PA3: Pick-and-Place Controller

By: Mason Harris

Since the goal of this assignment is to produce an animation demonstrating a controller, it was chosen to time-align all calculations to sync up with a “real-time” demonstration in the animation. A clock of 60Hz lines up with a 60 FPS animation and provides fine enough resolution for proper calculations in the controller.

```
26  # define how many calculations per second to perform to achieve proper animation alignment
27  clock = 60
28  period = 1/clock
29
30  # time per movement in seconds, movement being home --> pick and pick --> place
31  t_move = 5
32
33  # frames to evaluate per movement
34  frames = int(t_move * clock)
35
```

The controller uses a PD design by considering the current error of the joint and current velocity of the joint to calculate an applied torque. This torque is calculated using the following formula:

$$torque_{applied} = k_p(\theta_{desired} - \theta_{current}) - k_v * \omega_{current}$$

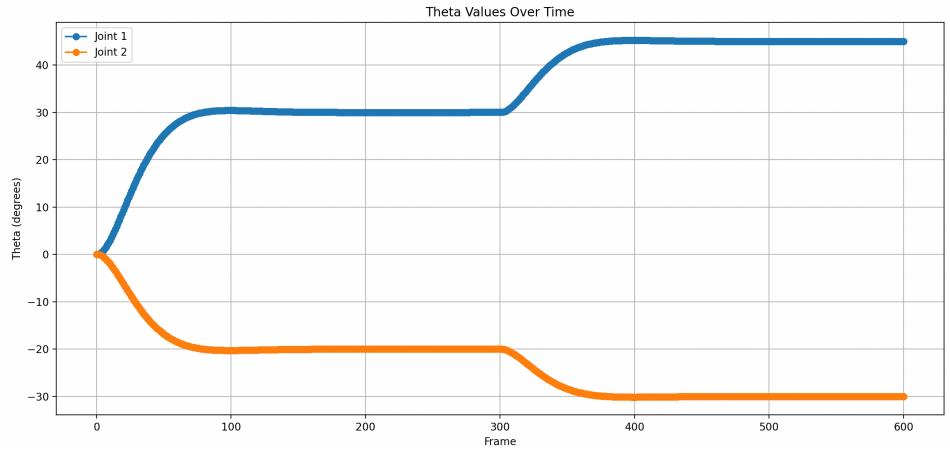
The applied torque is then applied to the current rotational velocity of the joint. It is important to note that this simulation does not assign a real value for the inertia experienced by both links. This is unrealistic and as a result the torque values are fully arbitrary. The values of kp and kv are adjusted to tune the controller.

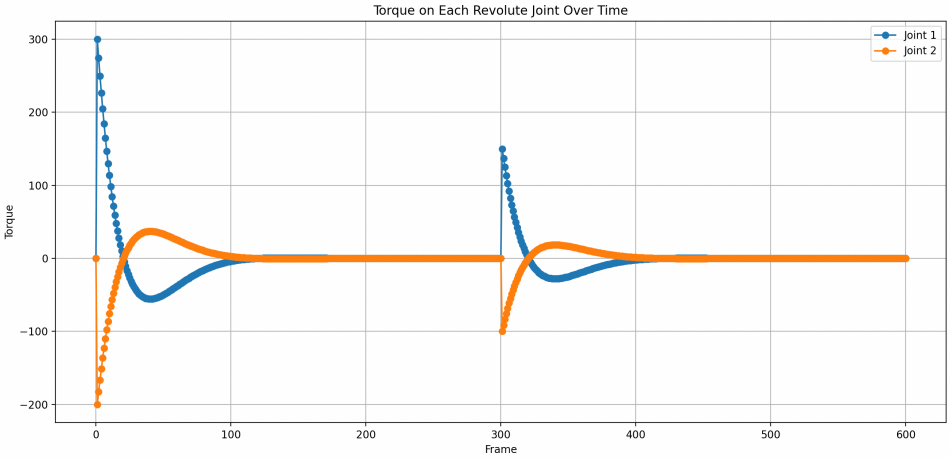
```
41  # increase kp for a stronger "spring"
42  kp = 10
43  # increase kv for a stronger "damper"
44  kv = 5
45
46  # kp and kv are global variables defined above
47  # inertia is arbitrary and set to 1 in the controller
48  # so it isn't super realistic for a 2R manipulator
49
50  def controller(current_w, current_theta, desired_theta):
51      # calculates error from desired theta
52      error = desired_theta - current_theta
53      # torque as a function of error and current rotational velocity
54      torque = kp*(error) - kv*current_w
55      # calculates rotational acceleration over period assuming an interia of 1
56      # new rotational velocity calculated at end of period
57      next_w = current_w + torque * period
58      return next_w, torque
59
```

Since the number of iterations through the controller is set by the clock and the time allowed for each movement, the matter of simulating the mechanics of each joint is as simple as iterating through each frame and keeping track of each updated joint variable per frame.

```
66  # calculating home --> pick thetas
67  for n in range(frames):
68      current_theta1 += period*w1
69      joint1.append(current_theta1)
70      w1, torque = controller(w1, current_theta1, pick_theta[0])
71      torques1.append(torque)
72      current_theta2 += period*w2
73      joint2.append(current_theta2)
74      w2, torque = controller(w2, current_theta2, pick_theta[1])
75      torques2.append(torque)
```

These values are used to create the following plots and are utilized in the creation of the animation. Notice that the torque is highest at the beginning of each movement. This is due to the error being the highest and the velocity the lowest.





The steady-state error can be analyzed by zooming in the calculated error over time. The y-axis shows that the steady-state error is well within one degree of error.