In [ ]:

```python
from motor_control.AROMotorControl import AROMotorControl
mc = AROMotorControl()
mc.readPosition(motorid=2)
```

Out[ ]: 199.11000000000058

In [ ]:

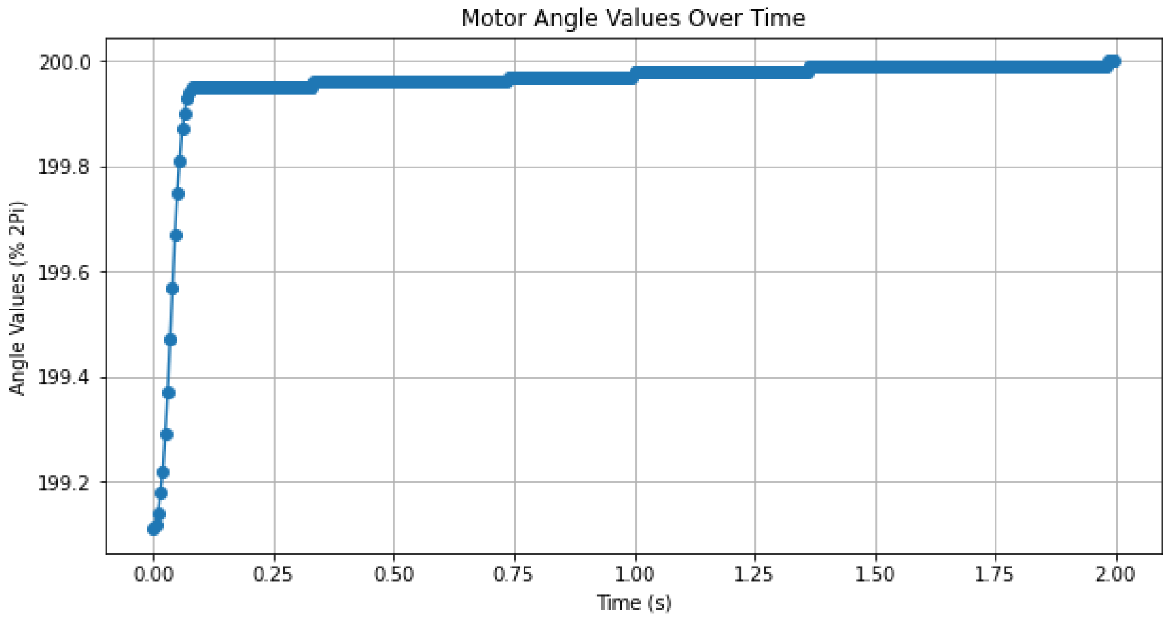
```python
from template import run_until
dt = 0.005
# Calculate how many iterations the motor should run
# based on a 2-second duration and the time step.
N = int(2. / dt)
try:
    # runs motor1 at 0.02 torque for 2 seconds
    run_until(mc.applyTorqueToMotor, N=N, dt=0.005, motorid=1, torque=0.0
except KeyboardInterrupt:
    print("KeyboardInterrupt received, stopping motors...")
except Exception as e:
    print(f"an error occurred: {e}")
finally:
    mc.applyTorqueToMotor(1, 0) # stop the motor
    mc.applyTorqueToMotor(2, 0)
    # applyTorqueToMotor takes the arguments:
    # motorid: the motor to apply the torque to
    # torque: the torque to apply to the motor
    print("motors stopped!")
```

```
motors stopped!
```

In [ ]:

```python
import matplotlib.pyplot as plt
anglevalues =[]
def store_values_and_apply_torques(motorid, torque):
    global anglevalues
    mc.applyTorqueToMotor(motorid=motorid, torque=torque)
    anglevalues.append(mc.readPosition(motorid=motorid))

try:
    run_until(store_values_and_apply_torques, N=N, dt=0.005, motorid=2, t
except KeyboardInterrupt:
    print("KeyboardInterrupt received, stopping motors...")
except Exception as e:
    print(f"an error occurred: {e}")
finally:
    mc.applyTorqueToMotor(1, 0)
    mc.applyTorqueToMotor(2, 0)
    print("motors stopped!")


time_values = [i * dt for i in range(len(anglevalues))]

# Plotting the angle values
plt.figure(figsize=(10, 5))
plt.plot(time_values, anglevalues, marker='o', linestyle='-')
plt.title('Motor Angle Values Over Time')
plt.xlabel('Time (s)')
plt.ylabel('Angle Values (% 2Pi)')
plt.grid()
plt.show()
```

```
motors stopped!
```



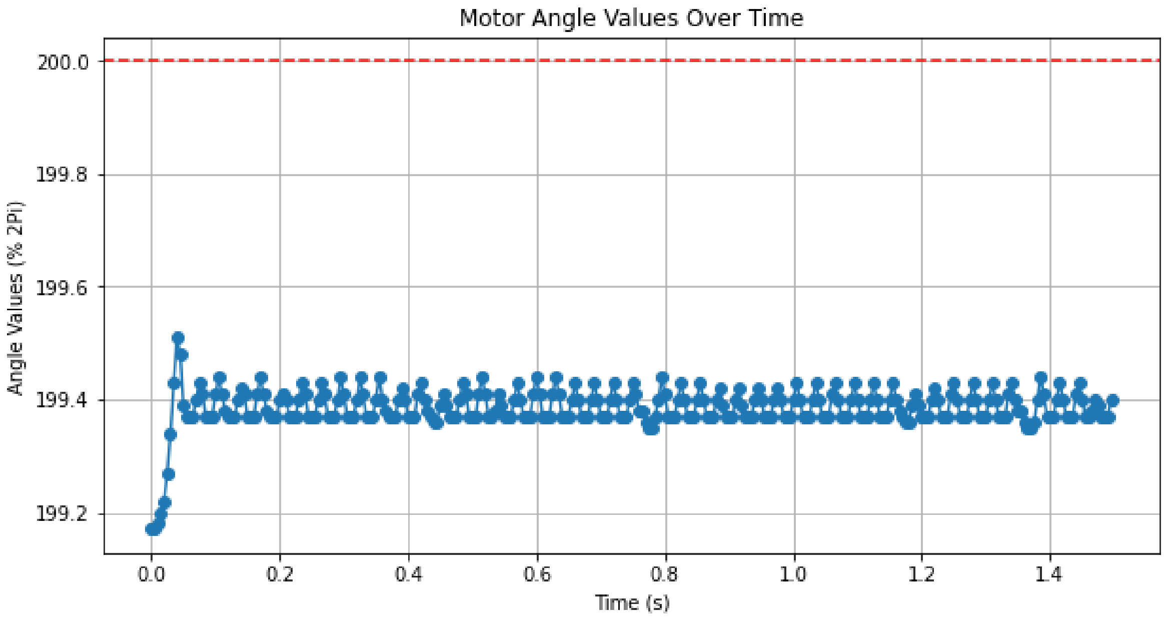
In [ ]:

```
def clip(output):
    """This function clips the output to a range of -0.1 to -0.02 and 0.0
    this is to ensure that, if there is a lot of friction, a minimal sign
    when the error is not 0 but small, as the motors tend not ro move at
    before 0.02 nM of torque is applied.
    """
    outabs = abs(output)
    if outabs < 1e-4:
        return 0
    clipped = max(min(outabs, 0.1), 0.02)
    return clipped if output > 0 else -clipped

class PController:
    def __init__(self, Kp, Kd):
        self.Kp = Kp
        self.Kd = Kd
        self.elapsed = 0
        self.prev_error = 0

    def reset(self):
        self.elapsed = 0

    def shortest_path_error(self, target, current):
        # difference between target and current
        # you sum 180 to the difference and then take the modulo 360 to g
        # then you subtract 180 to get the difference between the target
        diff = ( target - current + 180 ) % 360 - 180
        # e.g. target = 360, current = 0, diff = (360 - 0 + 180) % 360 -
        # e.g. target = 180, current = 90, diff = (180 - 90 + 180) % 360
        # e.g. target = 90, current = 180, diff = (90 - 180 + 180) % 360
        # e.g. target = 0, current = 360, diff = (0 - 360 + 180) % 360 -
            # in this case, the next line will sum 360 to the diff and re

        if diff < -180:
            diff = diff + 360
        if (current + diff) % 360 == target:
            # if the target is reached, return 0 (the difference)
            return diff
        else:
            # if the target is not reached, return the negative differenc
```

```
            return -diff

    def compute(self, target, current, dt):
        error = self.shortest_path_error(target, current)
        d_error = (error - self.prev_error) if self.elapsed >= dt else 0
        output = self.Kp*error  + self.Kd*d_error
        # Kp is the proportional gain.
        # It is a constant that determines how much the output will chang
        # the error. If Kp is too high, the system will be unstable,
        # if it is too low, the system will be slow.
        if (self.elapsed < 2*dt):
            output = clip(output)
        else:
            output = clip(output)
        self.elapsed += dt
        return output
```

In [ ]:

```
def goTo(controller, target, time = 1., dt = 0.005, motorid =1):
    anglevalues =[]
    N = (int)(time / dt) # number of iterations

    def oneStep():
        nonlocal anglevalues
        currentAngle = mc.readPosition(motorid)
        anglevalues+=[currentAngle]
        tau = controller.compute(target,currentAngle, dt) # returns the t

        mc.applyTorqueToMotor(motorid,tau)

    controller.reset()
    run_until(oneStep, N=N, dt=dt)

    mc.applyTorqueToMotor(1, 0)
    mc.applyTorqueToMotor(2, 0)

    time_values = [i * dt for i in range(len(anglevalues))]
    # Plotting the angle values
    plt.figure(figsize=(10, 5))
    plt.plot(time_values, anglevalues, marker='o', linestyle='-')
    plt.axhline(y=target, color='r', linestyle='--', label='Target Value'
    plt.title('Motor Angle Values Over Time')
    plt.xlabel('Time (s)')
    plt.ylabel('Angle Values (% 2Pi)')
    plt.grid()
    plt.show()

try:
    pc = PController(0.00016, 0.000)
    goTo(pc, 200, time = 1.5, motorid=2)
except KeyboardInterrupt:
    print("KeyboardInterrupt received, stopping motors...")
except Exception as e:
    print(f"an error occurred: {e}")
finally:
    mc.applyTorqueToMotor(1, 0)
    mc.applyTorqueToMotor(2, 0)
    print("motors stopped!")
```




```
motors stopped!
```

In [ ]:
```python
print(mc.readPosition(2))
```

```
199.23999999999978
```

In [ ]:
```python
"""Now, write a 30s control loop that does the following: + Motor 1 is co
of motor 2 + Motor 2 is configured to track the position of motor 1
Manually mess around with the motors while the loop is running and check
you expect. A similar system is implemented in some of the recent cars wh
the wheels are no longer mechanically connected (see https://en.wikipedia
dt = 0.005
N = (int)(30. / dt) # number of iterations
pc1 = PController(0.00016, 0.000)
pc2 = PController(0.00016, 0.000)
def controlLoop():

    def oneStep():
        angle1 = mc.readPosition(1)
        angle2 = mc.readPosition(2)
        tau1 = pc1.compute(angle2, angle1, dt)
        tau2 = pc2.compute(angle1, angle2, dt)
        mc.applyTorqueToMotor(1, tau1)
        mc.applyTorqueToMotor(2, tau2)
    run_until(oneStep, N=N, dt=dt)

    mc.applyTorqueToMotor(1, 0)
    mc.applyTorqueToMotor(2, 0)


controlLoop()
```

In [ ]:
```python
mc.applyTorqueToMotor(1, 0)
mc.applyTorqueToMotor(2, 0)
```

Out[ ]: (28, 0, 0, 147)