```
import numpy as np #used to work with arrays and also has functions that work in the domains of linear al
import matplotlib.pyplot as plt #used to create data visualization.
from scipy.optimize import minimize #minimization of scalar function of one or more variables.
from ipywidgets import interact #automatically creates user interface (UI) controls for exploring code an
```

Digunaakn untuk import library yang berguna untuk visualisasi data, linear algebra, transformasi fourier, matriks, dll

```
"""
Planar quadrotor animation using `matplotlib`.
Author: Spencer M. Richards
        Autonomous Systems Lab (ASL), Stanford
        (GitHub: spenrich)
"""

import numpy as np
import matplotlib.pyplot as plt
import matplotlib.patches as mpatches #patches are Artists with a face color and an edge color.
import matplotlib.transforms as mtransforms
import matplotlib.animation as animation #a base class for Animations.


def animate_planar_quad(t, x, y, θ, title_string=None, display_in_notebook=False):
    """Animate the planar quadrotor system from given position data.
    All arguments are assumed to be 1-D NumPy arrays, where `x`, `y`, and `θ`
    are the degrees of freedom of the planar quadrotor over time `t`.
    Example usage:
        import matplotlib.pyplot as plt
        from animations import animate_planar_quad
        fig, ani = animate_planar_quad(t, x, θ)
        ani.save('planar_quad.mp4', writer='ffmpeg')
        plt.show()
    """
    # Geometry
    rod_width = 2.
    rod_height = 0.15
    axle_height = 0.2
    axle_width = 0.05
```

```
    prop_width = 0.5 * rod_width
    prop_height = 1.5 * rod_height
    hub_width = 0.3 * rod_width
    hub_height = 2.5 * rod_height

    # Figure and axis
    fig, ax = plt.subplots(dpi=100) #ax = axis, an axis is an area in the figure where the data will be plotted.
    x_min, x_max = np.min(x), np.max(x)
    x_pad = (rod_width + prop_width) / 2 + 0.1 * (x_max - x_min)
    y_min, y_max = np.min(y), np.max(y)
    y_pad = (rod_width + prop_width) / 2 + 0.1 * (y_max - y_min)
    ax.set_xlim([x_min - x_pad, x_max + x_pad])
    ax.set_ylim([y_min - y_pad, y_max + y_pad])
    ax.set_aspect(1.)
    if title_string is not None:
        plt.title(title_string)

    # Artists
    rod = mpatches.Rectangle((-rod_width / 2, -rod_height / 2),
                             rod_width,
                             rod_height,
                             facecolor='tab:blue',
                             edgecolor='k')
    hub = mpatches.FancyBboxPatch((-hub_width / 2, -hub_height / 2),
                                  hub_width,
                                  hub_height,
                                  facecolor='tab:blue',
                                  edgecolor='k',
                                  boxstyle='Round,pad=0.,rounding_size=0.05') #BoxStyle is a container class which defines
several boxstyle classes, which are used for FancyBboxPatch.
    axle_left = mpatches.Rectangle((-rod_width / 2, rod_height / 2),
                                   axle_width,
                                   axle_height,
                                   facecolor='tab:blue',
                                   edgecolor='k')
    axle_right = mpatches.Rectangle((rod_width / 2 - axle_width, rod_height / 2),
                                    axle_width,
                                    axle_height,
                                    facecolor='tab:blue',
                                    edgecolor='k')
```

```
prop_left = mpatches.Ellipse(((axle_width - rod_width) / 2, rod_height / 2 + axle_height),
                             prop_width,
                             prop_height,
                             facecolor='tab:gray',
                             edgecolor='k',
                             alpha=0.7)
prop_right = mpatches.Ellipse(((rod_width - axle_width) / 2, rod_height / 2 + axle_height),
                              prop_width,
                              prop_height,
                              facecolor='tab:gray',
                              edgecolor='k',
                              alpha=0.7)
patches = (rod, hub, axle_left, axle_right, prop_left, prop_right)
for patch in patches:
    ax.add_patch(patch)
trace = ax.plot([], [], '--', linewidth=2, color='tab:orange')[0]
timestamp = ax.text(0.1, 0.9, '', transform=ax.transAxes)

def animate(k, t, x, y, θ):
    transform = mtransforms.Affine2D().rotate_around(0., 0., θ[k])
    transform += mtransforms.Affine2D().translate(x[k], y[k])
    transform += ax.transData
    for patch in patches:
        patch.set_transform(transform)
    trace.set_data(x[:k + 1], y[:k + 1])
    timestamp.set_text('t = {:.1f} s'.format(t[k]))
    artists = patches + (trace, timestamp)
    return artists

dt = t[1] - t[0]
step = max(int(np.floor((1 / 30) / dt)), 1)  # max out at 30Hz for faster rendering
ani = animation.FuncAnimation(fig,
                              animate,
                              t[::step].size,
                              fargs=(t[::step], x[::step], y[::step], θ[::step]),
                              interval=step * dt * 1000,
                              blit=True)
if display_in_notebook:
    try:
        get_ipython()
```

```
        from IPython.display import HTML
        ani = HTML(ani.to_html5_video())
    except (NameError, ImportError):
        raise RuntimeError("`display_in_notebook = True` requires this code to be run in jupyter/colab.")
plt.close(fig)
return ani
```

code diatas berguna untuk Kode tersebut merupakan sebuah animasi quadrotor planar menggunakan library matplotlib. Quadrotor planar merupakan sebuah sistem yang memiliki tiga derajat kebebasan: x, y, dan θ (sudut). Kode tersebut digunakan untuk membuat animasi dari posisi quadrotor planar berdasarkan data yang diberikan.

Beberapa komponen visual seperti ukuran dan warna dari bagian quadrotor diatur terlebih dahulu, seperti lebar dan tinggi dari batang, poros, dan baling-baling. Selain itu, warna dan jenis patch dari setiap bagian juga diatur.

Setelah itu, sebuah objek figure dan axis dibuat untuk menentukan area plot dan batas-batas sumbu x dan y pada plot. Lalu, setiap bagian quadrotor ditambahkan pada plot menggunakan method add_patch. Sebuah line plot juga ditambahkan untuk menunjukkan jejak pergerakan quadrotor dan sebuah teks untuk menunjukkan waktu pada animasi.

Untuk melakukan animasi, sebuah fungsi animate dibuat yang mengubah posisi dari setiap bagian quadrotor dan jejak pergerakan pada setiap frame animasi. Fungsi animate kemudian digunakan pada objek animation.FuncAnimation untuk membuat animasi berdasarkan data yang diberikan.

Pada akhirnya, jika pengguna ingin menampilkan animasi pada notebook, kode tersebut juga menyediakan opsi untuk menampilkan animasi tersebut sebagai video HTML menggunakan IPython.display.

```
class PlanarQuadrotorDynamics:

    def __init__(self):
        # Dynamics constants
        # yapf: disable
        self.g = 9.807        # gravity (m / s**2)
        self.m = 2.5          # mass (kg)
        self.l = 1.0          # half-length (m)
        self.I_zz = 1.0       # moment of inertia about the out-of-plane axis (kg * m**2)
        # yapf: enable

    def __call__(self, state, control):
        """Continuous-time dynamics of a planar quadrotor expressed as an ODE."""
```

```
        x, v_x, y, v_y, φ, ω = state
        T_1, T_2 = control
        return np.array([
            v_x,
            -(T_1 + T_2) * np.sin(φ) / self.m,
            v_y,
            (T_1 + T_2) * np.cos(φ) / self.m - self.g,
            ω,
            (T_2 - T_1) * self.l / self.I_zz,
        ])
dynamics = PlanarQuadrotorDynamics()
state_0 = np.array([4., 2., 6., 2., -np.pi / 4, -1.])
state_f = np.zeros(6)
t_f = 3  # fixed final time
```

code diatas adalah Ini adalah sebuah kelas yang merepresentasikan dinamika dari sebuah planar quadrotor (quadrotor yang bergerak di bidang datar). Kelas ini memiliki atribut-atribut konstanta seperti gravitasi, massa, panjang setengah, dan moment of inertia tentang sumbu yang tegak lurus terhadap bidang datar.

Fungsi utama kelas ini adalah __call__, yang merepresentasikan dinamika planar quadrotor secara kontinu dalam bentuk persamaan diferensial biasa (ODE). Fungsi ini mengambil dua parameter yaitu state dan control. Parameter state adalah kondisi saat ini dari quadrotor, yaitu posisi (x, y), kecepatan (v_x, v_y), sudut (φ), dan kecepatan sudut (ω). Parameter control adalah aksi kontrol yang diberikan pada quadrotor, yaitu gaya dorong (T_1, T_2) dari masing-masing rotor.

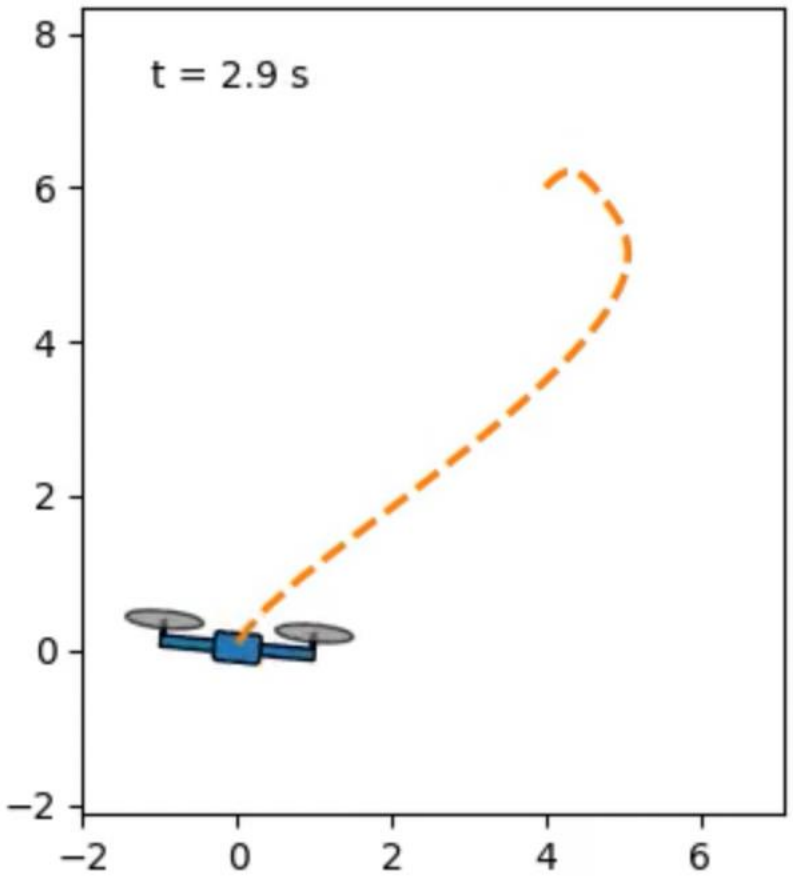
Fungsi __call__ akan mengembalikan array numpy yang merepresentasikan turunan dari state, yaitu kecepatan dan percepatan untuk setiap komponen state. Dalam kode ini, persamaan diferensial biasa dihitung menggunakan hukum Newton dan kinematika dasar.

```
N = 60
dt = t_f / N

cost = lambda z: dt * np.sum(np.square(z.reshape(N + 1, 8)[:, -2:])) #lambda - expression to create a function.


def constraints(z):
    states_and_controls = z.reshape(N + 1, 8)
    states = states_and_controls[:, :6]
    controls = states_and_controls[:, -2:]
    constraint_list = [states[0] - state_0, states[-1] - state_f]
```

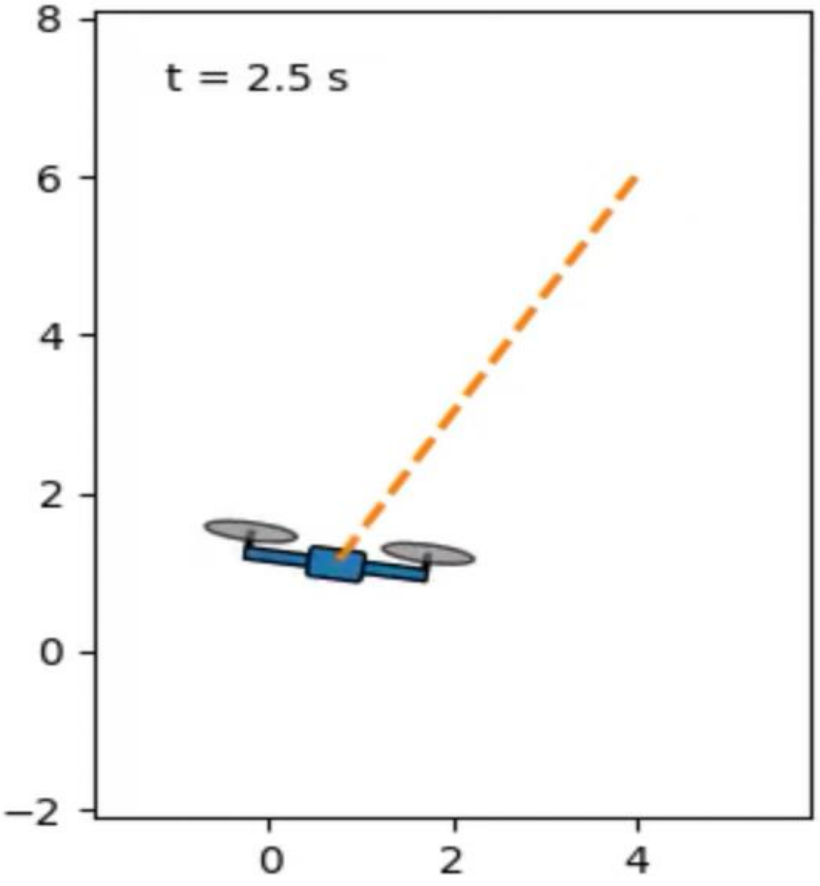
```
    for i in range(N):
        constraint_list.append(states[i + 1] - (states[i] + dt * dynamics(states[i], controls[i])))
    return np.concatenate(constraint_list)


z_guess = np.concatenate([np.linspace(state_0, state_f, N + 1), np.ones((N + 1, 2))], -1).ravel()
z_iterates = [z_guess]
result = minimize(cost,
                  z_guess,
                  constraints={
                      'type': 'eq',
                      'fun': constraints
                  },
                  options={'maxiter': 1000},
                  callback=lambda z: z_iterates.append(z))
z_iterates = np.stack(z_iterates)
result
```

Code di atas merupakan implementasi dari metode shooting untuk menyelesaikan permasalahan optimal control. Tujuannya adalah untuk mencari solusi terbaik untuk mengendalikan sistem dinamik dari suatu keadaan awal ke keadaan akhir yang diinginkan dengan meminimalkan suatu kriteria kinerja tertentu.Pada kode tersebut, terdapat variabel N yang menentukan jumlah iterasi yang akan dilakukan. Variabel dt menentukan ukuran langkah waktu yang akan digunakan dalam proses iterasi. Kemudian, terdapat definisi fungsi cost yang merupakan fungsi kriteria kinerja yang ingin diminimalkan. Fungsi constraints merupakan fungsi yang menghitung kendala-kendala yang harus dipenuhi oleh solusi. Variabel z_guess merupakan tebakan awal dari solusi yang dicari, yang kemudian akan dioptimalkan menggunakan metode minimize dari modul scipy. Constraints yang harus dipenuhi ditentukan sebagai persamaan, yang diwakili oleh tipe 'eq'. Parameter maxiter menentukan jumlah maksimum iterasi yang akan dilakukan dalam proses optimisasi. Terakhir, callback digunakan untuk mencatat setiap iterasi dari proses optimisasi, sehingga dapat digunakan untuk menganalisis konvergensi solusi.

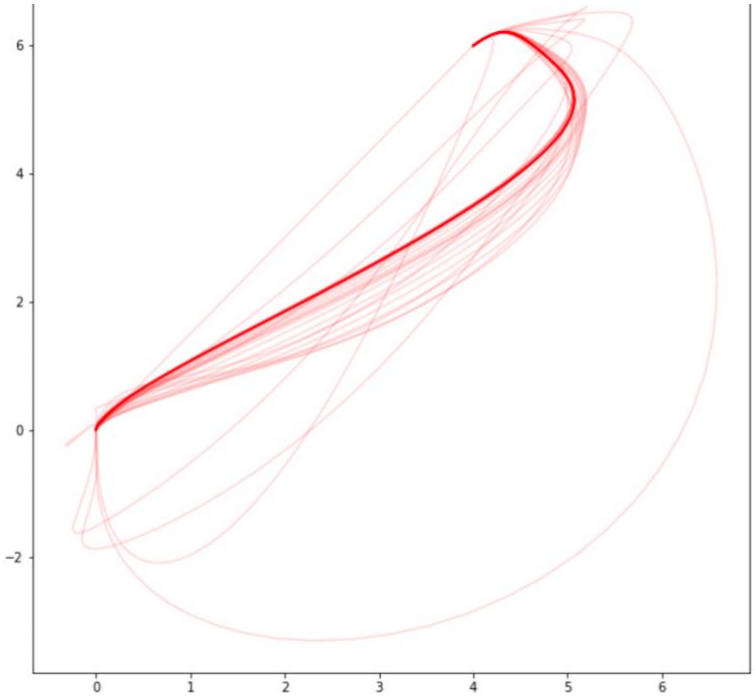
```
t = np.linspace(0, t_f, N + 1) #linspace, to create a set of numbers evenly spaced in a specified interval.
z = result.x.reshape(N + 1, 8)
animate_planar_quad(t, z[:, 0], z[:, 2], z[:, 4], display_in_notebook=True)
```




```
t = np.linspace(0, t_f, N + 1)
@interact(iteration=(0, len(z_iterates) - 1))
def f(iteration=0):
    z = z_iterates[iteration].reshape(N + 1, 8)
    return animate_planar_quad(t, z[:, 0], z[:, 2], z[:, 4], display_in_notebook=True)
```




```
states_and_controls_iterates = z_iterates.reshape(-1, N + 1, 8)
plt.figure(figsize=(10, 10))
plt.plot(states_and_controls_iterates[:, :, 0].T, states_and_controls_iterates[:, :, 2].T, color='red', alpha=0.15);
#alpha - level of clarity.
```




```
from numpy.polynomial import Polynomial #Polynomials in NumPy can be created, manipulated, and even fitted using the convenience classes.
def get_flat_output_derivatives(state, ydd=0, yddd=0, g=dynamics.g):
    x, xd, y, yd, φ, ω = state
    xdd = -(ydd + g) * np.tan(φ)
    xddd = -(ω * ((ydd + g)**2 + xdd**2) - yddd * xdd) / (ydd + g)
    return np.array([x, xd, xdd, xddd]), np.array([y, yd, ydd, yddd])

basis_functions = [Polynomial.basis(i) for i in range(8)]
basis_matrix = np.array([[ψ.deriv(d)(0) for ψ in basis_functions] for d in range(4)] +
                        [[ψ.deriv(d)(t_f) for ψ in basis_functions] for d in range(4)])

#np.linalg.solve - solve a linear matrix equation, or system of linear scalar equations.
x_poly = Polynomial(
    np.linalg.solve(basis_matrix,
                    np.concatenate([get_flat_output_derivatives(state_0)[0],
                                    get_flat_output_derivatives(state_f)[0]])))
y_poly = Polynomial(
    np.linalg.solve(basis_matrix,
                    np.concatenate([get_flat_output_derivatives(state_0)[1],
```

```
                        get_flat_output_derivatives(state_f)[1]])))


x_poly
y_poly
t = np.linspace(0, t_f, 30 * t_f)
animate_planar_quad(t,
                    x_poly(t),
                    y_poly(t),
                    np.arctan2(-x_poly.deriv(2)(t),
                               y_poly.deriv(2)(t) + dynamics.g),
                    display_in_notebook=True)
```

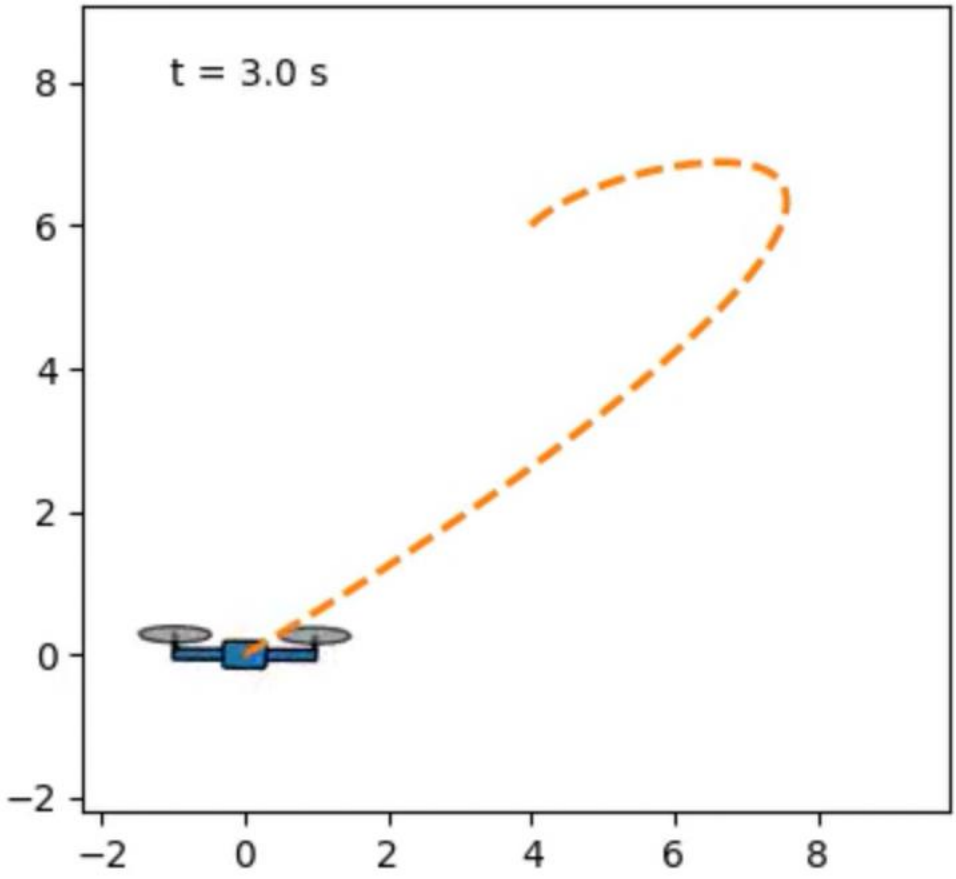
Code ini digunakan untuk menghasilkan polinomial yang merepresentasikan jalur gerakan quadrotor. Polinomial ini nantinya akan digunakan dalam fungsi animate_planar_quad untuk memvisualisasikan gerakan quadrotor.

Pertama, kita definisikan fungsi get_flat_output_derivatives yang menerima state (x, y, dan orientasi) dan turunannya ydd dan yddd. Fungsi ini akan mengembalikan turunan dari state flat output yang kemudian akan digunakan untuk membuat polinomial.

Selanjutnya, kita membuat basis function dengan Polynomial.basis dan basis matrix yang berisi turunan dari basis function pada t=0 dan t=t_f. Basis matrix ini kemudian digunakan dalam fungsi np.linalg.solve untuk memperoleh koefisien dari polinomial.

Setelah itu, kita menggunakan Polynomial dari NumPy untuk membuat polinomial x dan y dari koefisien yang diperoleh dari np.linalg.solve. Polinomial x dan y ini akan merepresentasikan jalur gerakan quadrotor.

Terakhir, kita menghasilkan visualisasi gerakan quadrotor dengan menggunakan fungsi animate_planar_quad dengan memberikan polinomial x dan y yang telah dibuat sebelumnya.


t = 3.0 s
8
6
4
2
0
−2
−2
0
2
4
6
8