Sistema FIEB | SENAI CIMATEC

PELO FUTURO DA INOVAÇÃO

CENTRO UNIVERSITÁRIO SENAI CIMATEC

Relatório Final

Manipulador robótico com visão computacional

Apresentada por: Tiago Barretto Sant'Anna

Orientado por: Prof. Marco Reis, M.Eng.

Dezembro de 2021

Tiago Barretto Sant'Anna

Manipulador robótico com visão computacional

Salvador
Centro Universitário SENAI CIMATEC
2020

Capítulo Um

Introdução

1.1 Objetivos

Os objetivos são realizar a programação de um manipulador robótico especializado, em conjunto com visão computacional para poder realizar a automação de tarefas.

1.1.1 Objetivos Específicos

Os objetivos específicos deste projeto são:

- Controlar um manipulador utilizando o ROS
- Identificar Objetos utilizando visão computacional
- Integrar o manipulador com a visão computacional

1.2 Justificativa

Justificativa

1.3 Organização do documento

Este documento apresenta 3 capítulos e está estruturado da seguinte forma:

- **Capítulo ?? - Introdução**: Contextualiza o âmbito, no qual a pesquisa proposta está inserida. Apresenta, portanto, a definição do problema, objetivos e justificativas da pesquisa e como este relatório final está estruturado;
- **Capítulo ?? - Dia: XX/XX/XX**: Explicita com toda a base teorica como o trabalho foi desenvolvido diariamente, o processo para a sua obtenção.

Capítulo Dois

Dia: 08/02/22

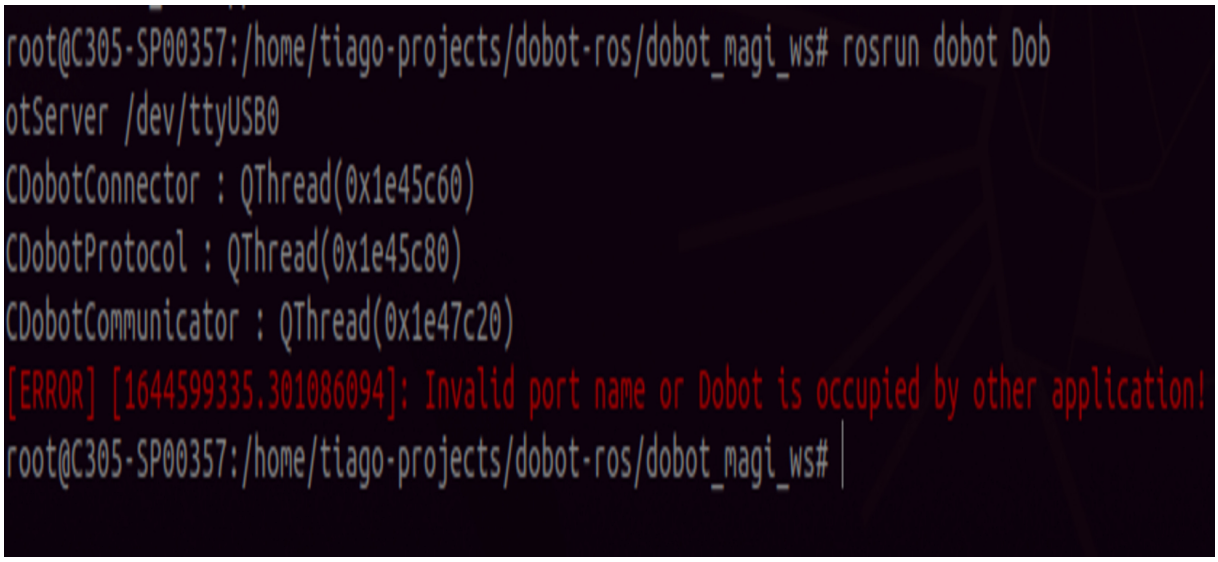
Após a reunião começou a buscar mais material sobre o funcionamento do Dobot. Nessa pesquisa se encontrou um código (**??**) que fazia com que o DOBOT desenha-se um espiral em um papel, podendo ser utilizado para se basear na programação da movimentação do braço. Também foi feito um estudo do manipulador

Capítulo Três

Dia: 09/02/22

Primeiro surgiu uma tentativa de testes do dobot usando a docker porém o seguinte erro aparecia no terminal:

Figura 3.1: terminal



Fonte: Autoria propria.

A primeira hipótese foi substituir a docker que estava sendo utilizada, que era uma docker de ROS Kinetic, e retorna para o tipo de docker que estava sendo utilizada anteriormente, que era uma docker de ubuntu xenial com ubuntu instalado. Isso foi feito com base no fato de que na primeira tentiva se tinha usado a docker dessa forma e tinha funcionado, precisaria de um OS para poder controlar as portas e no manual está recomendado o uso dessa versão do ubuntu com essa versão do ros instalado. A partir disso foi criado essa imagem do xenial com o ROS Kinetic instalado e armazenado no seguinte repositório no DockerHUB.

Porém, mesmo realizando isso o problema persistiu e decidiu investigar mais ainda o uso de portas USB na docker, a partir disso descobriu-se que precisava subir as portas USBs que seriam utilizadas, assim no código de docker_run_xenial.sh foi substituído a parte que sobe as portas de videos pela USB

Antes:

```
V4L2_DEVICES="  "
```

```
for i in {0..9}
do
    if [ -a "/dev/ttyVideo$i" ]; then
        V4L2_DEVICES="$V4L2_DEVICES --device /dev/ttyVideo$i
"
    fi
done

echo "V4L2_DEVICES:  $V4L2_DEVICES"
```

Depois:

```
V4L2_DEVICES=" "

for i in {0..9}
do
    if [ -a "/dev/ttyUSB$i" ]; then
        V4L2_DEVICES="$V4L2_DEVICES --device /dev/ttyUSB$i "
    fi
done

echo "V4L2_DEVICES:  $V4L2_DEVICES"
```

Depois disso o DOBOT conseguiu se conectar e se partiu para o desenvolvimento dos códigos

Capítulo Quatro

Dia: 10/02/22

Nesse dia foi majoritariamente focado no do desenvolvimento da programação para o funcionamento do Dobot.

A demo do DOBOT funciona utilizando Services e o (**??**) fez um tópico para poder movimentar o braço. Assim o primeiro desafio foi aprender a utilizar os Services do ROS, para isso foi utilizado a própria wiki oficial (**??**).

```
#!/usr/bin/env python
import rospy

from dobot.srv import GetPose

if __name__ == '__main__':
    print('1')
    # rospy.wait_for_service('print_pose')
    get_pose = rospy.ServiceProxy('DobotServer/GetPose',
GetPose)
    print('2')


    try:
        print('a')
        print(get_pose())
        print('b')
    except rospy.ServiceException as exc:
        print("Service did not process request: " + str(exc)
)

    var = get_pose()
    print(var.x)
```

Dessa forma foi desenvolvido um código para poder exibir na tela a posição atual, já que seria necessário receber a posição do braço para poder fazer os cálculos da sua movimentação. Assim após finalizado foram incrementados no código position_control.py para controle de posição desenvolvido esse services

```
#!/usr/bin/env python
from turtle import pu
import rospy
from geometry_msgs.msg import Pose
import time
from dobot.srv import GetPose
import math

def distancia(ini, fim):
    return math.sqrt((fim - math.sqrt((ini)**2))**2)

def position():
    rospy.init_node('position_control', anonymous=True)
    publisher = rospy.Publisher('geometry_pose', Pose,
queue_size=10)
    get_pose = rospy.ServiceProxy('DobotServer/GetPose',
GetPose)


    print("Say where you want me to go")
    x = int(input("X axis: "))
    y = int(input("Y axis: "))
    z = int(input("Z axis: "))    # -*- coding: utf-8 -*-
import rospy
from geometry_msgs.msg import Pose

# Vai importar todos os srv do dobot
from dobot.srv import SetEndEffectorSuctionCup
from dobot.srv import GetPose
from dobot.srv import SetHOMEParams

if __name__ == "__main__":
    rospy.init_node('pick_and_place', anonymous=True)
    suction_srv = rospy.ServiceProxy('DobotServer/
SetEndEffectorSuctionCup', SetEndEffectorSuctionCup)
    get_pose = rospy.ServiceProxy('DobotServer/GetPose',
GetPose)
    set_home = rospy.ServiceProxy('DobotServer/SetHOMEParams
', SetHOMEParams)
    publisher = rospy.Publisher('geometry_pose', Pose,
queue_size=10)
```

```
        # subscriber = rospy.Subscriber

        pose = get_pose()
        home = set_home(0, 0, 0, 0, False)
        ini_x = pose.x
        ini_y = pose.y
        ini_z = pose.z

        print('a')
        # Valores arbitrarios para delimitar a area de atuac o
    do robo
        while True:
            print('The actual pose is')
            print('X: ')
            print(pose.x)
            print('Y: ')
            print(pose.y)
            print('Z: ')
            print(pose.z)
            print()

            print("Say where the object is positioned in the
    table")
            x = int(input("X axis: "))
            y = int(input("Y axis: "))
            z = int(input("Z axis: "))
            if 0 <= x > 200:
                print('X value is not acceptable')
            elif -100 < y > 100:
                print('Y value is not acceptable')
            elif -100 < z > 100:
                print('Z value is not acceptable')
            else:
                break

        freq = rospy.Rate(10)

        while not rospy.is_shutdown():
            msg = Pose()
            resp = suction_srv(1, 0, False)
```

```
            print('Move ate o ponto para pegar o objeto')
            # dist = 1
            while (pose.x != x and pose.y != y and pose.z != z)
or not rospy.is_shutdown():
                msg.position.x = x
                msg.position.y = y
                msg.position.z = z
                publisher.publish(msg)
                # dist = ((x - pose.x) + (y - pose.y) + (z -
pose.z))/3



            print('Suga')
            resp = suction_srv(1, 254, False)

            # dist = 1
            print('Move ate o ponto inicial')
            while (pose.x != ini_x and pose.y != ini_y and pose.
z != ini_z) or not rospy.is_shutdown():
                msg.position.x = ini_x
                msg.position.y = ini_y
                msg.position.z = ini_z
                publisher.publish(msg)
                dist = ((x - pose.x) + (y - pose.y) + (z - pose.
z))/3

            print('Para de sugar')
            resp = suction_srv(1, 0, False)

            freq.sleep()

            break

        while dist >= 0.1:
            print('a')
            msg = Pose()
            pose = get_pose()

            msg.position.x = k * distancia(pose.x, x)
            msg.position.y = k * distancia(pose.y, y)
```

```
            msg.position.z = k * distancia(pose.z, z)
            publisher.publish(msg)

            dist = (distancia(pose.x, x) + distancia(pose.z, z)
+ distancia(pose.y, y)) / 3

            freq.sleep()

if __name__ == "__main__":
    try:
        position()
    except rospy.ROSInterruptException:
        pass
```

Esse código foi feito baseado no código desenvolvido por (**??**). Porém, nenhuma das suas versões conseguiram movimentar o braço robótico.

Depois de testar a movimentação decidi fazer um codigo para controlar as ferramentas do manipulador. Como a função do manipulador é pegar objetos as ferramentas escolhidas foram o *gripper* e o *suction cup*, ambos de funcionamento pneumático.

```
#! /usr/bin/env python
from numpy import uint8
import rospy
from dobot.srv import SetEndEffectorSuctionCup

if __name__ == "__main__":
    rospy.init_node('suction_test', anonymous=True)
    suction_srv = rospy.ServiceProxy('DobotServer/
SetEndEffectorSuctionCup', SetEndEffectorSuctionCup)

    x = uint8(input('Put value:  '))

    freq = rospy.Rate(10)

    while not rospy.is_shutdown():
        resp = suction_srv(1, x, False)
        print(resp.result)
        resp.result
        freq.sleep()
```

O código acima foi desenvolvido para testar o service do *suction cup*. Assim foi possível ativar a bomba pneumática e desativa-la. Porém a bomba não faz o movimento de succionar, apenas de soprar ar, mesmo testando diversas combinações com as variáveis do serviço.

```
    from numpy import uint8
    import rospy
    from dobot.srv import SetEndEffectorGripper

    if __name__ == "__main__":
        rospy.init_node('suction_test', anonymous=True)
        suction_srv = rospy.ServiceProxy('DobotServer/
  SetEndEffectorGripper', SetEndEffectorGripper)

        x = uint8(input('Put value:  '))

        freq = rospy.Rate(10)

        while not rospy.is_shutdown():
            resp = suction_srv(1, x, False)
            print(resp.result)
            resp.result
            freq.sleep()
```

O código acima teve como objetivo testar o service do *gripper*. Testando descobriu que este service também consegue ativar a bomba, mas não desativa-la nem sugar ar. Para que possa desativa-la foi usado o service do *suction cup*. Dessa forma foi decidido utilizar o service do *suction cup* chamado de *SetEndEffectorSuctionCup* para ambos, ja que só precisa ativar e desativar a bomba para poder usar ambos.

Alguns problemas mecanicos também foram encontrados a parte que parafusaria o *gripper* no braço fica muito folgada, dessa forma ele não prende no braço, mas tem como utilizar essa ferramenta se fizer um calço ou adaptador para prende-lo. Por outro lado o *suction cup* consegue ser preso facilmente no manipulador, ela não tem como segurar objetos pois a bomba não faz a sucção

Dia: 11/02/22

Foi corrigido o código do pick_and_place.py o DOBOT conseguiu se mover como mostrado no video seguinte: Dobot Porém a bomba ainda continua soprando ar ao invés de puxar e o braço ainda continua com o movimento muito lento

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
import rospy
from geometry_msgs.msg import Pose

# Vai importar todos os srv do dobot
from dobot.srv import SetEndEffectorSuctionCup
from dobot.srv import GetPose
from dobot.srv import SetHOMEParams

if __name__ == "__main__":
    rospy.init_node('pick_and_place', anonymous=True)
    suction_srv = rospy.ServiceProxy('DobotServer/
SetEndEffectorSuctionCup', SetEndEffectorSuctionCup)
    get_pose = rospy.ServiceProxy('DobotServer/GetPose',
GetPose)
    set_home = rospy.ServiceProxy('DobotServer/SetHOMEParams
', SetHOMEParams)
    publisher = rospy.Publisher('geometry_pose', Pose,
queue_size=10)
    # subscriber = rospy.Subscriber

    pose = get_pose()
    home = set_home(0, 0, 0, 0, False)
    ini_x = pose.x
    ini_y = pose.y
    ini_z = pose.z

    print('a')
    # Valores arbitrarios para delimitar a area de atuac o
do robo
    while True:
```

```
            print('The actual pose is')
            print('X: ')
            print(pose.x)
            print('Y: ')
            print(pose.y)
            print('Z: ')
            print(pose.z)
            print()

            print("Say where the object is positioned in the
table")
            x = int(input("X axis: "))
            y = int(input("Y axis: "))
            z = int(input("Z axis: "))
            if 0 <= x > 200:
                print('X value is not acceptable')
            elif -100 < y > 100:
                print('Y value is not acceptable')
            elif -100 < z > 100:
                print('Z value is not acceptable')
            else:
                break

        freq = rospy.Rate(10)

        while not rospy.is_shutdown():
            msg = Pose()
            resp = suction_srv(1, 0, False)

            print('Move ate o ponto para pegar o objeto')
            # dist = 1
            while (pose.x != x and pose.y != y and pose.z != z)
or not rospy.is_shutdown():
                msg.position.x = x
                msg.position.y = y
                msg.position.z = z
                publisher.publish(msg)
                # dist = ((x - pose.x) + (y - pose.y) + (z -
pose.z))/3
```

```
            print('Suga')
            resp = suction_srv(1, 254, False)

            # dist = 1
            print('Move ate o ponto inicial')
            while (pose.x != ini_x and pose.y != ini_y and pose.
z != ini_z) or not rospy.is_shutdown():
                msg.position.x = ini_x
                msg.position.y = ini_y
                msg.position.z = ini_z
                publisher.publish(msg)
                dist = ((x - pose.x) + (y - pose.y) + (z - pose.
z))/3

            print('Para de sugar')
            resp = suction_srv(1, 0, False)

            freq.sleep()

            break
```

O código acima foi desenvolvido com base nos códigos desenvolvidos anteriormente e testado no manipulador até que consiga o movimenta-lo.

Manipulador robótico com visão computacional

Tiago Barretto Sant'Anna

Salvador, Dezembro de 2021.