

Robotik - exercise 3

Team GIR : Evangelia Koumartzi , Julia Schuch

WiSe 2021/22

Assignment 3-1: Simple Parking Maneuver

The goal of this task is to park the model car between two virtual boxes.

Please look at the source code of the `simple_drive_control` and `simple_parking_maneuver` packages:

<https://github.com/AutoMiny/AutoMiny-exercises>

You should have a copy of these two packages in your repository. We provide `empty.world` with the assignment which needs to replace the existing file in `autominy/catkin_ws/src/autominy_simulator`. This places two boxes in your world which you should park the car in between.

```
cd autominy/catkin_ws/src
git clone https://github.com/AutoMiny/AutoMiny-exercises
catkin build simple_parking_maneuver
source devel/setup.bash # for bash
source devel/setup.zsh # for zsh
```

here is the sourcecode of the maneuver opened in my repository:

```

ros@pico:~/autonomy/kin_4/src/autonomy-exercises/simple_parking_manuever$ cat
simple_parking_manuever.py
#!/usr/bin/env python3
import rospy
from std_msgs.msg import String
from simple_drive_control.srv import DrivingManeuver
from nav_msgs.msg import Odometry

class SimpleParkingManeuver:
    def __init__(self):
        self.driving_manuever_client = rospy.ServiceProxy('driving_manuever', DrivingManeuver)
        self.parking_service = rospy.Service('parking_manuever', ParkingManeuver, self.parking_manuever)
        rospy.loginfo(rospy.get_caller_id() + "SimpleParkingManeuver initialized")

    def parking_manuever(self, request):
        rospy.loginfo(rospy.get_caller_id() + "SimpleParkingManeuver: direction = " + request.direction)
        # you use call the driving manuever service like this
        # direction can be backward/forward, steering can be left/right/straight
        # self.driving_manuever_client.call(direction="backward", steering="left", distance=1)

        if request.direction == "left":
            self.driving_manuever_client.call(direction="backward", steering="left", distance=0.5)
        elif request.direction == "right":
            self.driving_manuever_client.call(direction="backward", steering="left", distance=0.3)
        else:
            return ParkingManeuverResponse()
            # ERROR: Request can only be 'left' or 'right'

        return ParkingManeuverResponse("FINISHED")

if __name__ == '__main__':
    rospy.init_node('simple_parking_manuever')
    sm = SimpleParkingManeuver()
    rospy.spin()

```

```

ros@pico:~/autonomy/kin_4/src/autonomy-exercises/simple_drive_control$ cat
simple_drive_control.py
#!/usr/bin/env python3
import rospy
from math import sqrt
from autonomy_msgs.msg import SpeedCommand, NormalizedSteeringCommand
from simple_drive_control.srv import DrivingManeuver
from nav_msgs.msg import Odometry

class DriveControl:
    def __init__(self):
        self.speed = 0.3 # m/s
        self.angle_left = 0.3
        self.angle_straight = 0.0
        self.angle_right = -0.3

        self.request_time = rospy.Time() # to check for a timeout
        self.timeout = 30 # timeout after this amount of seconds
        self.distance = 0.0 # current drive distance
        self.odom = None # current position
        self.active = False

        rospy.init_node('simple_drive_control')
        self.speed_pub = rospy.Publisher("actuators/speed", SpeedCommand, queue_size=1)
        self.steering_pub = rospy.Publisher("actuators/steering", NormalizedSteeringCommand, queue_size=1)
        self.odom_sub = rospy.Subscriber("amcl/initialization/filtered_odom", Odometry, self.on_odom, queue_size=10)
        self.service_client = rospy.Service('driving_manuever', DrivingManeuver, self.drive, buffer_size=1)

    # calculates the distance if manuever is active
    def on_odom(self, msg):
        if self.odom is None and not self.active:
            self.odom = msg
            return

        self.distance += sqrt((self.odom.pose.pose.position.x - msg.pose.pose.position.x) ** 2 +
                              (self.odom.pose.pose.position.y - msg.pose.pose.position.y) ** 2)
        self.odom = msg

    # execute a simple manuever
    def drive(self, req):
        self.request_time = rospy.Time.now()
        self.distance = 0
        self.active = True

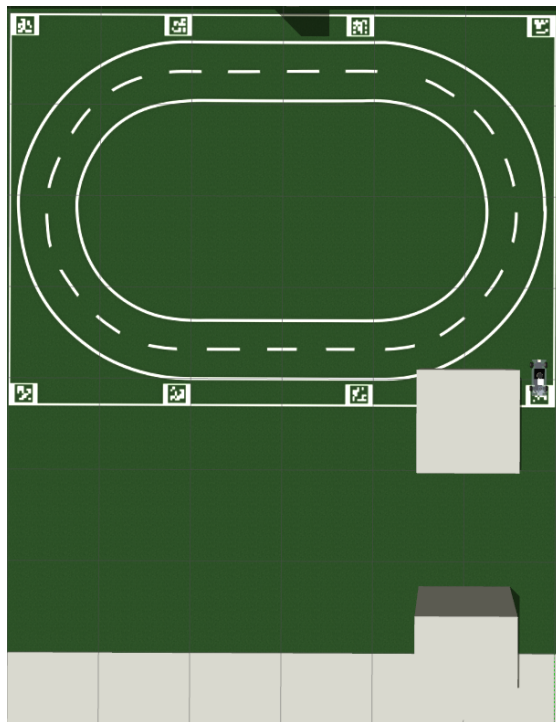
        # use the req direction command
        steering_cmd = NormalizedSteeringCommand()
        if req.steering == "left":
            steering_cmd.value = self.angle_left
        elif req.steering == "right":
            steering_cmd.value = self.angle_right
        elif req.steering == "straight":
            steering_cmd.value = self.angle_straight
        else:
            return False

        self.steering_pub.publish(steering_cmd)

        # use the req speed command
        if req.direction == "forward":
            direction = 1.0
        else:
            direction = -1.0

```

and here the new emty world can be seen:



This task is based on a given *driving_manuever* service which can execute simple driving maneuvers. We provide a launch file to start the driving maneuver service and your service conveniently:

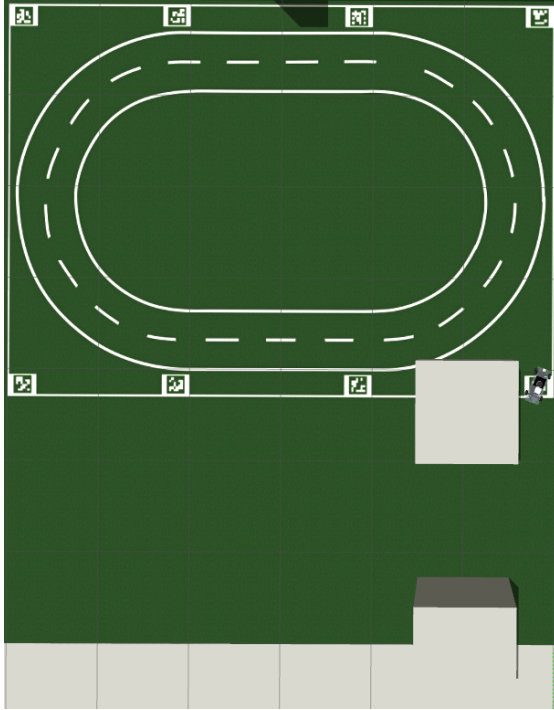
```
roslaunch simple_parking_manuever simple_parking_manuever.launch
```

You can start the parking maneuver by calling the *parking_manuever* service:

```
rosservice call /parking_maneuver "direction: 'left'"
```

here you can see the default parking maneuver:

```
pecco@fg7:~/autominy/catkin_ws$ rosservice call /parking_maneuver "direction: 'left'"  
info: "FINISHED"
```



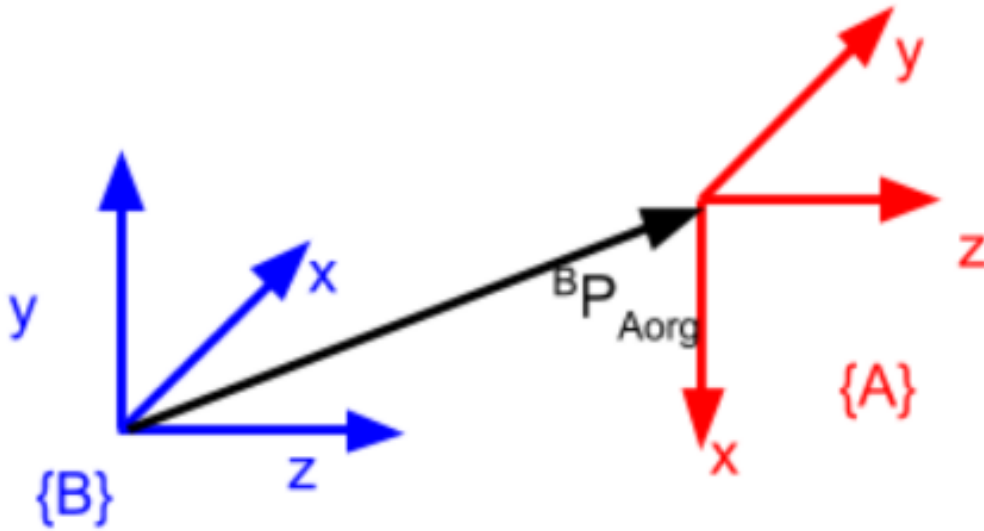
The default maneuver will not park the car properly between the boxes. Your task is to tune the parameters and the driving maneuver sequence in the `parking_maneuver.py` file.

https://github.com/AutoMiny/AutoMiny-exercises/blob/master/simple_parking_maneuver/src/parking_maneuver.py

The adjusted parking maneuver can be found here:

https://github.com/evakoumartzi/catkin_ws_GIR

Assignment 3-2: Coordinate System Transformation



Please provide the homogenous transformation matrix ${}^B_A T$, which maps a vector represented in coordinate frame {A} into the coordinate frame {B}. The translation vector between both coordinate frames is ${}^B P_{A_{org}} = (-1, 4, 5)$.

The homogeneous transformation matrix ${}^B_A T$ is composed of a rotational Part ${}^B_A R$ and a translational part ${}^B P_{A_{org}}$.

$${}^B_A T = \begin{bmatrix} {}^B_A R & {}^B P_{A_{org}} \end{bmatrix} \quad (1)$$

$${}^B_A R = ({}^B X_A, {}^B Y_A, {}^B Z_A) \quad (2)$$

$$= \begin{bmatrix} X_B \cdot X_A & X_B \cdot Y_A & X_B \cdot Z_A \\ Y_B \cdot X_A & Y_B \cdot Y_A & Y_B \cdot Z_A \\ Z_B \cdot X_A & Z_B \cdot Y_A & Z_B \cdot Z_A \end{bmatrix} \quad (3)$$

The scalar product is given with $\vec{a} \cdot \vec{b} = \cos(\gamma)|a||b|$. The in a Cartesian coordinate system the unity vectors (X, Z, Y) are of length 1 the scalar product will just be the cosine of the angle between the axes of the original system {B} and the new system {A}. Since in the exercise no more information is given in the exercise we believe that the angles between the axes of the different systems are supposed to be exactly $\angle(Y_B, X_A) = 180^\circ, \angle(X_B, Y_A) = \angle(Z_B, Z_A) = 0^\circ$ and $\angle(X_B, X_A) = \angle(X_B, Z_A) =$

$\angle(Y_B, Y_A) = \angle(Y_B, Z_A) = \angle(Z_B, X_A) = \angle(Z_B, Y_A) = 90^\circ$. Now we can set up the Rotation Matrix as:

$${}^A_B R = \begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (4)$$

So the homogeneous transformation matrix ${}^B_A T$ is given as:

$${}^B_A T = \begin{bmatrix} 0 & 1 & 0 & -1 \\ -1 & 0 & 0 & 4 \\ 0 & 0 & 1 & 5 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (5)$$

What is the inverse of your transformation matrix? To determine the inverse form of a matrix there are several different ways, we use cramer's rule:

$$A^{-1} = \frac{1}{\det(A)} * \text{adj}(A) \quad (6)$$

$$\det({}^B_A T) = -0 \cdot \begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 4 \\ 0 & 1 & 5 \end{bmatrix} + 0 \cdot \begin{bmatrix} 0 & 0 & -1 \\ -1 & 0 & 4 \\ 0 & 1 & 5 \end{bmatrix} - 0 \cdot \begin{bmatrix} 0 & 1 & -1 \\ -1 & 0 & 4 \\ 0 & 0 & 5 \end{bmatrix} + 1 \cdot \begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (7)$$

$$= 1 * \left(+0 \cdot \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix} - (-1) \cdot \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} + 0 \cdot \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} \right) \quad (8)$$

$$= 1 \cdot 1 \cdot (1 \cdot 1 - 0 \cdot 1) \quad (9)$$

$$= 1 \quad (10)$$

So it follows that ${}^B_A T$ has an inverse form (since $\det({}^B_A T) \neq 0$) which is given by ${}^B_A T^{-1} =$

$$adj({}^B_A T).$$

$${}^B_A T^{-1} = adj({}^B_A T) \quad (11)$$

$$= \begin{bmatrix} \det \begin{bmatrix} 0 & 0 & 4 \\ 0 & 1 & 5 \\ 0 & 0 & 1 \end{bmatrix} & -\det \begin{bmatrix} 1 & 0 & -1 \\ 0 & 1 & 5 \\ 0 & 0 & 1 \end{bmatrix} & \det \begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 4 \\ 0 & 0 & 1 \end{bmatrix} & -\det \begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 4 \\ 0 & 1 & 5 \end{bmatrix} \\ -\det \begin{bmatrix} -1 & 0 & 4 \\ 0 & 1 & 5 \\ 0 & 0 & 1 \end{bmatrix} & \det \begin{bmatrix} 0 & 0 & -1 \\ 0 & 1 & 5 \\ 0 & 0 & 1 \end{bmatrix} & -\det \begin{bmatrix} 0 & 0 & -1 \\ -1 & 0 & 4 \\ 0 & 0 & 1 \end{bmatrix} & \det \begin{bmatrix} 0 & 0 & -1 \\ -1 & 0 & 4 \\ 0 & 1 & 5 \end{bmatrix} \\ \det \begin{bmatrix} -1 & 0 & 4 \\ 0 & 0 & 5 \\ 0 & 0 & 1 \end{bmatrix} & -\det \begin{bmatrix} 0 & 1 & -1 \\ 0 & 0 & 5 \\ 0 & 0 & 1 \end{bmatrix} & \det \begin{bmatrix} 0 & 1 & -1 \\ -1 & 0 & 4 \\ 0 & 0 & 1 \end{bmatrix} & -\det \begin{bmatrix} 0 & 1 & -1 \\ -1 & 0 & 4 \\ 0 & 0 & 5 \end{bmatrix} \\ -\det \begin{bmatrix} -1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix} & \det \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix} & -\det \begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} & \det \begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \end{bmatrix} \quad (12)$$

$$= \begin{bmatrix} 0 & -1 & 0 & 4 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & -5 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (13)$$

Assignment 3-3: Coordinate Frames

Assume, you have the following vectors for coordinate axes of frame (or coordinate system):

$$\{A\} : x = (-\sqrt{0.5}, \sqrt{0.5}, 0); \quad y = (\sqrt{0.5}, \sqrt{0.5}, 0)$$

Calculate the vector for the z -axis of this frame $\{A\}$

In a cartesian coordinate system the unit vectors along the axes have to be perpendicular (90°) to each other and of the length 1. This can easily be determined by the skalar product:

$$x_i \cdot x_j = \begin{cases} 1 & , \text{if } i = j \\ 0 & , \text{if } i \neq j \end{cases} \quad (14)$$

this is given for x and y . Since x and y are both set in the x - y -plane of a naive Cartesian coordinate system the 3rd axes will be along the z -axes of the naive system, so we can set the 3rd unit vector to be either:

$$\{A\} : z = (0, 0, 1); \text{ or}$$

$$\{A\} : z = (0, 0, -1);$$

to choose which of these possible vectors is the right one, lastly comes into play that that conventionally the 3rd axes is chosen so that the 90° angle from x to y is counter-clockwise (right hand rule). We can calculate the correct z - unit vector by using the vector product:

$$z = x \times y \tag{15}$$

$$= (0, 0, -1) \tag{16}$$

.

The length of a vector product can be determinedd by $|a \times b| = |a||b| \cdot \sin(\theta)$. Since the angel between the unit vectors is 90° and x and y have unit length, we already know that also z has unit length, so we are done.