Systematic
December report

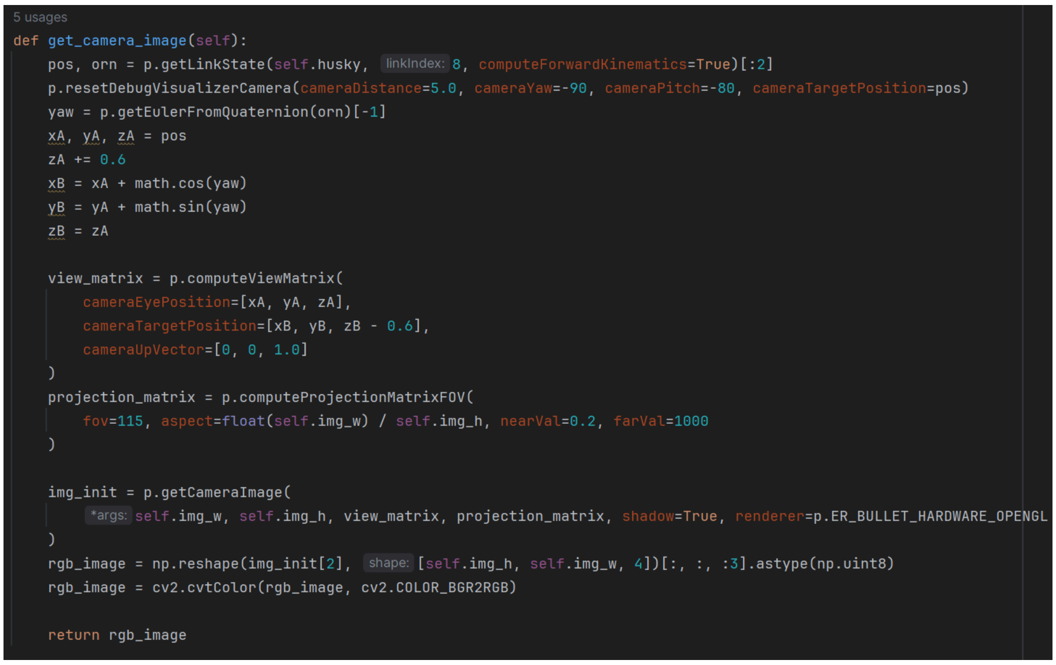
1. Simulated car

Before we got the car the first thing that we did was a simulation. We used Python for our programming language and Pybullet for our simulator. We used as the base track the Track_Test.svg from the GitHub repo and the Huskey car from the Pybullet assets as our car. This type of car has a differential driving system but we chose this one because we couldn't find a car with Ackermann steering in URDF format and we didn't want to spend more time creating an URDF car for our simulation instead focusing on the detection of the lane and training our YOLOv8 segmentation model.

As we said, we chose for the lane detection a YOLOv8 segmentation model. We did this because this model could be converted to a Hailo model that we could use in pair with the Hailo AI kit for the Raspberry Pi. As of the moment of writing this, YOLOV8 is the latest YOLO segmentation model supported by Hailo, but we can train all sorts of segmentation models that could theoretically run only on the Raspberry Pi without needing to add any additional hardware (but at a lower framerate), like the new YOLOv11.

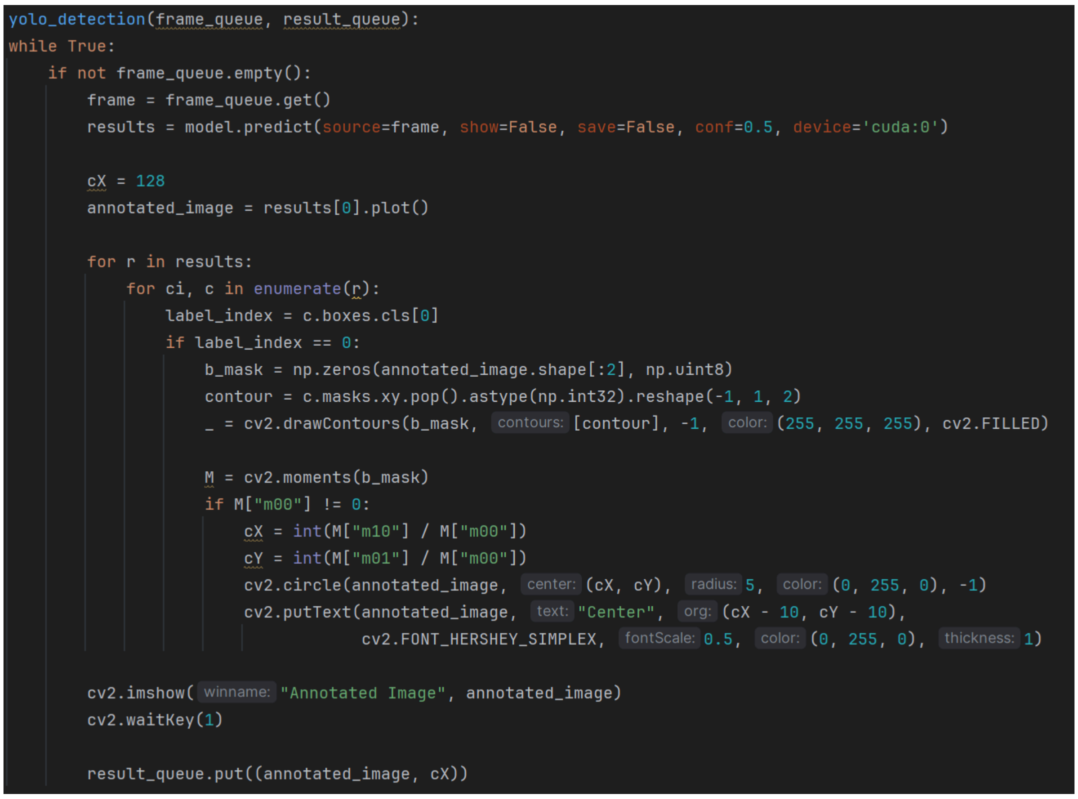
After training a test model that could segment only 3 “objects” (the road, the crosswalk, and the parking spots), we decided to also add self driving capabilities to the simulated car.

But before that, we needed to add a virtual camera to our car and add some configurations (see the code below):



```
5 usages
def get_camera_image(self):
    pos, orn = p.getLinkState(self.husky, linkIndex: 8, computeForwardKinematics=True)[:2]
    p.resetDebugVisualizerCamera(cameraDistance=5.0, cameraYaw=-90, cameraPitch=-80, cameraTargetPosition=pos)
    yaw = p.getEulerFromQuaternion(orn)[-1]
    xA, yA, zA = pos
    zA += 0.6
    xB = xA + math.cos(yaw)
    yB = yA + math.sin(yaw)
    zB = zA

    view_matrix = p.computeViewMatrix(
        cameraEyePosition=[xA, yA, zA],
        cameraTargetPosition=[xB, yB, zB - 0.6],
        cameraUpVector=[0, 0, 1.0]
    )
    projection_matrix = p.computeProjectionMatrixFOV(
        fov=115, aspect=float(self.img_w) / self.img_h, nearVal=0.2, farVal=1000
    )

    img_init = p.getCameraImage(
        *args: self.img_w, self.img_h, view_matrix, projection_matrix, shadow=True, renderer=p.ER_BULLET_HARDWARE_OPENGL
    )
    rgb_image = np.reshape(img_init[2], shape: [self.img_h, self.img_w, 4])[:, :, :3].astype(np.uint8)
    rgb_image = cv2.cvtColor(rgb_image, cv2.COLOR_BGR2RGB)

    return rgb_image
```

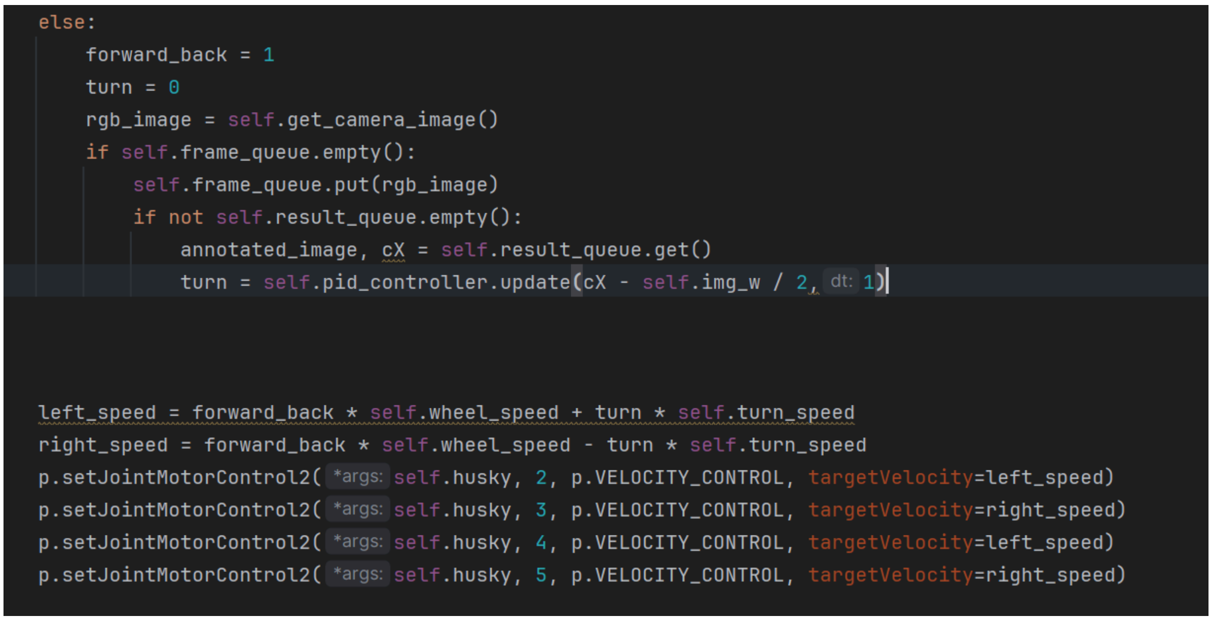
Then after this, we could perform the inference on the car's POV.



```
yolo_detection(frame_queue, result_queue):
while True:
    if not frame_queue.empty():
        frame = frame_queue.get()
        results = model.predict(source=frame, show=False, save=False, conf=0.5, device='cuda:0')

        cX = 128
        annotated_image = results[0].plot()

        for r in results:
            for ci, c in enumerate(r):
                label_index = c.boxes.cls[0]
                if label_index == 0:
                    b_mask = np.zeros(annotated_image.shape[:2], np.uint8)
                    contour = c.masks.xy.pop().astype(np.int32).reshape(-1, 1, 2)
                    _ = cv2.drawContours(b_mask, contours: [contour], -1, color: (255, 255, 255), cv2.FILLED)

                    M = cv2.moments(b_mask)
                    if M["m00"] != 0:
                        cX = int(M["m10"] / M["m00"])
                        cY = int(M["m01"] / M["m00"])
                        cv2.circle(annotated_image, center: (cX, cY), radius: 5, color: (0, 255, 0), -1)
                        cv2.putText(annotated_image, text: "Center", org: (cX - 10, cY - 10),
                                    cv2.FONT_HERSHEY_SIMPLEX, fontScale: 0.5, color: (0, 255, 0), thickness: 1)

        cv2.imshow( winname: "Annotated Image", annotated_image)
        cv2.waitKey(1)

        result_queue.put((annotated_image, cX))
```

By default if the car doesn't see any the road the “cX” which is the center of the center of the road will be 128 (our full image size / 2) and that will make the car move forward if it doesn't detect the road, but if it does the cX is the center of the road's shape.

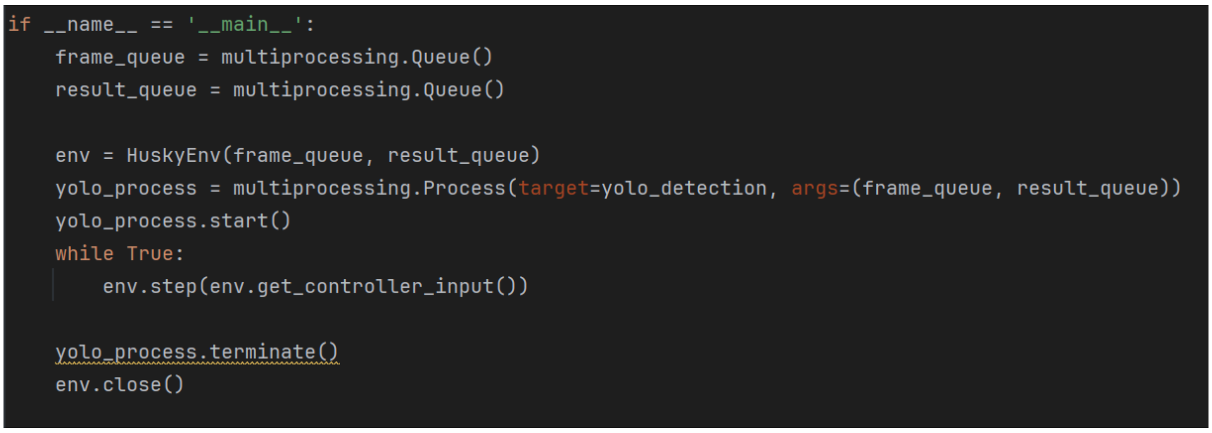
With this found center we run the value throw a PID controller, after this subtracting the obtained value from the left or the right motors depending on the case.



```
else:
    forward_back = 1
    turn = 0
    rgb_image = self.get_camera_image()
    if self.frame_queue.empty():
        self.frame_queue.put(rgb_image)
        if not self.result_queue.empty():
            annotated_image, cX = self.result_queue.get()
            turn = self.pid_controller.update(cX - self.img_w / 2, dt: 1)



left_speed = forward_back * self.wheel_speed + turn * self.turn_speed
right_speed = forward_back * self.wheel_speed - turn * self.turn_speed
p.setJointMotorControl2( *args: self.husky, 2, p.VELOCITY_CONTROL, targetVelocity=left_speed)
p.setJointMotorControl2( *args: self.husky, 3, p.VELOCITY_CONTROL, targetVelocity=right_speed)
p.setJointMotorControl2( *args: self.husky, 4, p.VELOCITY_CONTROL, targetVelocity=left_speed)
p.setJointMotorControl2( *args: self.husky, 5, p.VELOCITY_CONTROL, targetVelocity=right_speed)
```

As we mentioned this car has differential steering so we subtract the output from the PID controller from a base speed, and this differentiation of speed from the wheels makes the car turn, but on the actual car, we just need to subtract or add the output to the base angle of the steering servo. We also could find a way to also make the speed of the vehicle relative to the steering angle in some way but this will require some more testing.

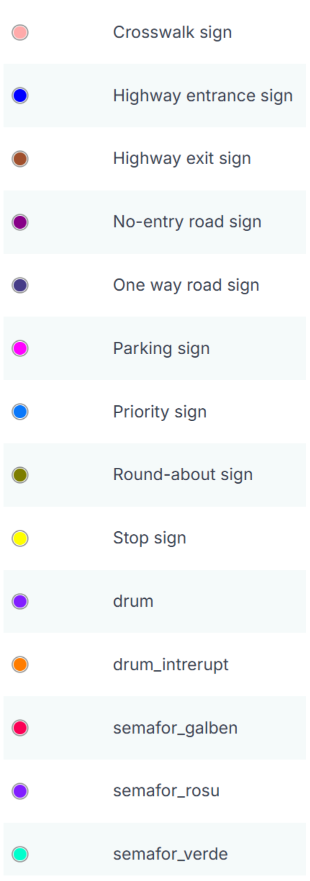
This simulation also uses multiprocessing because the simulator we used was influenced by the detection code so we decided to run the lane detection on another process therefore speeding all the execution.



```python
if __name__ == '__main__':
    frame_queue = multiprocessing.Queue()
    result_queue = multiprocessing.Queue()

    env = HuskyEnv(frame_queue, result_queue)
    yolo_process = multiprocessing.Process(target=yolo_detection, args=(frame_queue, result_queue))
    yolo_process.start()
    while True:
        env.step(env.get_controller_input())

    yolo_process.terminate()
    env.close()
```

2. New detection

After the simulation part, we started training the actual detection model with all the classes that we will need in this competition. We used Roboflow to annotate our images, and we trained the model locally. Roboflow is a great choice for annotating images due to its image augmentation function that can perform multiple transformations to our images making the final model more reliable. This is why we chose a segmentation model in the first place because we could make something that is not influenced by external factors like reflection, or motion blur, but we can consider making a separate lane detection that uses edge detection and HSV color detection as a failsafe.



●	Crosswalk sign
●	Highway entrance sign
●	Highway exit sign
●	No-entry road sign
●	One way road sign
●	Parking sign
●	Priority sign
●	Round-about sign
●	Stop sign
●	drum
●	drum_intrerupt
●	semafor_galben
●	semafor_rosu
●	semafor_verde

3. Getting started with the actual car

After receiving the kit we made the set-up and started making some test codes to became more familiar with the code structure and the hardware that we have.