Homework 2 (Python version)

ME570 - Prof. Tron

2021-09-21

The goal of this homework is to practice the notions about configuration spaces seen in class. In the first part of the homework you will see how the same configuration spaces applies to a "physical" two-link manipulator. In the second part of the homework you will go through a couple of examples of viewing a configuration space as a differential manifold and the corresponding implications (charts, embeddings, transfer of curves and tangents). On the way, you will learn more about rotation matrices and how to pass functions as arguments of other functions in Python.

General instructions

Programming For your convenience, together with this document, you will find a `zip` archive containing Python files with stubs for each of the questions in this assignment; each stub contains an automatically generated description and header of the function or class. You will have to complete these files with the requested code. The goal of this files is to save you a little bit of time, and to avoid misspellings in the function or argument names. The files for the parts marked as **provided** (see also the *Grading* paragraph below) contain already the body of the function.

Homework help For best coding practices, please refer to the guidelines on Blackboard under Class content/Programming Tips & Tricks/Python. For questions specific to the content of the homework, please post on the Blackboard discussion board.

Homework report Along the programming of the requested functions, prepare a PDF report containing one or two sentences of comments for each question marked as **report**, and including: embedded figures and outputs that are representative of those generated by your code. Include comments on the questions marked as **code** only to explain any difficulty you might have encountered.

A small amount of *beauty points* are dedicated to reward reports that present their content in a professional way (see the *Grading criteria* section in the syllabus).

Analytical derivations To include the analytical derivations in your report you can type them in LaTeX(preferred method), any equation editor or clearly write them on paper and use a scanner (least preferred method).

Submission

The submission will be on Gradescope through four separate assignments: two for the questions marked as **code**, and one for those marked as **report**, and one for providing

feedback. Further details are explained below. You can submit as many times as you would like, up to the assignment deadline. Each question is worth 1 point unless otherwise noted. Please refer to the Syllabus on Blackboard for late homework policies.

Report Upload the PDF of you report, and then indicate, for each question marked as **report**, on which page it is answered (just follow the Gradescope interface). Note that some of the questions marked as **report** might include a coding component, which however will be evaluated from the output figures you include in the report. In general, these questions are intended as checkpoints for you to visually check the results of your functions.

Code questions Upload all the necessary `.py` files, both those written by you, and those provided with the assignment. You will see *two* assignments on Gradescope. In the one marked *Python files*, you will submit the `.py` files directly; Gradescope will run one test checking for completeness, and one test checking the style (using Pylint); these automated tests will not check the correctness your answers. In the other assignment marked *Python PDF listing*, please submit a PDF containing a print-out of the contents of the `.py` files (see the footnote[1] for how to generate such file). I will use this to manually grade the code. However, you can use the output of the test functions to judge if your code gives correct results or not.

Optional and provided questions. Questions marked as **optional** are provided just to further your understanding of the subject, and not for credit (if submitted, I will provide comments but it will not count toward your grade).

Hints

Some hints are available for some questions, and can be found at the end of the assignment (you are encouraged to try to solve the questions without looking at the hints first). If you use these hints, please state so in your report (your grading will not change based on this information, but it is a useful feedback for me).

Use of external libraries and toolboxes You are **not allowed** to use functions or scripts from external libraries or toolboxes (e.g., mapping toolbox), unless specifically instructed to do so (e.g., CVX).

Problem 1: Rotations

A 2-D rotation $R \in \mathbb{R}^{2\times 2}$ rotating points in a counterclockwise manner can be expressed as:

$$R(\theta) = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix}. \tag{1}$$

Question provided 1.1. Write a Python function that implements (1); this simple function will be used in subsequent problems.

[1] The provided `.zip` file includes a `pygments_homework2.py` file that you can use to generate a HTML file with all the necessary pretty-printed listings via the command `python pygments_homework2.py`. The script requires the `pygments` Python package. You can then generate a PDF file using a browser and its *"Print to pdf"* functionality.

File name: `me570_geometry.py`
Function name: `rot2d`
Description: Create a 2-D rotation matrix from the angle `theta` according to (1).
Input arguments

- `theta` (dim. $[1 \times 1]$): An angle θ.

Output arguments

- `rot_theta` (dim. $[2 \times 2]$): A matrix containing $R(\theta)$.

Question report 1.1 (0.5 points). Eq. 1 can also be used to build 3-D rotations for some particular axes of rotation. Let $r_{ij}(\theta)$ represent the individual entries of the matrix $R(\theta)$, i.e. $R(\theta) = \begin{bmatrix} r_{11}(\theta) & r_{12}(\theta) \\ r_{21}(\theta) & r_{22}(\theta) \end{bmatrix}$. Explain the geometrical meaning (i.e., the rotation axis and the direction of rotation) of the following 3-D rotations:

$$R_1(\theta) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & r_{11}(\theta) & r_{12}(\theta) \\ 0 & r_{21}(\theta) & r_{22}(\theta) \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\theta) & -\sin(\theta) \\ 0 & \sin(\theta) & \cos(\theta) \end{bmatrix} \tag{2}$$

$$R_2(\theta) = \begin{bmatrix} r_{11}(\theta) & 0 & r_{12}(\theta) \\ 0 & 1 & 0 \\ r_{21}(\theta) & 0 & r_{22}(\theta) \end{bmatrix} = \begin{bmatrix} \cos(\theta) & 0 & -\sin(\theta) \\ 0 & 1 & 0 \\ \sin(\theta) & 0 & \cos(\theta) \end{bmatrix} \tag{3}$$

$$R_3(\theta) = \begin{bmatrix} r_{11}(\theta) & r_{12}(\theta) & 0 \\ r_{21}(\theta) & r_{22}(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{4}$$

$$R_4(\theta) = \begin{bmatrix} -r_{11}(\theta) & -r_{12}(\theta) & 0 \\ -r_{21}(\theta) & -r_{22}(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} -\cos(\theta) & \sin(\theta) & 0 \\ -\sin(\theta) & -\cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{5}$$

Hints are available for this question.

Problem 2: Free configuration space for a two-link manipulator

In this problem we will simulate a 2-D two-link manipulator moving in a workspace containing numerous point obstacles, with functions that plot the manipulator and check for collisions.

The 2-D manipulator is composed of the two links shown in Figure 1. The vertices of the two links are given by the output of the function `TwoLink.polygons(_)` (which was provided in Homework 1).

We consider three reference frames: $\mathcal{W}$, which is fixed to the world, $\mathcal{B}_1$, which rotates with the first link, and $\mathcal{B}_2$, which rotates with the second link.

The translation of the first link in the world reference frame is zero, i.e., ${}^{\mathcal{W}}T_{\mathcal{B}_1} = 0$, while the rotation ${}^{\mathcal{W}}R_{\mathcal{B}_1}$ has angle θ_1; the translation of the second link in the first link's reference is ${}^{\mathcal{B}_1}T_{\mathcal{B}_2} = \begin{bmatrix} 5 \\ 0 \end{bmatrix}$, while the rotation ${}^{\mathcal{B}_1}R_{\mathcal{B}_2}$ has angle θ_2 (see also the annotations in Figure 1).

Question report 2.1 (1.5 points). Derive two separate expressions that compute the coordinates of a point p expressed in $\mathcal{W}$, ${}^{\mathcal{W}}p$, for each one of these inputs, respectively:

1) The coordinates of the point given in $\mathcal{B}_1$, denoted as ${}^{\mathcal{B}_1}p$;

2) The coordinates of the point given in $\mathcal{B}_2$, denoted as ${}^{\mathcal{B}_2}p$.

Please make sure that the use of rotation matrices is clear, otherwise you might not receive full credit for your answer; although we will use these computations for 2-D points, the same expressions should hold for 3-D points.

rtron Fix one grid_struct, one ifmtlbb

Question code 2.1. Create a function to compute the coordinates of the end effector and of the vertices of the manipulator for a given configuration. The end effector is defined to be the point ${}^{\mathcal{B}_2}p_{\text{eff}} = \begin{bmatrix} 5 \\ 0 \end{bmatrix}$.

> Class name: `TwoLink`
> File name: `me570_robot.py`

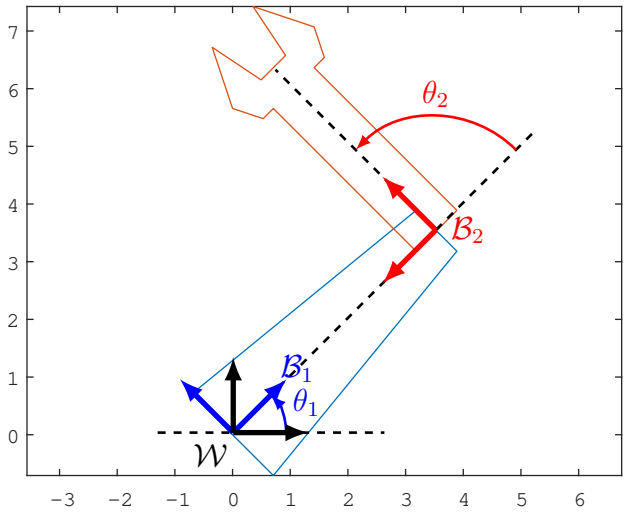

Figure 1: The two-link manipulator for this exercise. The angle of the first link θ_1 is measured from the horizontal x axis, the angle of the second link θ_2 is measured with respect to the axis of the first link.

Method name: `kinematic_map`
Description: The function returns the coordinate of the end effector, plus the vertices of the links, all transformed according to θ_1, θ_2.
Input arguments

- `theta` (dim. $[2 \times 1]$) A vector containing `theta(1)` $= \theta_1$ and `theta(2)` $= \theta_2$, the two joint angles for the two-link manipulator.

Output arguments

- `vertex_effector_transf` (dim. $[2 \times 1]$): The coordinates of the point ${}^{\mathcal{B}_2}p_{\text{eff}}$ transformed according to θ_1, θ_2 (i.e., the coordinates of the points in the world frame); this point represents the transformed end effector of the manipulator.
- `polygon1_transf` (type `Polygon`), `polygon2_transf` (type `Polygon`): the polygons given by the method `TwoLink.polygons()`, transformed according to θ_1, θ_2.

Requirements: Use the results from Question report 2.1 to guide your implementation. This function must use `TwoLink.polygons()` to obtain the vertices of the polygons of the matrix, and it must use `rot2d()` from Question provided 1.1. Note that here we are simply computing the vertices of the transformed polygons, without plotting them. The next function will be used to plot the transformed vertices.

optional Create another method `kinematic_map(i)` in the class `Polygon` to generate the transformed versions of the polygons.

Question provided 2.1. Create a function that plots the manipulator in a given configuration according to the following.

Class name: `TwoLink`
File name: `me570_robot.py`

Method name: `plot`
Description: This function should use `TwoLink.kinematic_map()` from the previous question together with the method `Polygon.plot()` from Homework 1 to plot the manipulator.
Input arguments

- `theta` (dim. $[2 \times 1]$): A vector containing the configuration angles as used by `TwoLink.kinematic_map()`.
- `color` (dim. $[1 \times 1]$, type `string`): Color specification (e.g., `'r'`, `'g'`, `'b'`).

Note that, by default, this plotting function does not enforce the same proportions for the horizontal and vertical axes (i.e., the manipulator might appear skewed). To enforce this, you can use `[ax]=matplotlib.pyplot.gca()` to get the current axes, and then the

command `ax.axis(['equal'])` (.)

Question code 2.2. Create a function that checks for collisions between the manipulator and a sparse set of points (representing obstacles).

> Class name: `TwoLink`
> File name: `me570_robot.py`
>
> > Method name: `is_collision`
> > Description: For each specified configuration, returns `True` if *any* of the links of the manipulator collides with *any* of the points, and `False` otherwise. Use the function `Polygon.is_collision` ( ) to check if each link of the manipulator is in collision.
> > Input arguments
> >
> > - `theta` (dim. [2 × `nb_theta`]): An array of configurations where each column represents a different configuration according to the convention used by `TwoLink.kinematic_map` ( ).
> > - `points` (dim. [2 × `nb_points`]): An array of coordinate of points (obstacles). Each column represents a different point.
> >
> > Output arguments
> >
> > - `flag_theta` (dim. [1 × `nb_theta`], type `bool`): each element of this array should be `True` if the configuration specified in the corresponding column of `theta` causes a collision, and `False` otherwise.
> >
> > Requirements: For this question, *do not* consider self-collision (i.e., if the two polygons overlap but they do not cover any of the points, then it is not a collision).

Question report 2.2. Make a function that combines the functions from the previous two questions.

> Class name: `TwoLink`
> File name: `me570_robot.py`
>
> > Method name: `plot_collision`
> > Description: This function should:
> >
> > *1)* Use `TwoLink.is_collision` ( ) for determining if each configuration is a collision or not.
> >
> > *2)* Use `TwoLink.plot` ( ) to plot the manipulator for all configurations, using a red color when the manipulator is in collision, and green otherwise.
> >
> > *3)* Plot the points specified by `points` as black asterisks.
> >
> > Input arguments
> >
> > - `theta` (dim. [2 × `NTheta`]), `points` (dim. [2 × `NPoints`]): As used by `TwoLink.is_collision` ( )

All the drawings from the different configurations should overlap on the same figure (it is fine if many configurations overlap). You should use the command `axis equal` if you want to avoid seeing the manipulator "distorted" (e.g., to see right angles as real right angles on the screen). In your report, include the output of the function `twolink_plot_collision_test(_)` provided in the next question.

Question **provided** **2.2.** This function loads data from the file `twolink_testData.mat` (provided with the homework) and it can be used to test and see if the other functions from this problem are working correctly.

> File name: `me570_hw2.py`
> Method name: `twolink_plot_collision_test`
> Description: This function generates 30 random configurations, loads the `points` variable from the file `twolink_testData.mat` (provided with the homework), and then display the results using `twolink_plotCollision` to plot the manipulator in red if it is in collision, and green otherwise.

Question **optional** **2.1.** Write a method `TwoLink.free_space(_)` that loads points from `twolink_testData.mat`, calls `TwoLink.is_collision(_)` for θ_1, θ_2 sampled along a regular fine grid, and stores the results in a matrix of logical values `free_space_map`. Display `free_space_map` as an image. This is an approximate representation of the free configuration space. You can use `matplotlib.pyplot.imshow(_)` to plot the map as an image. Using the functions above, generating the map should be straightforward, but it might take a very long time.

Problem 3: Grids, function handles, and evaluating functions on grids

In this homework and in future assignments, we will need to evaluate different functions on discretized domains (i.e., on regular grids). To aid in this representation, we will use a class `Grid` with the following constructor.

> Class name: `Grid`
> File name: `me570_geometry.py`
> Description: A function to store the coordinates of points on a 2-D grid and evaluate arbitrary functions on those points.
>
> > Method name: `__init__`
> > Description: Stores the input arguments in attributes.
> > Input arguments
> >
> > - `xx_grid` (dim. [1 × `nb_grid`]), `yy_grid` (dim. [1 × `nb_grid`]): arrays containing, respectively, the x and y values corresponding to the horizontal and vertical lines of the grid.

Function handles. In Python, functions are *first-class objects*, i.e., they can be stored in variables and passed to other functions. For instance, consider the function `math.sin(_)` in the

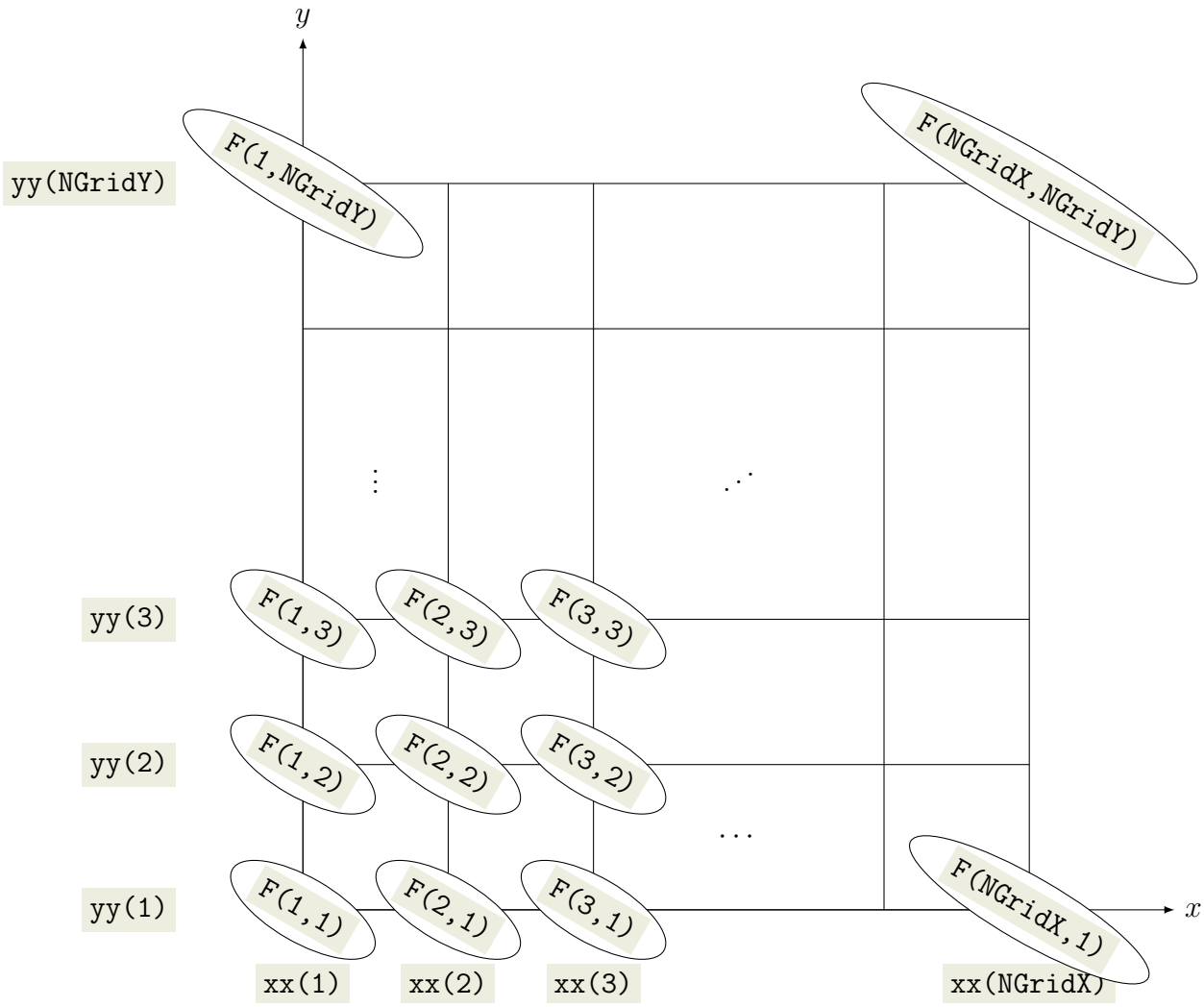

Figure 2: Representation a (possibly vector-valued) function on a grid domain. The values in `xx_grid` and `yy_grid` contain the x and y coordinates of, respectively, the vertical and horizontal lines defining the grid (for space reasons, `xx_grid` and `yy_grid` are abbreviated as `xx` and `yy`, respectively). The variable `F` contains the value (possibly a vector) associated to each location. Notice the organization of the indexes of `F` with respect to `xx_grid` and `yy_grid`; this organization is transposed with respect to what is expected by `plot_surface` ( ), but it has the advantage that the indeces of `F` align more naturally with the indeces of `xx_grid` and `yy_grid`.

`math` module. You can store[2] it in a variable `myfun` using the assignment `myfun=math.sin`; notice that, in the assignment, there are no parentheses (otherwise you would *call* the function, not store it). You can pass a function as an argument to another function in a similar way (i.e., using the name of the function without parentheses).

Question provided 3.1. Implement an object that represents a grid and that is used to evaluate functions on this grid.

> Class name: `Grid`
> File name: `me570_geometry.py`
>
> > Method name: `eval`
> > Description: This function evaluates the function `fun` (which should be a function) on each point defined by the grid.
> > Input arguments
> > - `fun` (type `function`): an handle to a function that takes a vector `x` of dimension [2 × 1] as a single argument, and returns a scalar or a vector. An example of this function could be the function `norm(_)`.
> >
> > Output arguments
> > - `fun_eval` (dim. [`nb_grid` × `nb_grid` × `d`]): the function `norm(_)` evaluated on each point of the grid.

We will use this method any time we will need to evaluate a map on a rectangular region. For instance, in Problem 5 `xx` and `yy` will correspond to the two entries of the vector of angles θ. You can find a very simple example of how to use it in the function `grid_eval_test(_)` (in the file `me570_hw2.py`).

Problem 4: Charts for the circle using rotations

In this question you will consider the circle $\mathbb{S}^1$ as a manifold, and consider charts based on rotation matrices. For this problem, most questions are optional, but understanding them will help you with the answers to Problem 5.

From angles to circles to the torus using rotations. Consider the following maps

$$\phi_{circle}(\theta) = R(\theta) \begin{bmatrix} 1 \\ 0 \end{bmatrix}. \tag{6}$$

Equation (1) defines a 2-D rotation with angle $\theta \in \mathbb{R}$, while (6) maps an angle θ to a point on the circle $\mathbb{S}^1$. By choosing different intervals for θ, we can cover different parts of the circle.

Question report 4.1. Show that $R(\theta)$ as given in (1) effectively is a rotation (i.e., $R(\theta) \in SO(2)$) and that $\phi_{circle}(\theta) \in \mathbb{S}^1$ for all $\theta \in \mathbb{R}$.

Question optional 4.1. Two charts are necessary and sufficient to cover the circle, for instance $\{U_1, \phi_{circle}\}$ and $\{U_2, \phi_{circle}\}$, where the intervals U_1, U_2 are $U_1 = (-\frac{3}{4}\pi; \frac{3}{4}\pi)$ and $U_2 = (\frac{1}{4}\pi; \frac{7}{4}\pi)$. Why do we need two charts to cover the circle?

[2] Technically, in Python, it would be more accurate to say that you are creating a new label for an object that is already in memory.

Question optional 4.2. Generate a vector `theta1` and `theta2` of points that uniformly cover the intervals U_1, U_2, respectively. Plot the results of applying ϕ_{circle} to `theta1` and `theta2` on the same plot, but in different colors. Explain what you see.

Problem 5: Charts for the torus using rotations

In this question you will consider the torus $T = \mathbb{S}^1 \times \mathbb{S}^1$ as a manifold, and use the material from the previous question to make charts for the torus and visualize an embedding in $\mathbb{R}^3$ together with some curves in the configuration space.

From angle pairs to the torus. A point on the torus can be identified with two angles, which we collect in a single vector $\vec{\theta} = \begin{bmatrix}\theta_1 \\ \theta_2\end{bmatrix}$. Let $r = 3$, and consider the map:

$$\phi_{torus}(\vec{\theta}) = \mathrm{diag}\big(R(\theta_2), 1\big) \left(\begin{bmatrix}1 & 0 \\ 0 & 0 \\ 0 & 1\end{bmatrix} \phi_{circle}(\theta_1) + \begin{bmatrix}r \\ 0 \\ 0\end{bmatrix} \right), \tag{7}$$

where diag is an operator that creates a block diagonal matrix with its arguments (in Matlab, this is implemented by the function `blkdiag(_)`), and R and ϕ are the same as in (1), (6). This map might look a little intimdating at first, but, intuitively, it boils down to setting the circle from Problem 4 onto the $x - z$ plane, translating it along the x axis, and then revolving it around the z axis. As you will see later, the map ϕ_{torus} can be used to embed a torus in $\mathbb{R}^3$, and use it to represent the configuration space of the two-link manipulator.

Question code 5.1. Implement the following function, which will be used to embed the torus plot in the Euclidean space $\mathbb{R}^3$.

> Class name: `Torus`
> File name: `me570_geometry.py`
> Description: A class that holds functions to compute the embedding and display a torus and curves on it.
>
> > Method name: `phi`
> > Description: Implements equation (7).
> > Input arguments
> >
> > - `theta` (dim. [2 × `nb_points`]): An array of values for $\vec{\theta}$ (each column corresponds to a different $\vec{\theta}$).
>
> > Output arguments
> >
> > - `x_torus` (dim. [3 × `nb_points`]): The values of $\phi_{torus}(\vec{\theta})$ corresponding to all the angles in `theta`.

Question report 5.1. Choose N square regions $U_i \subset \mathbb{R}^2$, $i \in \{1, \ldots, N\}$, such that these regions together with the map ϕ_{torus} define an atlas for the torus; what is the minimum sufficient and necessary number of charts N? This question is best answered while working on it in parallel with the next. You should *not* use charts that start and end at the same coordinates (e.g., $U_1 = (0, 2\pi) \times (0, 2\pi)$), but leave small "gaps" (e.g., $U_1 = (0, 2\pi - 0.1) \times (0, 2\pi - 0.1)$);

this typically does not fundamentally change the number of charts that you use, and it will make it easier to spot problems. In the report, include a picture illustrating the charts on the *flat* representation of the torus seen in class (the square with opposite sides glued together).

Question report **5.2.** Make a function that shows the embedding of the torus given by (7).

Class name: `Torus`
File name: `me570_geometry.py`

Method name: `plot_charts`
Description: For each one of the chart domains U_i from the previous question:

1) Fill a `grid` structure with fields `xx_grid` and `yy_grid` that define a grid of regular point in U_i. Use `nb_grid=33`.

2) Call the function `Grid.eval(_)` with argument `Torus.phi(_)`.

3) Plots the surface described by the previous step using the the Matplotlib function `ax.plot_surface(_)` (where `ax` represents the axes of the current figure) in a separate figure.

Plot a final additional figure showing all the charts at the same time.

optional To better show the overlap between the charts, you can use different colors each one of them, and making them slightly transparent.

Use the same charts that you gave for Question report 5.1. The output of the function should show a torus (i.e., a doughnut) with different sections in different colors. To give you an idea, Figure 3 gives an example of how the final figure would look like for a sphere (that, as seen in class, requires $N = 2$ charts).

Question report **5.3.** Using what seen in class, explain why, in the previous question (Question report 5.2), you should not see the same colored part overlap on itself, and why you should not see parts of the surface of the torus that are left uncovered.

Question provided **5.1.** The following function can be used to generate points on a curve $\theta(t)$ that is a straight line.

File name: `me570_geometry.py`
Function name: `line_linspace`
Description: Generates a discrete number of `nb_points` points along the curve $\vec{\theta}(t) =$ (`a(1)` $\cdot t +$ `b(1)`, `a(2)` $\cdot t +$ `b(2)`) $\in \mathbb{R}^2$ for t ranging from `tMin` to `tMax`.
Input arguments

- `a_line` (dim. $[2 \times 1]$), `b_line` (dim. $[2 \times 1]$): First and second parameter of the curve.
- `t_min` , `t_max` : Defines the interval for t.
- `nb_points` : Number of points in the interval to generate.

Output arguments

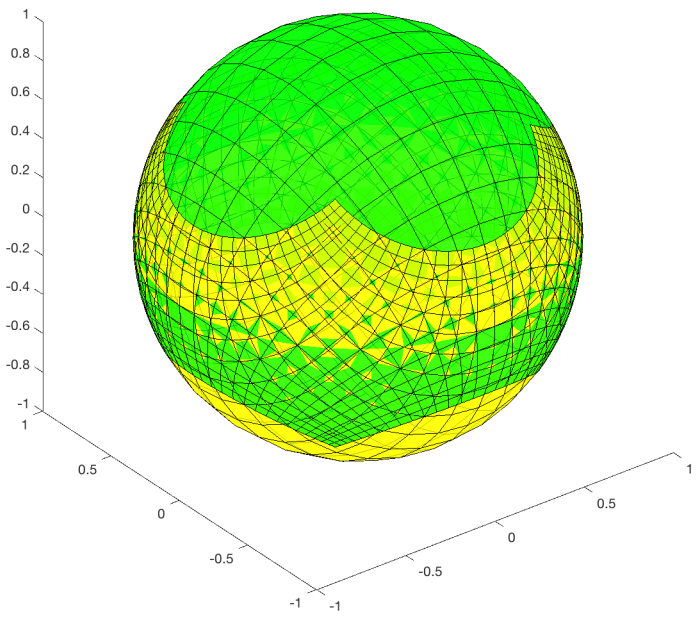

Figure 3: Example of visualization of a minimal choice of charts for the sphere.

- `theta_points` (dim. $[2 \times \mathtt{N}]$): The points generated along the curve $\vec{\theta}(t)$.

Question report 5.4 (0.5 points). Compute the tangent $\dot{\theta}(t)$ of the curve $\theta(t)$ defined in the previous question.

Question code 5.2. This question is based on the idea of generating a curve $\vec{\theta}(t) \in \mathbb{R}^2$ (in fact, a straight line) in the configuration space, "push it through" the embedding ϕ_{torus} to generate the curve $x(t) = \phi_{torus}\big(\vec{\theta}(t)\big) \in \mathbb{R}^3$.

Class name: `Torus`
File name: `me570_geometry.py`

Method name: `phi_push_curve`

Description: This function evaluates the curve $x(t) = \phi_{torus}\left(\vec{\theta}(t)\right) \in \mathbb{R}^3$ at `nb_points=31` points generated along the curve $\vec{\theta}(t)$ using `Line.linspace(_)` with `tMin` $= 0$ and `tMax` $= 1$, and a, b as given in the input arguments.

Input arguments

- `a_line` (dim. $[2 \times 1]$), `b_line` (dim. $[2 \times 1]$): Parameters of the curve to be passed to `Line.linspace(_)`.

Output arguments

- `x_points` (dim. $[3 \times$ `nb_points`$]$): Array of points generated by evaluating ϕ_{torus} on `thetaPoints` (i.e., array of points on the curve $x(t)$).

Question **report** **5.5.** The following function combine all the functions from this problem to visualize the embedding of the torus, together with four curves.

Class name: `Torus`

File name: `me570_geometry.py`

Method name: `plot_charts_curves`

Description: The function should iterate over the following four curves:

- `a_line=np.array([[3/4*pi],[0]])`,
- `a_line=np.array([[3/4*pi],[3/4*pi]])`,
- `a_line=np.array([[-3/4*pi],[3/4*pi]])`,
- `a_line=np.array([[0],[3/4*pi]])`,

and `b_line=np.array([[-1],[-1]])`. The function should show an overlay containing:

- The output of `Torus.plotCharts(_)`;
- The output of the functions `Torus.pushCurve(_)` for each one of the curves.

Requirements: This function needs to use `plot(_)` to show the output of `Torus.pushCurve(_)`. You should see that all the curves start at the same point on the torus.

optional Use different colors to display the results of the different curves.

Problem 6: Jacobians and end effector velocities

In this question you will create functions to compute and plot the velocity of the end effector of our two-link manipulator as a function of the position and velocities of the two links.

Recall that we defined the end effector of our two-link manipulator to be the point ${}^{\mathcal{B}_2}p_{\text{eff}} = \begin{bmatrix} 5 \\ 0 \end{bmatrix}$.

Question **report** **6.1.** Seeing the coordinates ${}^{\mathcal{W}}p_{\text{eff}}$ as a function from $\theta = \begin{bmatrix} \theta_1 \\ \theta_2 \end{bmatrix}$ to $\mathbb{R}^2$,

give an expression for computing $\frac{\mathrm{d}}{\mathrm{d}t}({}^{\mathcal{W}}p_{\text{eff}})$ as a function of $\dot{\theta} = \begin{bmatrix}\dot{\theta}_1 \\ \dot{\theta}_2\end{bmatrix}$. This expression is a map representing the Jacobian of the end effector. For this question, you can consult the hints, or use the Matlab Symbolic Toolbox (for the latter, save your code in the file `twolink_jacobian_sym.m`).

Question optional 6.1. Use the answer to the previous question (Question report 6.1) to find the Jacobian matrix J such that $\frac{\mathrm{d}}{\mathrm{d}t}({}^{\mathcal{W}}p_{\text{eff}}) = J\dot{\theta}$.

Question code 6.1. Make a function that implements the Jacobian map derived in the previous function.

> Class name: `TwoLink`
> File name: `me570_robot.py`
> Description: This class was introduced in a previous homework.
>
> > Method name: `jacobian`
> > Description: Implement the map for the Jacobian of the position of the end effector with respect to the joint angles as derived in Question report 6.1.
> > Input arguments
> >
> > - `theta` (dim. [2 × `nb_theta`]) An array where each column contains $\begin{bmatrix}\theta_1 \\ \theta_2\end{bmatrix}$, two joint angles for the two-link manipulator.
> > - `theta_dot` (dim. [2 × `nb_theta`]): An array where each column contains the time derivatives of the joint angles, $\dot{\theta}$, for each one of the corresponding columns in `theta`.
> >
> > Output arguments
> >
> > - `vertex_effector_dot` (dim. [2 × `nb_theta`]): An array where each column contains the time derivative of the end effector position, expressed in the world's frame, $\frac{\mathrm{d}}{\mathrm{d}t}({}^{\mathcal{W}}p_{\text{eff}})$, for the corresponding columns in the input arguments.

Question report 6.2. Assume $\theta(t)$ follow a line as defined in Questions report 5.3 and report 5.4. What is the corresponding value of $\frac{\mathrm{d}}{\mathrm{d}t}({}^{\mathcal{W}}p_{\text{eff}})$ for the following values of θ, $\left\{\begin{bmatrix}0\\0\end{bmatrix}, \begin{bmatrix}0\\\frac{\pi}{2}\end{bmatrix}, \begin{bmatrix}\pi\\\frac{\pi}{2}\end{bmatrix}, \begin{bmatrix}\pi\\\pi\end{bmatrix}\right\}$, and the following values of `a`, $\left\{\begin{bmatrix}1\\0\end{bmatrix}, \begin{bmatrix}0\\1\end{bmatrix}\right\}$; in total, you need to compute eight values for $\frac{\mathrm{d}}{\mathrm{d}t}({}^{\mathcal{W}}p_{\text{eff}})$. You can do the calculations by hand, or use the function `Twolink.jacobian(_)` from the previous question. In addition, use the function `Twolink.plot(_)` from Question provided 2.1 to plot the eight configurations, and use the `quiver(_)` function to plot $\frac{\mathrm{d}}{\mathrm{d}t}({}^{\mathcal{W}}p_{\text{eff}})$ as an arrow with its base at the end effector's position. Make comments on anything interesting that you might notice from the results; in particular, try to explain why for some configurations the end effector velocity might be zero despite the fact that the robot as a whole is moving.

Question report 6.3. In the following question we analyze the correspondence between the torus and the two-link manipulator from Problem 2.

File name: `me570_hw2.py`
Method name: `torus_twolink_plot_jacobian`
Description: For each one of the curves used in Question report 5.5, do the following:

- Use `Line.linspace(_)` to compute the array `thetaPoints` for the curve;
- For each one of the configurations given by the columns of `thetaPoints`:

  1) Use `Twolink.plot(_)` to plot the two-link manipulator.

  2) Use `Twolink.jacobian(_)` to compute the velocity of the end effector, and then use `quiver(_)` to draw that velocity as an arrow starting from the end effector's position.

The function should produce a total of four windows (or, alternatively, a single window with four subplots), each window (or subplot) showing all the configurations of the manipulator superimposed on each other. You can use `matplotlib.pyplot.ion(_)` and insert a `time.sleep(_)` command in the loop for drawing the manipulator, in order to obtain a "movie-like" presentation of the motion.
Requirements:
optional For each window (or subplot), use the color of the corresponding curve as used in Question report 5.5.

Question report 6.4 (0.5 points). Comment on the relation between the results of Questions report 6.3 and report 5.5. Instead of focusing of the individual curves, comment on how the two sets of results can be conceptually linked together.

Hint for question report 1.1: Consider what happens when you apply each rotation to the standard basis, i.e., to the vectors $e_x = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$, $e_y = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}$, $e_z = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$.

Hint for question report 1.1: Remember that a rotation, when applied to the vector $v \in \mathbb{R}^3$ representing its own rotation axis, it does not change it (i.e., $Rv = v$).

Hint for question optional 4.1: Think about the properties that the sets U_i must have.

Hint for question report 5.1: The answer to this question can be seen as a generalization of Question optional 4.1 from the 1-D to 2-D case.

Hint for question report 6.1: To derive the map, imagine that θ is a function of time, i.e. $\theta(t)$, and then compute the time derivative of ${}^{\mathcal{B}}p_{\text{eff}}$. The simplest way to derive this expression is by using the derivative of rotation matrices seen in class.

Question report 6.5 (2 points). Include the output of `Torus.plotChartsCurves()` in your report. Include also a second figure where you show that all the tangents are aligned along a common plane which is itself tangent to the surface of the torus. Explain how this relates to the material we covered in class.