Robotics

Final Project Presentation

Jacopo Raffi – Simone Marzeddu

Braitenberg vehicle navigation

Goal of the Project

Robot

A Braitenberg vehicle has to navigate a maze using its on-board camera.

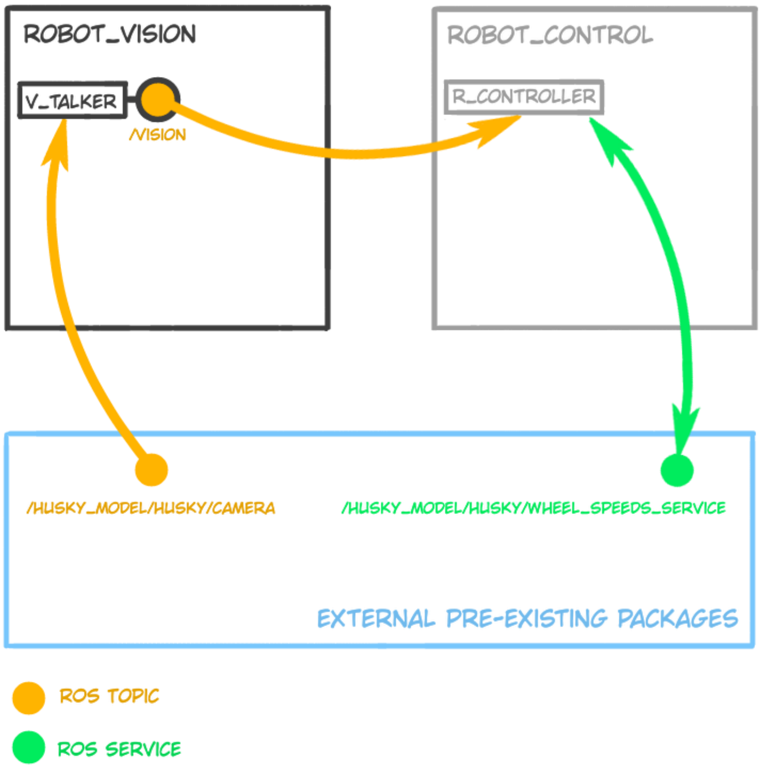
Task

The robot will move towards visual guides so to move towards the exit.

Tools

A vision model based on ML will be implemented to detect human models as visual cues

System Design


ROBOT_VISION
V_TALKER
/VISION
ROBOT_CONTROL
R_CONTROLLER
/HUSKY_MODEL/HUSKY/CAMERA
/HUSKY_MODEL/HUSKY/WHEEL_SPEEDS_SERVICE
EXTERNAL PRE-EXISTING PACKAGES
ROS TOPIC
ROS SERVICE

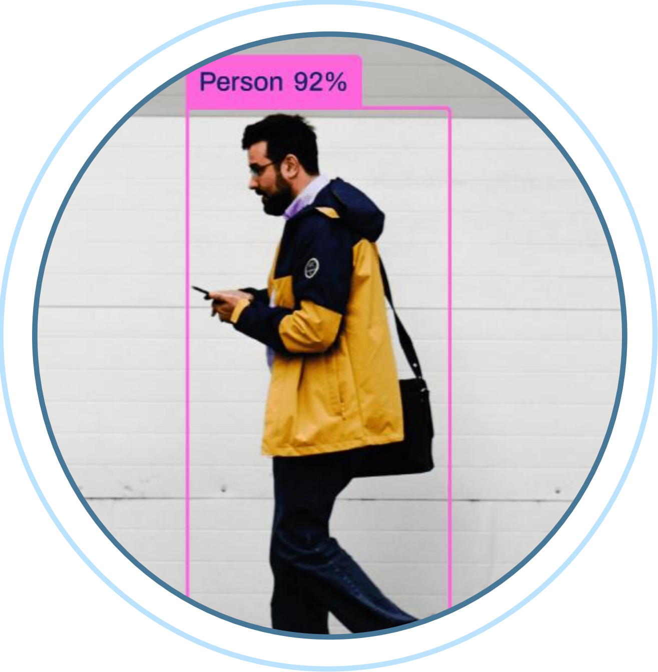
The implementation involves two packages in addition to the underlying system:

- **robot_vision**
- **robot_control**

robot_vision encapsulates support for the YOLOv8 Machine Learning model that identifies humans in the robot's field of view.

robot_control exploits this information is exploited to implement the robot's controller

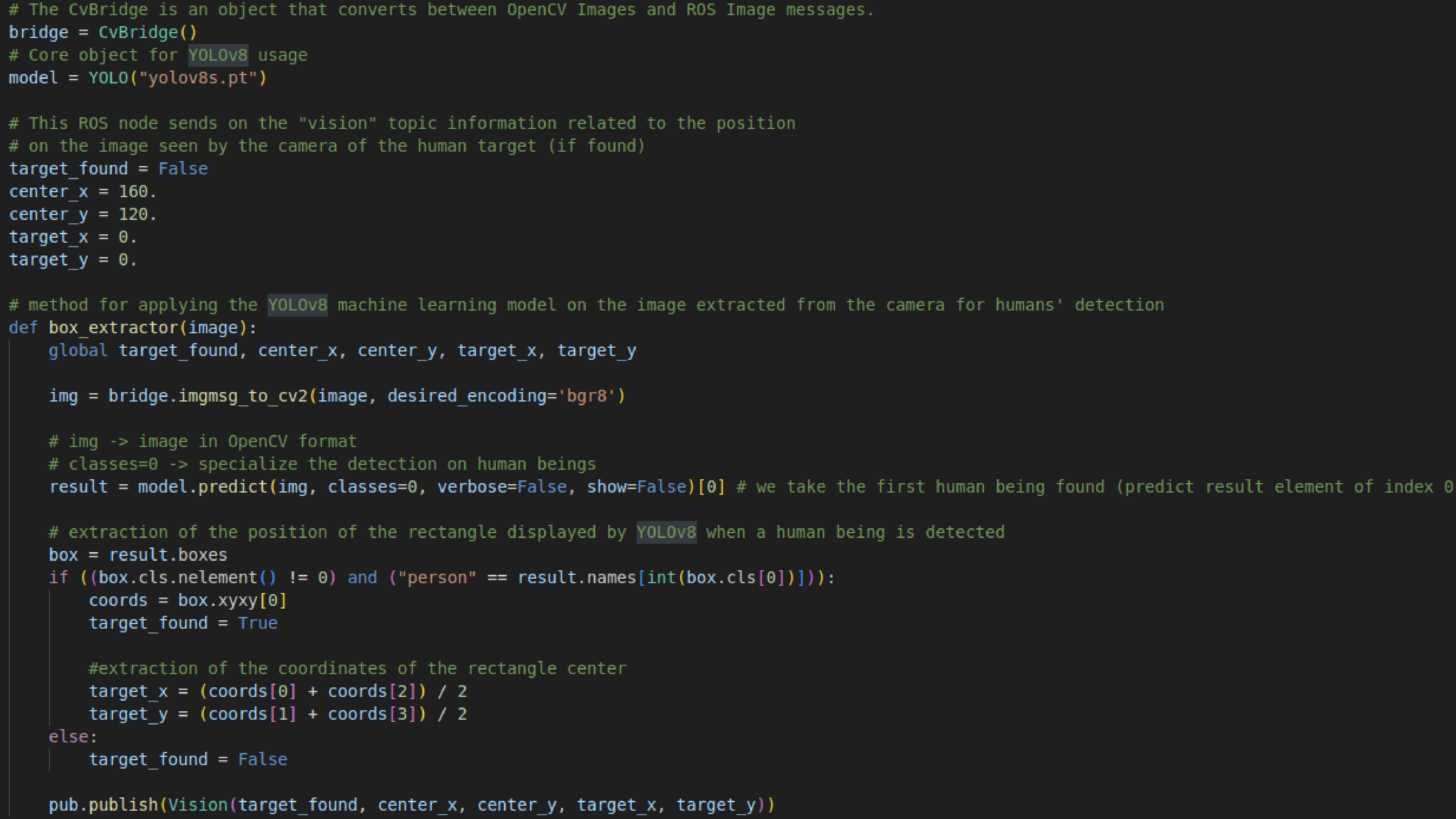
Robot Vision Package



- The **v_talker** node receives through the ROS topic "/husky_model/husky/camera" messages of type "Image" gaining access to information from the camera sensor.

- After invoking **YOLOv8**'s predict method, information on the coordinates of the centre of the image and any identified human is packaged in a message of type "Vision"

- The **v_talker** node sends through the ROS topic "/vision" messages of type "Vision" passing information to the **r_controller** node in the robot_control package.



```
# The CvBridge is an object that converts between OpenCV Images and ROS Image messages.
bridge = CvBridge()
# Core object for YOLOv8 usage
model = YOLO("yolov8s.pt")

# This ROS node sends on the "vision" topic information related to the position
# on the image seen by the camera of the human target (if found)
target_found = False
center_x = 160.
center_y = 120.
target_x = 0.
target_y = 0.

# method for applying the YOLOv8 machine learning model on the image extracted from the camera for humans' detection
def box_extractor(image):
    global target_found, center_x, center_y, target_x, target_y

    img = bridge.imgmsg_to_cv2(image, desired_encoding='bgr8')

    # img -> image in OpenCV format
    # classes=0 -> specialize the detection on human beings
    result = model.predict(img, classes=0, verbose=False, show=False)[0] # we take the first human being found (predict result element of index 0

    # extraction of the position of the rectangle displayed by YOLOv8 when a human being is detected
    box = result.boxes
    if ((box.cls.nelement() != 0) and ("person" == result.names[int(box.cls[0])])):
        coords = box.xyxy[0]
        target_found = True

        #extraction of the coordinates of the rectangle center
        target_x = (coords[0] + coords[2]) / 2
        target_y = (coords[1] + coords[3]) / 2
    else:
        target_found = False

    pub.publish(Vision(target_found, center_x, center_y, target_x, target_y))
```

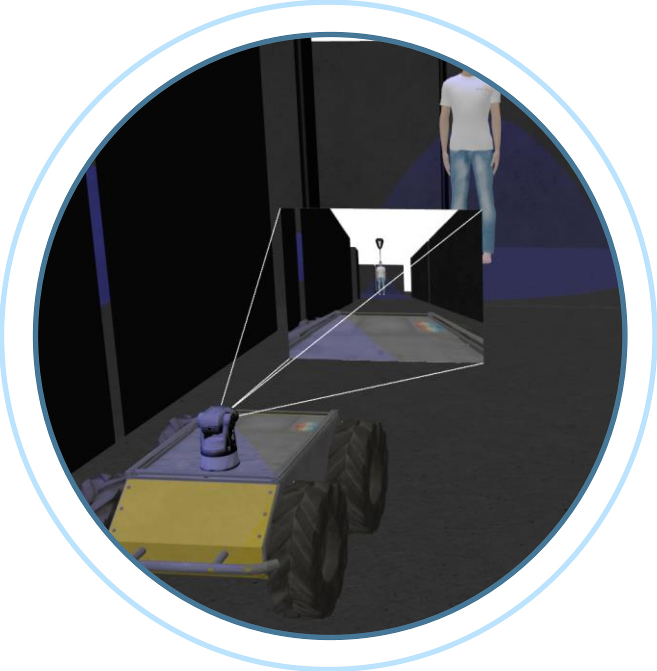
Robot Control Package

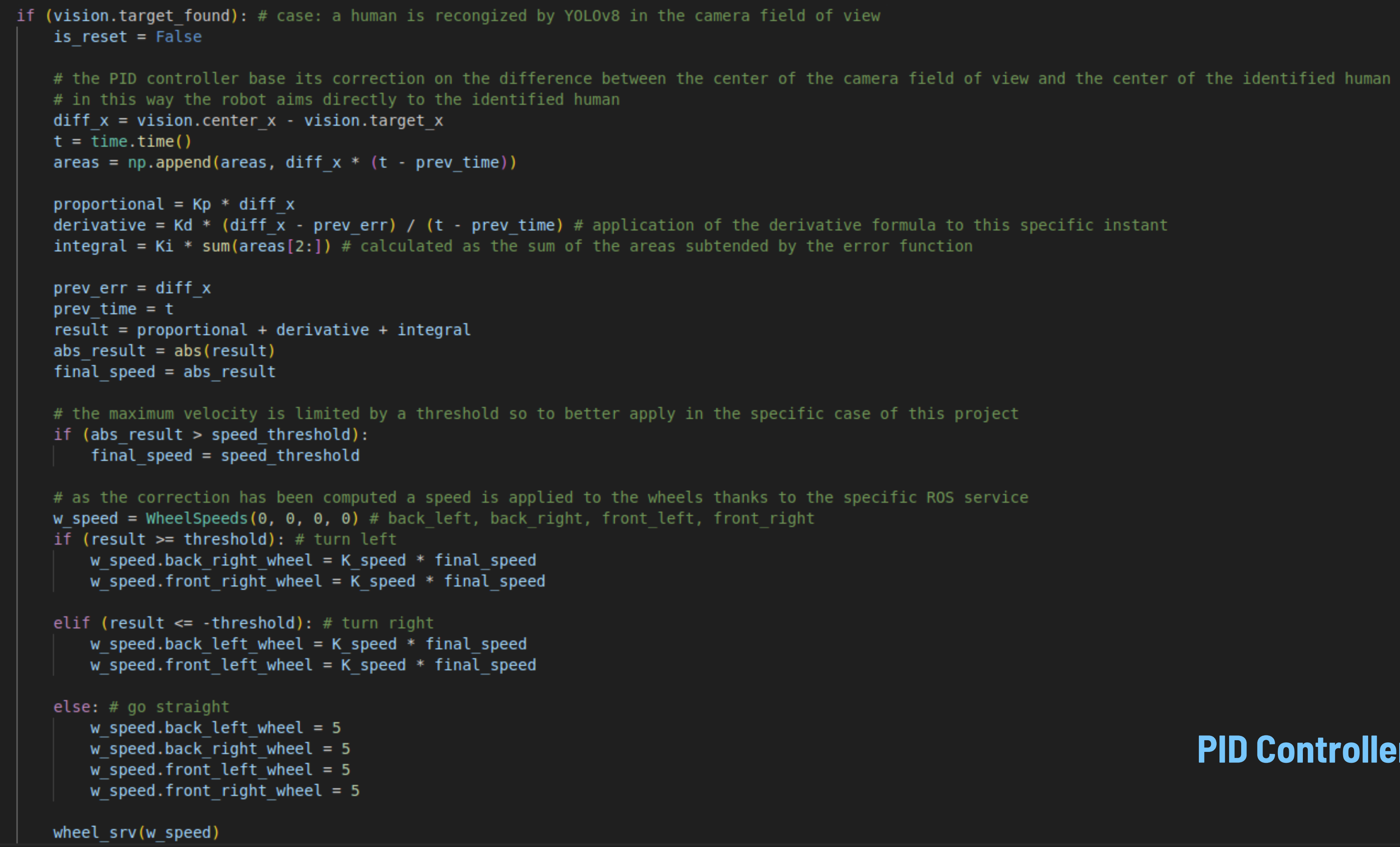
- The **r_controller** node receives through the ROS topic "/vision" information on the coordinates of the centre of the camera image and the relative position of any identified human.

- Considering the x-axis distance between the two points, a simple synchronous PID Controller considers it as an error to calibrate the cruise of the robot towards the human.

- if no human is identified by the robot, it will start a counter-clockwise rotation on the spot to identify the next human in its path

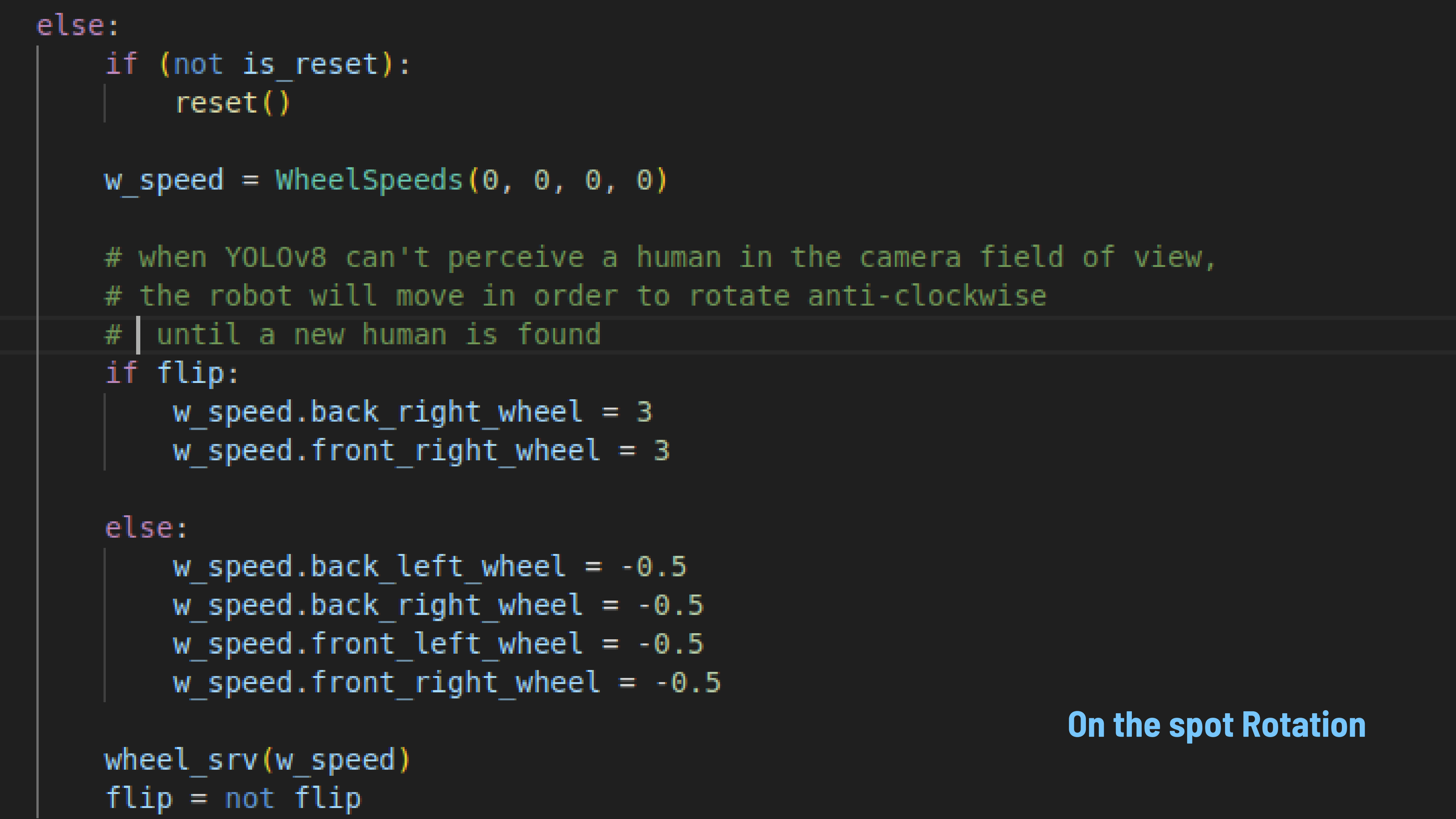
- The **r_controller** node exploits the ROS service "/husky_model/husky/wheel_speeds_service" to impose the corrected motor input to the wheels.





```
if (vision.target_found): # case: a human is recongized by YOLOv8 in the camera field of view
    is_reset = False

    # the PID controller base its correction on the difference between the center of the camera field of view and the center of the identified human
    # in this way the robot aims directly to the identified human
    diff_x = vision.center_x - vision.target_x
    t = time.time()
    areas = np.append(areas, diff_x * (t - prev_time))

    proportional = Kp * diff_x
    derivative = Kd * (diff_x - prev_err) / (t - prev_time) # application of the derivative formula to this specific instant
    integral = Ki * sum(areas[2:]) # calculated as the sum of the areas subtended by the error function

    prev_err = diff_x
    prev_time = t
    result = proportional + derivative + integral
    abs_result = abs(result)
    final_speed = abs_result

    # the maximum velocity is limited by a threshold so to better apply in the specific case of this project
    if (abs_result > speed_threshold):
        final_speed = speed_threshold

    # as the correction has been computed a speed is applied to the wheels thanks to the specific ROS service
    w_speed = WheelSpeeds(0, 0, 0, 0) # back_left, back_right, front_left, front_right
    if (result >= threshold): # turn left
        w_speed.back_right_wheel = K_speed * final_speed
        w_speed.front_right_wheel = K_speed * final_speed

    elif (result <= -threshold): # turn right
        w_speed.back_left_wheel = K_speed * final_speed
        w_speed.front_left_wheel = K_speed * final_speed

    else: # go straight
        w_speed.back_left_wheel = 5
        w_speed.back_right_wheel = 5
        w_speed.front_left_wheel = 5
        w_speed.front_right_wheel = 5

    wheel_srv(w_speed)
```

PID Controller



```python
else:
    if (not is_reset):
        reset()

    w_speed = WheelSpeeds(0, 0, 0, 0)

    # when YOLOv8 can't perceive a human in the camera field of view,
    # the robot will move in order to rotate anti-clockwise
    #  until a new human is found
    if flip:
        w_speed.back_right_wheel = 3
        w_speed.front_right_wheel = 3

    else:
        w_speed.back_left_wheel = -0.5
        w_speed.back_right_wheel = -0.5
        w_speed.front_left_wheel = -0.5
        w_speed.front_right_wheel = -0.5

    wheel_srv(w_speed)
    flip = not flip
```

On the spot Rotation

Thank you for your attention!

Jacopo Raffi – Simone Marzeddu