Problem set 2 for CS 498 GC

☐ This homework consists of a coding part only, which must be submitted individually at Gradescope. You only need to upload the python file *problem_set2.py* (see attached template)

Total points: 100

Question 1 (30 points). Figure 1 shows three coordinate frames (a,b and c). Given the rotation matrices Rab and Rbc, and the relative translations Tab and Tbc, complete the function *Transformations* (in *problem_set2.py*) to return the rotation (in matrix, quaternion and euler angles) and translation from c to a.

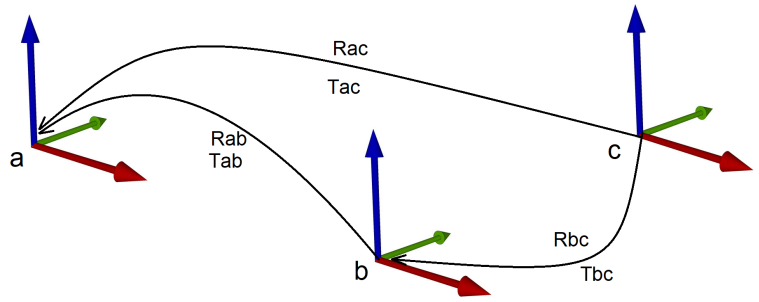


Figure 1. Coordinate frames
*Rab: passes points expressed in reference frame b to reference frame a
*Tab is the translation of b with respect to a.

You can either implement the function from scratch using the numpy library or implement the function using the scipy.spatial.transform library (check functions *Rotation.from_matrix*, *Rotation.as_euler*, *Rotation.as_quad*).
https://docs.scipy.org/doc/scipy/reference/generated/scipy.spatial.transform.Rotation.from_matrix.html
Feel free to apply homogeneous transformations theory if you want.

Example of inputs and expected outputs

```
Rab = np.array([[0,0,1],[-1,0,0],[0,-1,0]])
Rbc = np.array([[1,0,0],[0,0,1],[0,-1,0]])
Tab = np.array([1,2,3])
Tbc = np.array([4,5,6])
Rac, quat, euler, Tac = transformations(Rab, Rbc,Tab, Tbc)

Must return:
Rac = [[ 0. -1.  0.]
       [-1.  0. -0.]
       [ 0.  0. -1.]]
quat = [ 0.70710678 -0.70710678  0.          0.        ]
euler =  [ 3.14159265  0.         -1.57079633]
Tac =  [ 7 -2 -2]
```

Question 2 (70 points). In this exercise, we will control a differential drive robot to follow a desired trajectory.

The kinematic of a differential drive mobile robot described in the inertial frame {xI, yI,θ} is given by

$$^{I}\begin{bmatrix}\dot{x}\\ \dot{y}\\ \dot{\theta}\end{bmatrix} = \begin{bmatrix}\cos\theta & 0\\ \sin\theta & 0\\ 0 & 1\end{bmatrix}\begin{bmatrix}v\\ \omega\end{bmatrix}$$

where x' and y' are the linear velocities in the direction of the xI and yI of the inertial frame.

Let α denote the angle between the xR axis of the robot's reference frame and the vector connecting the center of the axle of the wheels with the final position.

Then consider the coordinate transformation into polar coordinates with its origin at the goal position:

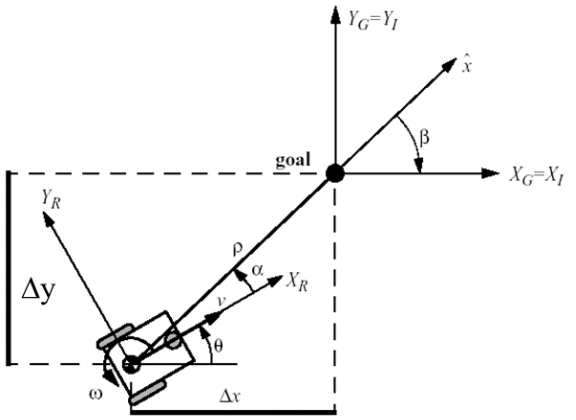
$$\rho = \sqrt{\Delta x^2 + \Delta y^2}\,.$$

$$\alpha = -\theta + \operatorname{atan2}(\Delta y, \Delta x)\,.$$

$$\beta = -\theta - \alpha\,.$$

References

Siegwart, R., Nourbakhsh, I. R., & Scaramuzza, D. (2011). Introduction to autonomous mobile robots. MIT press.



Control commands: v, w

We can control the robot by regulating the linear and angular velocities (v and w) with PID controllers. Rho and alpha in the last equations represent the error from the robot to the goal that we want to achieve. As the desired trajectory is changing with time, we need to continuously update the state of the robot (x, y, theta) and the desired goal (xd, yd).

a). Complete the function *update_robot_state* (using euler integration and the kinematic model) to update the values of self.x, self.y and self.theta. Use self.dt as the sampling interval, self.v as the forward velocity and self.w as the angular velocity.

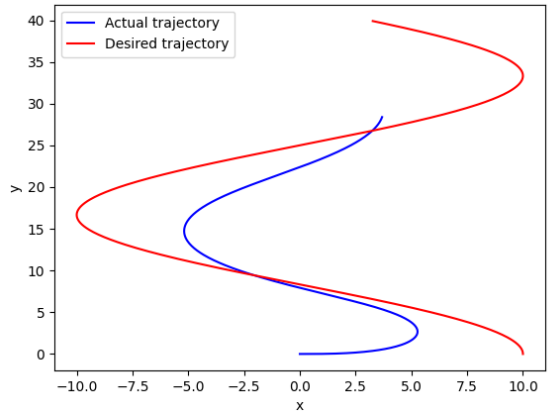
b). Complete the function *compute_error* to return the error in angle and distance to the goal.

c). Look at the *compute_control* function. It returns the control commands v and w. Edit the pid_w controller (at the beginning of the class), setting the output limits of angular velocity from -5 rad/s to 5 rad/s.
Further details on the PID library: https://github.com/m-lundberg/simple-pid
Install with: >>pip install simple-pid

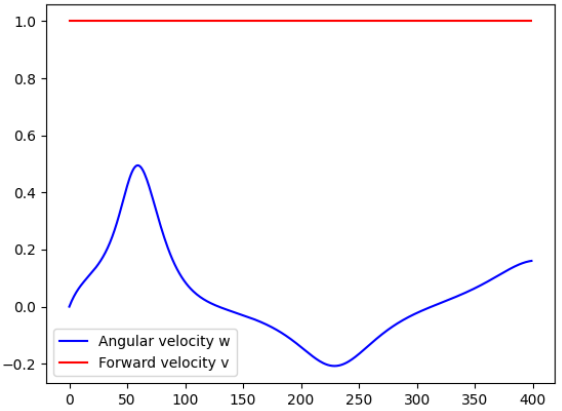
d). Run the main function to get the desired and actual trajectories. Since the robot is not following the trajectory accurately, edit the function *compute_control,* by adding another PID controller to control the distance error by changing the forward velocity. Make sure to declare a new pid object at the beginning of your class with output limits between 0 and 2 m/s. The set_points should be zero, as we want the error to be zero. Edit the controllers' gains to follow the actual trajectory as close as possible.

e). Edit the function *desired_trajectory* to make the robot follow a circle with radius 10 m at a frequency of 0.03 Hz.

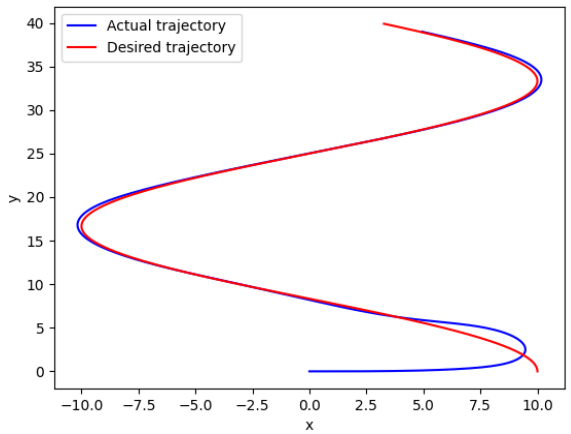
Expected results



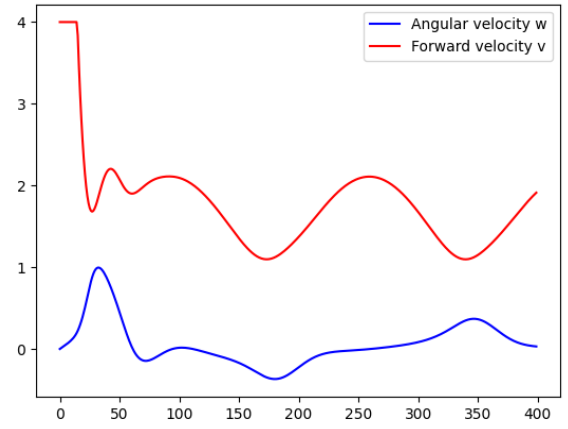
Controlling angle to the goal only



Control commands



Controlling angle and distance to the goal



Control commands