Lab4_20

Trajectory Generation

```
def quintic_trajectory(self,T,dt):
    p_init = 0
    p_final = 1
    a0 = p_init
    a1 = 0
    a2 = 0*0.5
    a3 = -(20*p_init - 20*p_final)/(2*(T**3))
    a4 = (30*p_init - 30*p_final)/(2*(T**4))
    a5 = -(12*p_init - 12*p_final)/(2*(T**5))

    if(dt<=T):
        ref_p = (a0)+(a1*dt)+(a2*dt**2)+(a3*dt**3)+(a4*dt**4)+(a5*dt**5)
        ref_p_dot = (a1)+(2*a2*dt)+(3*a3*dt**2)+(4*a4*dt**3)+(5*a5*dt**4)
    else:

        ref_p = (a0)+(a1*T)+(a2*T**2)+(a3*T**3)+(a4*T**4)+(a5*T**5)
        ref_p_dot = 0
    return ref_p , ref_p_dot
```

```
def timer_callback(self):
    if(np.array_equal(self.p_init, self.p_init)!=True):
        self.t = 0.
    #Create p and p_dot reference
    ref_p , ref_p_dot = self.quintic_trajectory(self.T,self.t)
    self.t += self.dt
    #Recieve from sub via_point_callback and change to np array
    p_init = np.array(self.p_init)
    p_final = np.array(self.p_final)

    #Linear Interpolation
    p_r = (1-ref_p)*p_init + ref_p * (p_final)
    p_r_dot = ref_p_dot * (p_final-p_init)

    print(p_r)
    #Inverse position
    Gamma = [1,-1]
    self.q_r = IPK(Gamma[0],Gamma[1],p_r[0],p_r[1],p_r[2])
    #Inverse Velocity
    self.q_r_dot = IVK(self.q_r[0],p_r_dot)
    #pub to tracker to PID
    pub_msg = JointState()
    pub_msg.position = list(self.q_r[0])
    pub_msg.velocity = list(self.q_r_dot)

    self.pub.publish(pub_msg)
```

การสร้าง Trajectory โดยใช้ Quintic Polynomial ในกสารสร้าง Trajectory โดยมี Parameter ดังนี้

โดยที่

Θ_0 คือ จุดเริ่มต้นของ Trajectory

$\dot{\Theta}_0$ คือ ความเร็วเริ่มต้นของ Trajectory

$\ddot{\Theta}_0$ คือ ความเร่งเริ่มต้นซอง Trajectory

$$a_0 = \Theta_0, \quad a_1 = \dot{\Theta}_0, \quad a_2 = \frac{\ddot{\Theta}_0}{2},$$

$$a_3 = \frac{20\Theta_f - 20\Theta_0 - (8\dot{\Theta}_f + 12\dot{\Theta}_0)t_f - (3\ddot{\Theta}_0 - \ddot{\Theta}_f)t_f^2}{2t_f^3},$$

$$a_4 = \frac{30\Theta_0 - 30\Theta_f + (14\dot{\Theta}_f + 16\dot{\Theta}_0)t_f + (3\ddot{\Theta}_0 - 2\ddot{\Theta}_f)t_f^2}{2t_f^4},$$

$$a_5 = \frac{12\Theta_f - 12\Theta_0 - (6\dot{\Theta}_f + 6\dot{\Theta}_0)t_f - (\ddot{\Theta}_0 - \ddot{\Theta}_f)t_f^2}{2t_f^5}.$$

โดยที่จาก Parameter ด้านบนเราจะสามารถหาตำแหน่ง และความเร็ว ณ เวลาใดๆ ได้ดังนี้

$$\theta(t) = a_0 + a_1 t + a_2 t^2 + a_3 t^3 + a_4 t^4 + a_5 t^5$$

$$\dot{\theta}(t) = a_1 + 2a_2 t + 3a_3 t^2 + 4a_4 t^3 + 5a_5 t^4$$

Linear Interpolation

```
def timer_callback(self):
    if(np.array_equal(self.p_init, self.p_init)!=True):
        self.t = 0.
    #Create p and p_dot reference
    ref_p , ref_p_dot = self.quintic_trajectory(self.T,self.t)
    self.t += self.dt
    #Recieve from sub via_point_callback and change to np array
    p_init = np.array(self.p_init)
    p_final = np.array(self.p_final)

    #Linear Interpolation
    p_r = (1-ref_p)*p_init + ref_p * (p_final)
    p_r_dot = ref_p_dot * (p_final-p_init)

    print(p_r)
    #Inverse position
    Gamma = [1,-1]
    self.q_r = IPK(Gamma[0],Gamma[1],p_r[0],p_r[1],p_r[2])
    #Inverse Velocity
    self.q_r_dot = IVK(self.q_r[0],p_r_dot)
    #pub to tracker to PID
    pub_msg = JointState()
    pub_msg.position = list(self.q_r[0])
    pub_msg.velocity = list(self.q_r_dot)

    self.pub.publish(pub_msg)
```

การแปลงค่าให้อยู่ในช่วงเส้นตรงระหว่างจุดสองจุดซึ่งสามารถทำได้จากสมการดังนี้

$$\mathbf{p}_\mathrm{r}(t) = \big(1 - \alpha(t)\big) \cdot \mathbf{p}_\mathrm{i} + \big(\alpha(t)\big) \cdot \mathbf{p}_\mathrm{f}$$
$$\dot{\mathbf{p}}_\mathrm{r}(t) = \dot{\alpha}(t) \cdot (\mathbf{p}_\mathrm{f} - \mathbf{p}_\mathrm{i})$$

Linear Inverse Kinematic

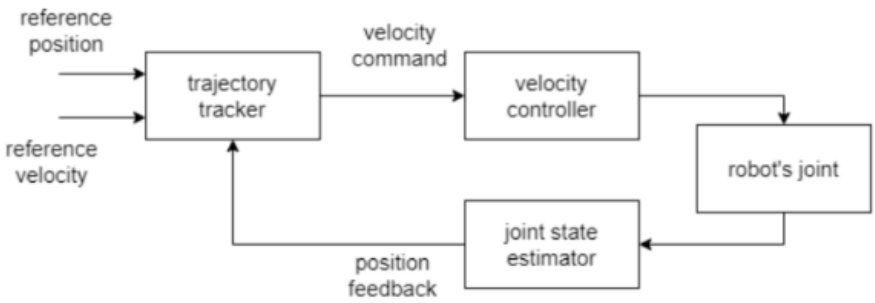
เนื่องจากค่าที่ได้จาก Trajectory Generator และ Linear Interpolation เป็นค่าที่อยู่ใน Task space ในการควบคุมหุ่น เราจำเป็นที่จะต้องควบคุมโดยใช้ตำแหน่งภายใน Joint Space โดยการใช้ Inverse Kinematic จากสมการดังนี้

$$\mathbf{q}_\mathrm{r}(t) = \mathbf{IPK}\big(\mathbf{p}_\mathrm{r}(t)\big)$$
$$\dot{\mathbf{q}}_\mathrm{r}(t) = \mathbf{IVK}\big(\mathbf{q}_\mathrm{r}(t), \dot{\mathbf{p}}_\mathrm{r}(t)\big)$$

การทำ Velocity Controller

```
def timer_callback(self):

    if(self.enableTracker):
        self.sum_error_q += (self.q_ref - self.q)
        velo_command = list(self.q_ref_dot + (self.Kp * (self.q_ref - self.q)) + self.Ki * self.sum_error_q)
        print(velo_command)
        command = Float64MultiArray()
        command.data = velo_command
        self.pub.publish(command)
    else:
        command = Float64MultiArray()
        command.data = [0.,0.,0.]
        print(command.data)
        self.pub.publish(command)
```



$$\pi(t, q) = \dot{q}_\mathrm{r}(t) + K_p \cdot (q_\mathrm{r}(t) - q) + K_i \cdot \int_{\tau=0}^{t} (q_\mathrm{r}(\tau) - q)\ \mathrm{d}\tau$$

การรับค่า Reference ที่ได้จาก Trajectory Generator เพื่อนำมาเปรียบเทียบกับ Position ที่ได้รับค่าจริงจาก Gazebo เพื่อใช้สำหรับการทำ Velocity Control ของหุ่นโดยที่จะมี Feedback ที่ได้รับคือ Position และส่ง ค่า Command Velocity ไปยัง Joint ของหุ่นยนต์ในการควบคุม

Scheduler

```
class Scheduler(Node):
    def __init__(self):
        super().__init__('Scheduler')
        #Pub to trajectory
        self.Pi_pub = self.create_publisher(Float64MultiArray,"/Pi" , 10)
        self.Pf_pub = self.create_publisher(Float64MultiArray,"/Pf" , 10)
        self.T_pub = self.create_publisher(Float64,"/T" , 10)

        #Subfrom Proximity
        self.hasReached_sub = self.create_subscription(Bool,"/hasReached" ,self.hasReached_callback,10)
        # #service EnableTracker
        self.enableTracker_Cli = self.create_client(Enabletracker, '/enableTracker')
        self.enableTracker_req = Enabletracker.Request()

        self.enableTracker_pub = self.create_publisher(Bool, '/enableTracker',10)

        self.Hi = self.create_publisher(Float64MultiArray, '/HI',10)
        #modify this
        self.hasReached = False
        self.K = 1
        self.current_p = [0.31,0.,0.135]
        self.next_p = [0.,0.,0.]
        self.T = 4
        # x - 0.15  y - 0.05 z = 0.134999

        with open(r'lab4_robot_control/config/via_point.yaml') as file:
            via_point = yaml.load(file, Loader=yaml.FullLoader)
        self.via_points = via_point['via_point']
        self.N = len(self.via_points)

        if(self.K<self.N):
            self.enableTracker_req.tracker_enable.data = True
            self.enableTracker_Cli.call_async(self.enableTracker_req)
        else:
            self.enableTracker_req.tracker_enable.data = False
            self.enableTracker_Cli.call_async(self.enableTracker_req)
        self.S()
```

```python
def S(self):
    if(self.K<self.N):
        pub_msg = Float64MultiArray()
        # self.current_p = self.via_points['via_points'][self.K-1]
        # self.next_p = self.via_points['via_points'][self.K]
        self.current_p = self.via_points[self.K-1]
        self.next_p = self.via_points[self.K]
        pub_msg.data = list(self.current_p)
        self.Pi_pub.publish(pub_msg)

        pub_msg.data = list(self.next_p)
        self.Pf_pub.publish(pub_msg)

        pub_msg = Float64()
        pub_msg.data = float(self.T)
        self.T_pub.publish(pub_msg)

        a = Bool()
        a.data = True
        print(self.K)
        pub_msg = Float64MultiArray()
        pub_msg.data = list(self.next_p)
        self.Hi.publish(pub_msg)
        self.enableTracker_pub.publish(a)
        self.K +=1
    elif(self.K>=self.N and self.enableTracker_req.tracker_enable.data == True):
        self.enableTracker_req.tracker_enable.data = False
        self.enableTracker_Cli.call_async(self.enableTracker_req)
```

ทำหน้าที่อ่านไฟล์ yaml สำหรับการระบุตำแหน่งเป้าหมายในการเคลื่อนที่ของหุ่นยนต์ โดยที่จะส่งค่าของตำแหน่งเริ่มต้นและจุดสุดท้ายในการเคลื่อนที่ไปยัง Trajectory Generator ในการควบคุมหุ่นโดยจะมีกาเปิดปิด ตัว Tracker ที่ทำหน้าที่สำหรับการเป็น Velocity Controller โดยที่จะมีค่า Boolean enableTracker สำหรับการเปิดและปิดตัว Tracker และจะรับค่า Boolean hasReached ซึ่งเป็นการบอกถึงว่าแขนหุ่นยนต์เคลื่อนที่ถึงเป้าหมายแล้วหรือยัง หาก Endeffector ยังเคลื่อนที่ไปถึงจุดหมาย Scheduler จึงจะทำการส่งตำแหน่งถัดไปให้ยัง Trajectory generator ต่อไป

Proximity Detector

```python
class proximity_detector(Node):
    def __init__(self):
        super().__init__("proximity_detector")
        self.q_sub = self.create_subscription(JointState,"/joint_states" ,self.q_callback, 10)
        self.p_f_sub = self.create_subscription(Float64MultiArray,"/HI" ,self.p_f_sub_callback, 10)
        self.hasReached_pub = self.create_publisher(Bool,"/hasReached", 10)
        self.timer = self.create_timer(1/10,self.timer_callback)
        self.hasReached = False
        self.p_f = np.array([0.,0.,0.])
        self.q = np.array([0.,0.,0.])

    def p_f_sub_callback(self,msg):
        self.p_f = msg.data
    def q_callback(self,msg:JointState):
        self.q = msg.position

    def timer_callback(self):
        P = FPK(self.q)
        pos = np.array(P[-1][:3,3])
        if((abs(np.array(self.p_f) - pos) <= 0.001).all()):
            self.hasReached = True
        else:
            self.hasReached = False
        if(self.hasReached):
            msg = Bool()
            msg.data = self.hasReached
            self.hasReached_pub.publish(msg)
            print(self.hasReached)
```

ทำหน้าที่ตรวจสอบจุดตำแหน่งเป้าหมายสุดท้าย Reference ที่ได้จาก Scheduler และนำค่าที่ได้จาก Joinstate ของหุ่นยนต์จาก Gazebor นำมาทำ Forward Kinematic เพื่อนำมาเปรียบเทียบว่าหุ่นยนต์ถึงจุดเป้าหมายแล้วหรือยัง หากถึงเป้าหมายแล้วก็จะส่ง Msg hasReached ไปยัง Scheduler เพื่อเปลี่ยนตำแหน่งไปยังตำแหน่งถัดไป

ผลลัพธ์ที่ได้