

# Boot Camp Part 1: Task 2: Cartesian Controller

## Theoretical Background: DC Motors and Differential Encoders

As we saw in the previous (optional) task it is necessary to have a sensor mounted on the motors which provides some sort of measurement of the output position/velocity of the motor to be able to control it.

In practice, DC motors are fabricated with encoders to accomplish that. There are different kinds of encoders, but the majority of them are basically digital sensors that indicate the change in the angle of the motor shaft. <http://www.ikallogic.com/closed-loop-speed-and-position-control-of-dc-motors/>

As described before, the DC motors simulated in this lab have **360 ticks per revolution** (1 revolution = 360 degrees of rotation =  $2\pi$  radians of rotation).

This means that a value of **1** on the `/motor/encoders/delta_encoder_left` topic indicates that the motor has rotated **1 degree** since the last control cycle. Each control cycle is **100 ms long** (10 Hz control), which means that a value of **1** on the differential encoder topic indicates that the wheel is spinning at (roughly, up to quantization error) **1 degree/100 ms = 10 degrees/s**

## Create a Cartesian Controller

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

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

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

```
'cartesian_controller' = controller.cartesian_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'
    ],
},
)

```

Build the workspace so the new node is installed

```

cd ~/dd2419_ws
colcon build --symlink-install

```

Implement the **cartesian\_controller**

(~/dd2419\_ws/src/controller/controller/cartesian\_controller.py) using the following skeleton

```

#!/usr/bin/env python

import rclpy
from rclpy.node import Node

from robp_interfaces.msg import DutyCycles, Encoders
from geometry_msgs.msg import Twist

class CartesianController(Node):
    def __init__(self):
        super().__init__('cartesian_controller')

        # TODO: Implement

        # TODO: Implement

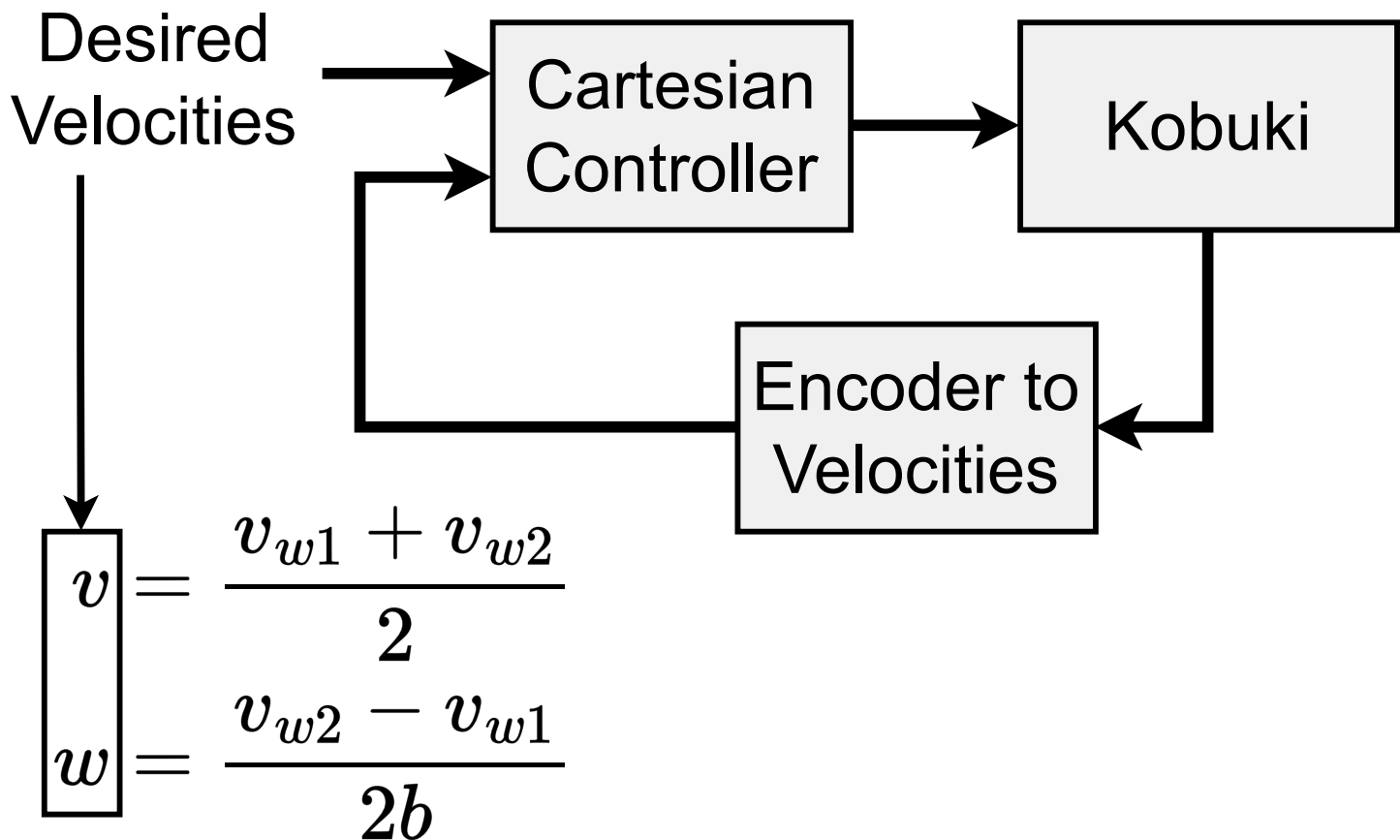
def main():
    rclpy.init()
    node = CartesianController()
    try:
        rclpy.spin(node)
    except KeyboardInterrupt:
        pass

    rclpy.shutdown()

if __name__ == '__main__':
    main()

```

This controller takes as input the **encoder feedback** from the motors and produces **duty cycle** signals for controlling the motor speeds. Below is an illustration of the high-level view of the controller:



The motor controller takes as input a **twist** (**linear and angular velocity**;  $v$  and  $w$  respectively) of the robot and controls the power of the motors through the **duty cycle** signal such that each wheel spins at the **desired angular velocity**. To estimate the **error** between the **desired angular velocity** for each wheel and the **actual angular velocity** the controller must use **feedback from the motor encoders**. Your controller should run at **10 Hz** (every 100 ms).

For this task, it is enough for you to implement a **P-controller** (**proportional controller**) for each of the motors. A PI-controller (proportional and integral) should work much better.

## ROS Interface

Your **cartesian\_controller** node should **subscribe** to the following topics:

- **/motor\_controller/twist** (*message type: geometry\_msgs/msg/Twist*): in this topic, the controller will receive the linear and angular velocity at which we wish to move the robot (expressed in the base frame of the robot). The message is a **6D twist** (3D for linear velocity and 3D for angular velocity) but since we are controlling the robot in a 2D plane you only need to use one component of the linear velocity (**the x-component**) and one component of the angular velocity (**the z-component**). Using the kinematics equations for differential mobile robot configurations, one can

then calculate the individual contributions of each wheel (in terms of angular velocity) to achieve the desired twist. To view the complete twist message definition run in a terminal:

```
ros2 interface show geometry_msgs/msg/Twist
```

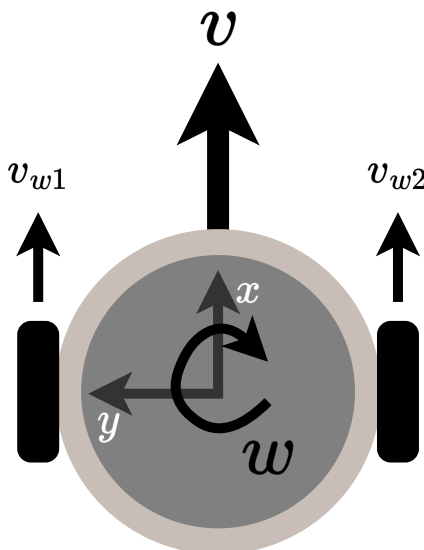
- **/motor/encoders** (message type: *robp\_interfaces/msg/Encoders*): through this topic, the controller will receive the encoder feedback signals for estimating the angular velocity of the wheels.

Your node should **publish** to the following topic:

- **/motor/duty\_cycles** (message type: *robp\_interfaces/msg/DutyCycles*): duty cycle signal for controlling the power fed to the motors.

For more details on the **encoder** and **duty cycle** topics refer back to the [Kobuki description page \(https://canvas.kth.se/courses/45204/pages/boot-camp-part-1-the-kobuki-robot-setup\)](https://canvas.kth.se/courses/45204/pages/boot-camp-part-1-the-kobuki-robot-setup).

## Calculating the Wheel Angular Velocities



$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \end{bmatrix} = \begin{bmatrix} \cos \theta & 0 \\ \sin \theta & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v \\ w \end{bmatrix}$$

$$v = \frac{v_{w1} + v_{w2}}{2}$$

$$w = \frac{v_{w2} - v_{w1}}{2b}$$

$$v_{w_i} = \frac{2\pi r f \Delta_{\text{enc}}}{\text{ticks per rev}}$$

## A Simple Controller

The pseudo-code for a simple motor controller looks like this:

```
error = desired_w - estimated_w
int_error = int_error + error * dt
```

```
pwm = alpha * error + beta * int_error
```

Where **desired\_w - estimated\_w** is the *angular velocity error*. **desired\_w** is the desired angular velocity at which we want the wheel to rotate (commonly expressed in radians/second), while **estimated\_w** is the angular velocity at which the wheel is actually rotating. **alpha** is a positive control gain that you have to tune yourself by running the controller and seeing how the robot moves.  $\Delta t$  is the time difference between two consecutive iterations of the controller.

The desired angular velocity for each wheel depends on the input twist (linear + angular velocity) and can be calculated using the kinematic equations described previously.

**TIP for tuning the controller gain:** start with LOW gains, and slowly increase until you have a decent control performance or before the system becomes unstable. Keep in mind that the gains for each of the motors could be different.

When you have implemented your Cartesian controller you can run it

```
ros2 run controller cartesian_controller
```

Remember to also run

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

in a different terminal/tab.

To make the robot move, either use `rqt` and publish to the **/motor\_controller/twist** topic or directly in the terminal

```
ros2 topic pub /motor_controller/twist geometry_msgs/msg/Twist '{linear: {x: 1.0, y: 0.0, z: 0.0}, angular: {x: 0.0, y: 0.0, z: 0.3}}' -r 10
```

## Tips

- A PI controller works well for the duty cycle commands.
- Use radians not degrees for the kinematic calculations.