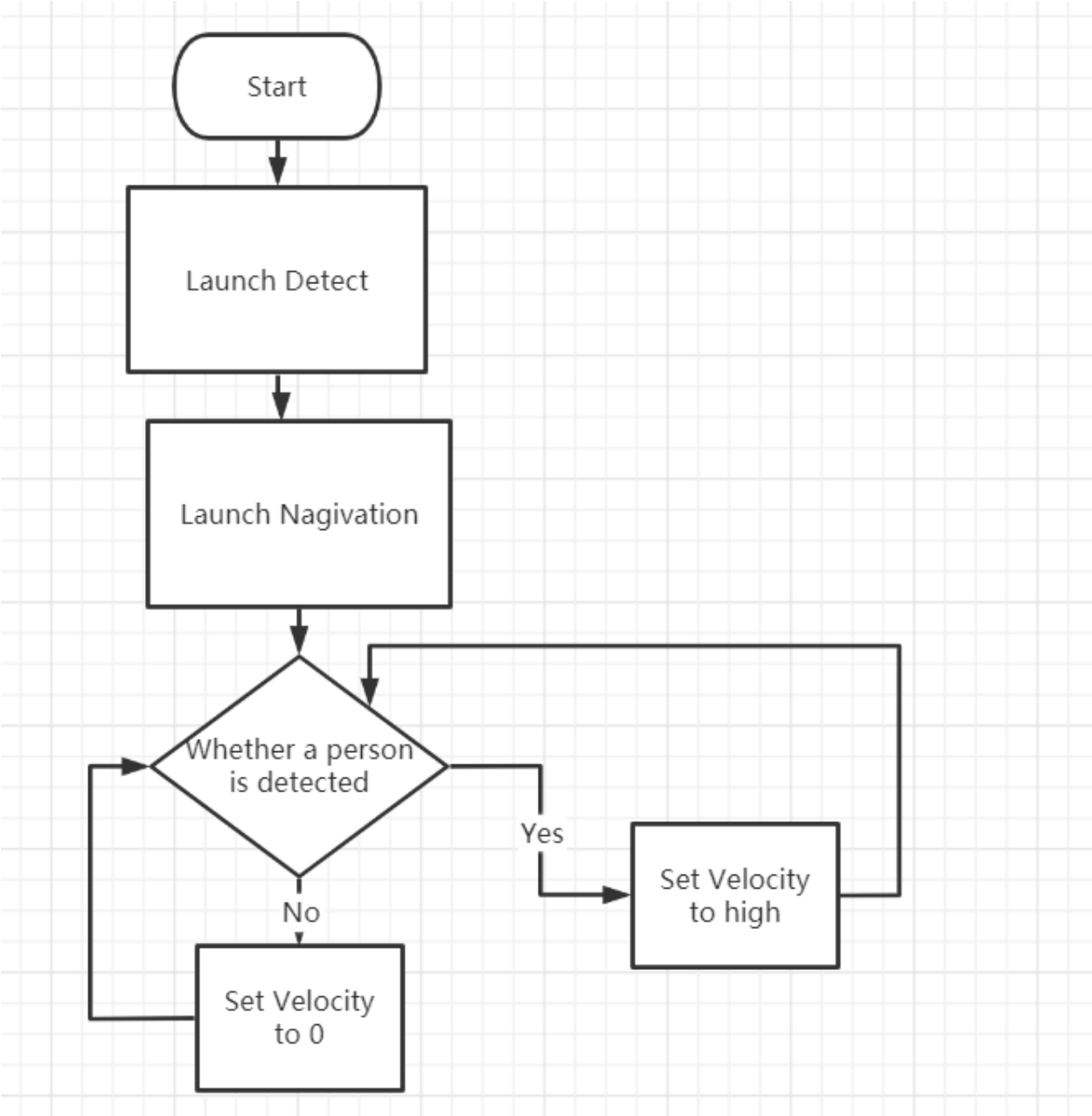
$ rosrun rqt_graph rqt_graph



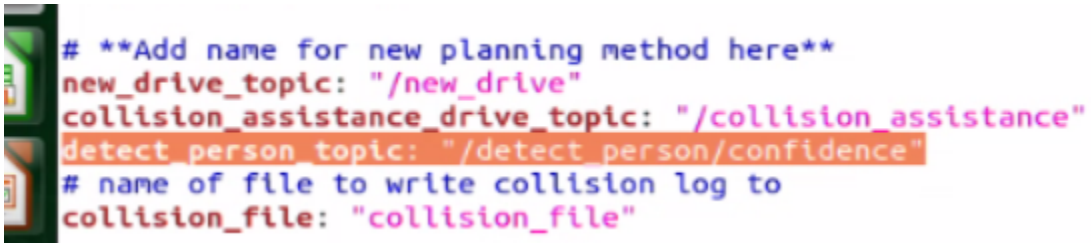
Algorithm: detect person function

```
camera = jetson.utils.videoSource("v4l2:///dev/video2")

net = jetson.inference.detectNet("ssd-mobilenet-v2", threshold=0.5)

def detect_person():
    net = jetson.inference.detectNet("ssd-mobilenet-v2", threshold=0.5)
    Flag=0
    person_confidences=[]

    while(Flag==0):
        img = camera.Capture()
        detections = net.Detect(img, overlay="box,labels,conf")
        for detection in detections:
            class_id = net.GetClassDesc(detection.ClassID)
            confidence = detection.Confidence
            if class_id=="person":
                person_confidences.append(confidence)
            print(person_confidences)

            for confidence in person_confidences:
                if confidence >= 0.7:
                    print("person is detected, start to initiate")
                    Flag=1
                    break
                else:
                    Flag=0
                    print("person is not detected!!!!!!!!!!")

                return Flag
```

```
    if angle_deg>=0 and angle_deg<=self.angle_threshold_low:
        velocity=self.velocity_high

    elif angle_deg > self.angle_threshold_low and angle_deg <= self.angle_threshold_high:
        velocity=self.velocity_medium

    else:
       velocity=self.low

    Flag = detect_person()
    if  Flag==1:
            velocity=self.velocity_high
    # Publish to driver topic
    drive_msg = AckermannDriveStamped()
    drive_msg.header.stamp = rospy.Time.now()
    drive_msg.header.frame_id = "base_link"
    drive_msg.drive.steering_angle = delta_d
    drive_msg.drive.speed = velocity
    self.drive_pub.publish(drive_msg)
```



Add new ros topic:



```
# **Add name for new planning method here**
new_drive_topic: "/new_drive"
collision_assistance_drive_topic: "/collision_assistance"
detect_person_topic: "/detect_person/confidence"
# name of file to write collision log to
collision_file: "collision_file"
```

```
 */
ROS_CREATE_PUBLISHER(vision_msgs::Detection2DArray, "detections", 25, detection_pub);
ROS_CREATE_PUBLISHER(sensor_msgs::Image, "overlay", 2, overlay_pub);

ROS_CREATE_PUBLISHER_STATUS(vision_msgs::VisionInfo, "vision_info", 1, info_callback, info_pub);

ROS_CREATE_PUBLISHER(std_msgs::Int8, "ID", 25,  ID_pub);
```

```
// populate timestamp in header field
msg.header.stamp = ROS_TIME_NOW();

// publish the detection message
detection_pub->publish(msg);
ID_pub->publish(Myid);
```

Results

As we showed in the demonstration and also insimulation, when we launch the launch file and turn on the navigation on joystick, if the camera did not detect a person (we use an item to block the camera), the EV did not move. When we move the item and the camera detected a person, the EV will start to move. If we block the camera again, the EV will stop. This proves that the data captured by the camera is utilized in our EV. But, at the same time, we have a big trouble, the original navigation did not work. We have been trying to debug this issue, but given such as the short amount of time. We haven't been able to solve this issue before our demonstration. But as we discussed with TA after the demonstration, they think we are so closed to the final step.The basic ideas and methods were on the right track, but the miscellaneous error in our detect function may lead to this issue. For example, if the confidence level does not meet the threshold, the for loop will keep running until the level meets the threshold. We should add something between each iteration of the for loop. This is the possible solution to this issue.

Terminal Run Results:

```
[0.62646484375]
person is not detected!!!11!!!!!!!11111
```

Conclusions

In conclusion, this bonus activity allowed us to explore the integration of a camera into our McMaster EV's autonomous driving system. By demonstrating that the EV could respond to the detection of a person through the use of the camera, we were able to prove that the captured data can be utilized in our EV's navigation system and create a new function that utilized the publish and subscribe principle in ROS. We created a topic called detect_person/confidence for the detect_person node to publish confidence, which allowed us to selectively respond to only a person in front of the EV. Nevertheless, we faced challenges when the original navigation system malfunctioned, posing difficulties for our team. However, our interaction with the TA following the demonstration provided constructive criticism that revealed we were almost at the final stage. Our fundamental approaches and techniques were on track, but we discovered a minor flaw in our detect function that had the potential to cause complications. Although we were unable to fully resolve the problem due to time constraints, this activity has provided valuable insights into the potential applications and benefits of integrating a camera into our EV's autonomous driving system.The workflow diagram below illustrates the process of detection, which relied on the DNN Inference Library. This successful integration of advanced technology has provided valuable insights into the potential applications and benefits of incorporating sensors such as cameras and DNN into autonomous driving systems to enhance their safety and performance.