Boot Camp Part 1: Task 3: Wall Follower

The goal of the final wall following controller is to have the robot move in a straight line parallel to the wall. The alignment of the robot with respect to the wall will be corrected using the distance sensors through a cartesian control scheme.

Add a new node to the **controller** package by adding a file

```
touch ~/dd2419_ws/src/controller/controller/wall_following_controller.py
```

and edit the file `~/dd2419_ws/src/controller/setup.py` by adding

```
'wall_following_controller = controller.wall_following_controller:main'
```

as an entry point. The file should look as follows after you have edit it

```
from setuptools import find_packages, setup

package_name = 'controller'

setup(
    name=package_name,
    version='0.0.0',
    packages=find_packages(exclude=['test']),
    data_files=[
        ('share/ament_index/resource_index/packages',
            ['resource/' + package_name]),
        ('share/' + package_name, ['package.xml']),
    ],
    install_requires=['setuptools'],
    zip_safe=True,
    maintainer='you',
    maintainer_email='you@todo.todo',
    description='TODO: Package description',
    license='MIT',
    tests_require=['pytest'],
    entry_points={
        'console_scripts': [
            'open_loop_controller = controller.open_loop_controller:main',
            'cartesian_controller = controller.cartesian_controller:main',
            'wall_following_controller = controller.wall_following_controller:main'
        ],
    },
)
```

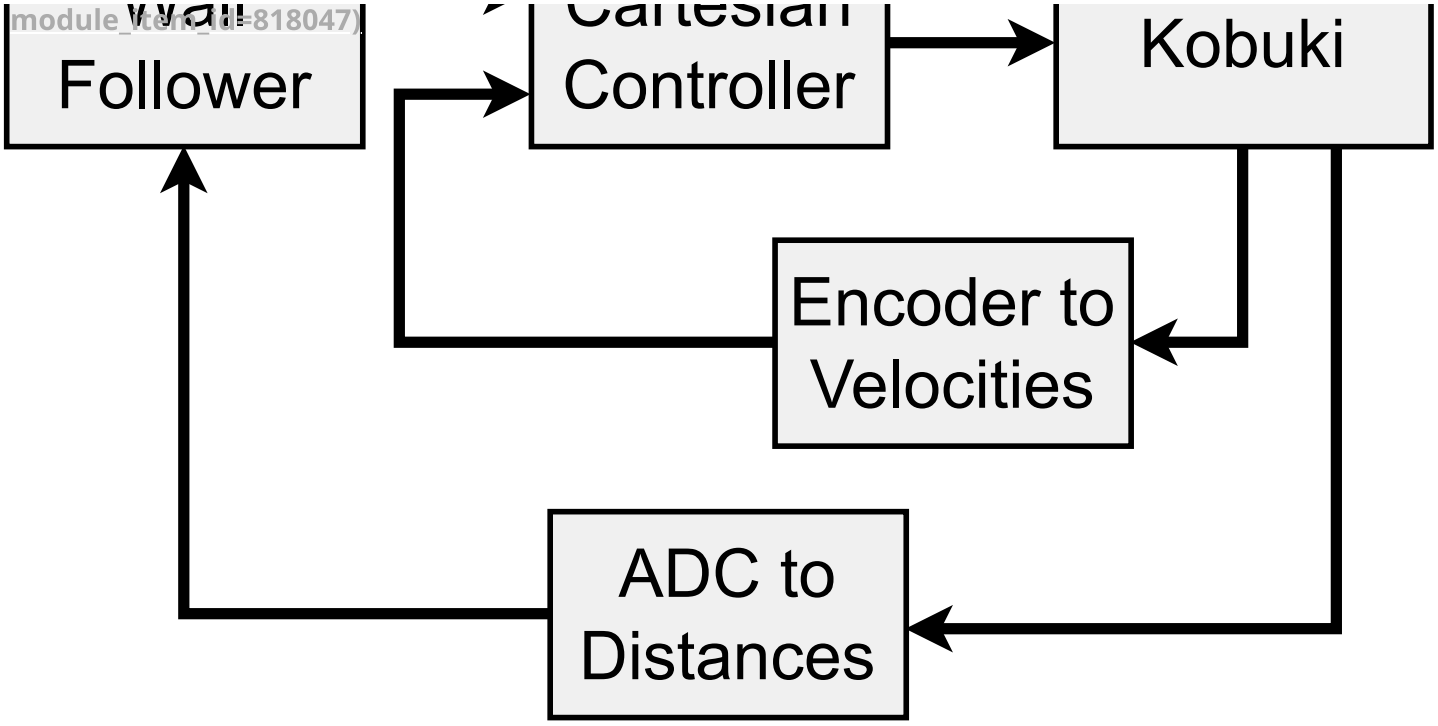
Build the workspace so the new node is installed

```
cd ~/dd2419_ws
colcon build --symlink-install
```

(`~/dd2419_ws/src/controller/controller/wall_following_controller.py`) using the following skeleton

```
#!/usr/bin/env python

import rclpy
from rclpy.node import Node

from robp_boot_camp_interfaces.msg import ADConverter
from geometry_msgs.msg import Twist


class WallFollowingController(Node):

    def __init__(self):
        super().__init__('wall_following_controller')

        # TODO: Implement

    # TODO: Implement


def main():
    rclpy.init()
    node = WallFollowingController()
    try:
        rclpy.spin(node)
    except KeyboardInterrupt:
        pass

    rclpy.shutdown()


if __name__ == '__main__':
    main()
```

The wall following controller + cartesian controller will follow this structure:



Edit the **wall_following_controller** node by editing the file `~/dd2419_ws/src/controller/controller/wall_following_controller.py`. This node should publish a **geometry_msgs/msg/Twist** type message for the robot so that it **aligns to the wall by rotating with an angular velocity that depends on the readings of the distance sensors**. The linear velocity will be set constant for simplicity. The pseudocode for a simple wall following Cartesian controller is:

```
linear_vel = < SOME CONSTANT >
angular_vel = alpha * ( distance_sensor1 - distance_sensor2 )
```

The angular velocity is a P-control that generates angular velocities **proportional to the difference of the distance sensor ADC values**. Remember to place the linear velocity in the **x-component** and the angular velocity in the **z-component** in the twist message.

Notice that for doing the controller we do not really need to convert the distance measurements to meters, if designed properly the controller can just operate on raw ADC signal values.

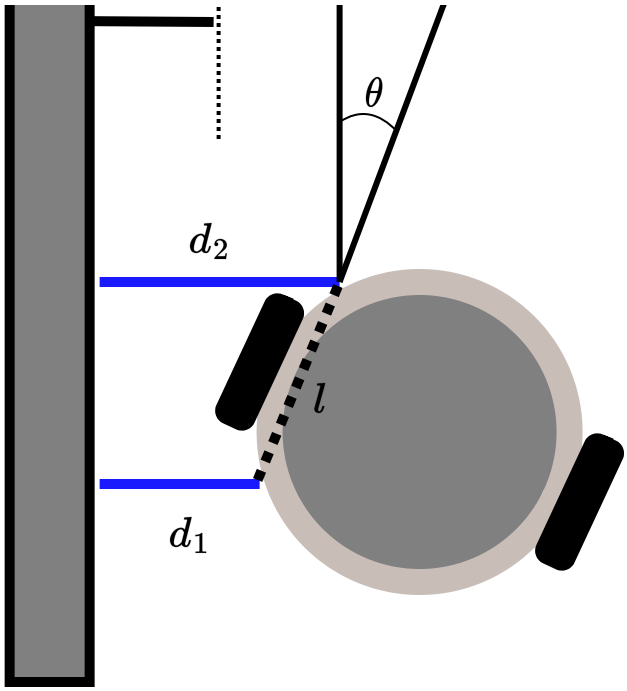
This control scheme **decouples** the control in two parts: one Cartesian control that controls the alignment of the robot with respect to the wall and the motor control which does the lower level control of the wheels' angular velocities.

There are many ways to do a wall following control, the decoupled scheme presented here is one that is easy enough to understand and code. We have not considered, e.g., a minimum distance that the robot has to keep to the wall, we simply align as we drive forward from the initial position. You are welcome to try any other control scheme if you want, the recipe presented here is enough to get a passing grade.

$$\theta = \arctan\left(\frac{d_1 - d_2}{\sqrt{(d_1 - d_2)^2 + l^2)}}\right)$$

$$l = 0.2$$

$$d_i = 1.114e^{-0.004\text{adc}}$$



Testing the Controller

For testing the wall-following controller, you can first publish to the simulation nodes (**/mobile_base/commands/velocity** topic) at **10 Hz**.

Once you are happy with the results, stop publishing to the **/mobile_base/commands/velocity** topic and publish the twist message to the **/motor_controller/twist** topic of your **Cartesian controller** that you coded in **task 2**. You can compare the performance of your Cartesian controller vs. the simulation by observing the output trajectory.

When you have implemented your wall following controller you can run it

```
ros2 run controller wall_following_controller
```

Remember to also run you Cartesian controller

```
ros2 run controller cartesian_controller
```

and also

```
ros2 launch robp_boot_camp_launch boot_camp_part1_launch.xml
```

in different terminals/tabs.

Tips

```
call /reset_world std_srvs/srv/Empty {}
```

- A P controller is enough for the wall follower errors.
- Try changing the desired angle to the wall based on the distance between the robot and the wall.
- Use radians not degrees for the kinematic calculations.