

Section 6: Object Detection

Overview

The goals for this section:

1. Run an image classifier on the TurtleBot.
2. Detect and classify real-world objects using the classifier.

Setup

Task 1.1 — Cleanup. Remove directory `~/autonomy_ws` if it exists. This may be leftover from a previous section

Task 1.2 – Git clone your repo. Clone your group's workspace repository into the `~/autonomy_ws/src` directory:

Shell

```
cd ~/autonomy_ws/src  
git clone <repo>
```

Task 1.3 – Verify the presence of the heading controller. We will be building upon the heading controller completed in section 4. Your group's workspace repository should already contain the `autonomy_repo` ROS package from that section. Verify that the `autonomy_repo` package is present and that it contains an updated `scripts/heading_controller.py` file. Verify the heading controller is functional by running these commands:

Shell

```
# In Terminal 1  
cd ~/autonomy_ws  
colcon build  
ros2 launch asl_tb3_sim root.launch.py
```

```
# In Terminal 2
source ~/autonomy_ws/install/setup.bash
ros2 launch autonomy_repo heading_control.launch.py
```

- **Checkpoint — Call a CA to show that your robot turns when a new goal pose is issued.**

Circular Motion

For this section, we want to create a dummy controller, `PerceptionController`, that causes the robot to move in a circular motion until a stop command is sent. We will update a copy of the heading controller script and launch file for this purpose.

Task 2.1 - Create a copy of the `heading_controller.py` file in the `scripts/` directory and call it `perception_controller.py`. Change the name of the class in this file to `PerceptionController`, and it should also be a `BaseHeadingController` child class. Name your node `perception_controller`.

Task 2.2 - In the `__init__` method of the class declare a parameter `active` with a default value of `True`. Also, use the `@property` decorator that defines the property `active` (of type `bool`) and gets the real-time active value of the controller.

NOTE: This parameter definition should be similar to the definition of `kp` in the `HeadingController` class.

Task 2.3 - Update the `compute_control_with_goal` method to return a constant angular velocity `omega` of `0.2` (you can play around with this value if you want) when `active` is `true` and `0` when it is `false`.

Task 2.4 - Update the autonomy repo `CMakeLists.txt` file to include our new script.

Task 2.5 - Create a copy of the `heading_control.launch.py` launch file in the `launch/` directory and call it `perception_controller.launch.py`. In this launch file, replace the `heading_controller.py` node with our newly created `perception_controller.py` node.

Task 2.6 - Build the `~/autonomy_ws` workspace and launch the simulator and the newly created controller. To demonstrate that the `active` parameter works, specify a goal pose in RViz and verify that your robot starts spinning after the goal pose is set. Also, verify that the robot stops spinning when the `active` parameter is set to `false` and spins when `true`.

None

Some helpful commands

```
ros2 launch <name of package> <name of launch file>
ros2 param set <node_name> <parameter_name> <value>
ros2 param list
ros2 launch asl_tb3_sim root.launch.py
```

- **Checkpoint — Call a CA to show that your robot turns when `active` is `true` and it is static otherwise.**

Object Detection

Task 3.1 - End your simulator process, get a CA to give you a robot based on your `$ROS_DOMAIN_ID`, set up the robot using the same steps covered in past sections, and demonstrate that your controller works on the real robot as well.

None

```
ssh aa274@<robot_name>.local
```

```
# Enter the password given by the CA
# Start the ROS environment
```

```
micromamba activate ros_env

# Bring up the Turtlebot
ros2 launch asl_tb3_driver bringup.launch.py
```

In Homework 3 you got some experience using a neural network to detect and classify the objects in an image. In this section, we will be doing something similar. However, rather than running the detection on a single image frame, we will run the detection task on a sequence of images being returned by the robot. The robot performs detection using an [SSDLITE320_MOBILENET_V3_LARGE](#) PyTorch model trained on the [COCO dataset](#).

Task 3.2 - Create a new terminal window, SSH into the robot and launch the detector we created for you using these commands:

```
None
ssh aa274@<robot_name>.local

# Enter the password given by the CA
# Start the ROS environment
micromamba activate ros_env

# Bring up the Turtlebot
ros2 launch asl_tb3_driver test_detector.launch.py
```

Task 3.3 - Next, display the list of active ROS2 parameters and verify that you see the **threshold** and **target_class** parameters under the **/mobilenet_detector** node.

- The **threshold** parameter indicates the minimum probability a prediction has to have for it to be considered by the detector node we wrote for you.
- The **target_class** parameter indicates the current COCO label class we are trying to detect.

Task 3.4 - What are the current values of those parameters?

None

Some helpful commands

```
ros2 param list
```

```
ros2 param get <node_name> <parameter_name>
```

Task 3.5 - Next, display the list of active ROS2 topics and verify that you see the `/detector_image` and `/detector_bool` topics.

- The `/detector_bool` topic publishes a `Boolean` message which indicates whether our detector node has identified the object in the class specified by the `target_class` parameter.



This `Boolean` message type is different from the regular Python `bool`.

- The `/detector_image` topic publishes an `Image` message that contains the image frame returned by the robot's camera overlaid with bounding boxes around objects identified by our model with a probability above that set in the `threshold` parameter.

Task 3.6 - Visualize the image returned by the `/detector_image` topic in RViz. To add the topic, you need to click on the *Add* button on the bottom left. Once you've clicked this, a popup will come up. In the popup, click the *By topic* tab and scroll down until you see the `/detector_image` topic. Under that topic click `Image`. Now you should see what the robot sees.

Task 3.7 - Update the value of our `target_class` parameter and set it to "`stop sign`". Echo the messages from the `/detector_bool` topic and verify that our classifier node can classify the object already specified in the `target_class` parameter.

- **Checkpoint — Call a CA to indicate that you have gotten to this point in the section.**

Control with Perception

Task 4.1 - In the `PerceptionController` node, create a Boolean variable `image_detected` and set its value to `False` (this is not a ROS parameter, just an attribute of the class). In that same node, create a subscriber with a callback function that listens to the `/detector_bool` topic and sets `image_detected` to `True` when an object in the `target_class` parameter class is detected.

Task 4.2 - In addition, replace the logic in the `compute_control_with_goal` method in the `PerceptionController` node such that it sets `0.2` as the angular velocity if `image_detected` is `False`. Otherwise, it sets `0` as the angular velocity.

Task 4.3 - Build and test these updates.

- **Checkpoint — Call a CA to demonstrate that your robot's circulation motion stops when the target object is detected!**

Clean Up

Clean up the workstation.

1. Push your code (`add`, `commit`, and `push`) to your `s5_inperson` branch!
2. Create a PR either from the GitHub web page or by running `gh pr create` in your terminal.
3. Verify that you've pushed correctly by checking if the code you wrote is in the repository on [GitHub.com](https://github.com)
4. Delete the `~/autonomy_ws/`.
5. Log out of GitHub on the workstation.
6. Close all terminals and windows!
7. Shutdown laptop

- **Checkpoint — Call a CA over to sign you out!**