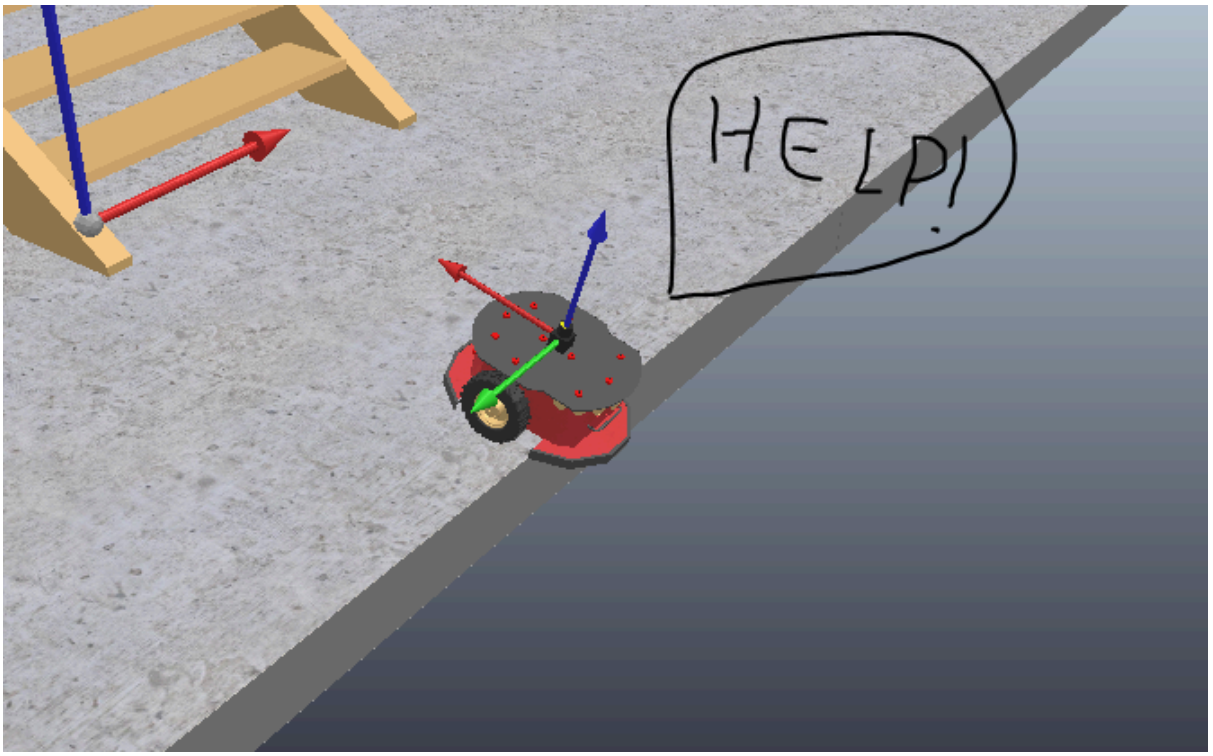
UNIVERSIDADE FEDERAL
DE MINAS GERAIS



TRABALHO PRÁTICO 1
FERRAMENTAS E TRANSFORMAÇÕES

ALUNOS:
FELIPPE VELOSO MARINHO
MATRÍCULA: 2021072260

JOÃO VITOR MATEUS SILVA
MATRÍCULA: 2020425801

DISCIPLINA: ROBÓTICA MÓVEL

1 - Introdução

- O presente trabalho tem o objetivo de documentar a implementação dos métodos e códigos na intenção de familiarizar-se com os conceitos básicos de descrição espacial, transformações e a utilização do software *CoppeliaSim* para simulação.

- Para facilitar a integração e o controle das simulações, assim como nas aulas, utilizamos uma API oferecida pelo próprio software. Neste caso, foi utilizado a *zmqRemoteApi*, sendo uma versão mais atualizada e robusta da utilizada em aula. Por fim, para o trabalho foi utilizado a linguagem Python por maior familiaridade e disponibilidade de materiais de referência.

2 - Como executar o código

Primeiramente, o trabalho foi realizado utilizando notebooks para melhor organização e visualização das postagens. Portanto, é necessário executar as cédulas em ordem para que não ocorram problemas de funções utilizadas em outras cédulas.

Para que possamos executar o código da forma correta, temos que seguir alguns passos:

- Inicialmente temos que baixar as bibliotecas necessárias para utilização do Simulador, CoppeliaSim através do seguinte código: `pip install coppeliasim-zmqremoteapi-client`

- Outra biblioteca necessária para que o projeto funcione é a keyboard. Utilizada unicamente na função utilizada para as plotagens do item 6 do trabalho.Sua instalação é feita pelo código: `pip install keyboard`

- Em seguida, importamos as bibliotecas do Python utilizadas para transformação de matrizes e plotagem dos gráficos de resultados, sendo elas:

```
import numpy as np
import matplotlib.pyplot as plt
from coppeliasim_zmqremoteapi_client import *
import time
import struct
```

- Após essas importações, basta criar a conexão do cliente de API remota, através do seguinte código e rodar o sistema normalmente:

Link para repositório da SimZMQRemoteApi:
https://github.com/CoppeliaRobotics/zmqRemoteApi/tree/master/clients/python

```
# create a client to connect to zmqRemoteApi server:
# (creation arguments can specify different host/port,
# defaults are host='localhost', port=23000)
client = RemoteAPIClient()

# get a remote object:
sim = client.require('sim')

# call API function fo test:
```

```
h = sim.getObject('/Floor')
print("Printando o chão: " + str(h))
```

Observações:

- **Todas** as funções auxiliares devem ser executadas.
- Em determinados trechos é necessário que a simulação esteja em execução. Isso é avisado sempre no início do trecho do código.

3 - Método

3.1 - Criação da Cena

Para criação da cena fizemos um cenário simples com três objetos do tipo Floor empilhados e com os seguintes objetos espalhados pelo mapa, totalizando **<u>cinco</u>**:

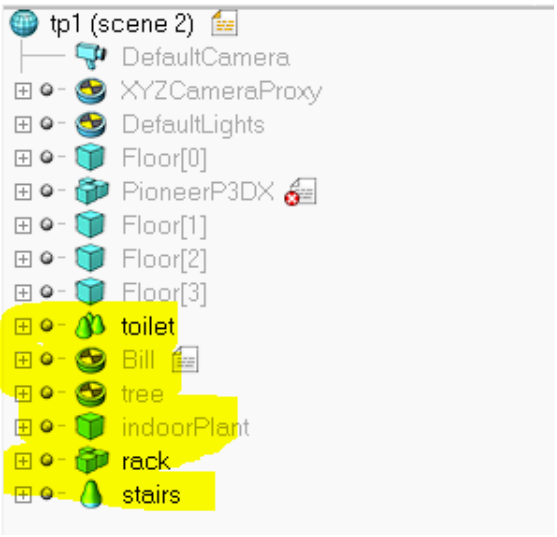


Fig 1 - Objetos do Cenário

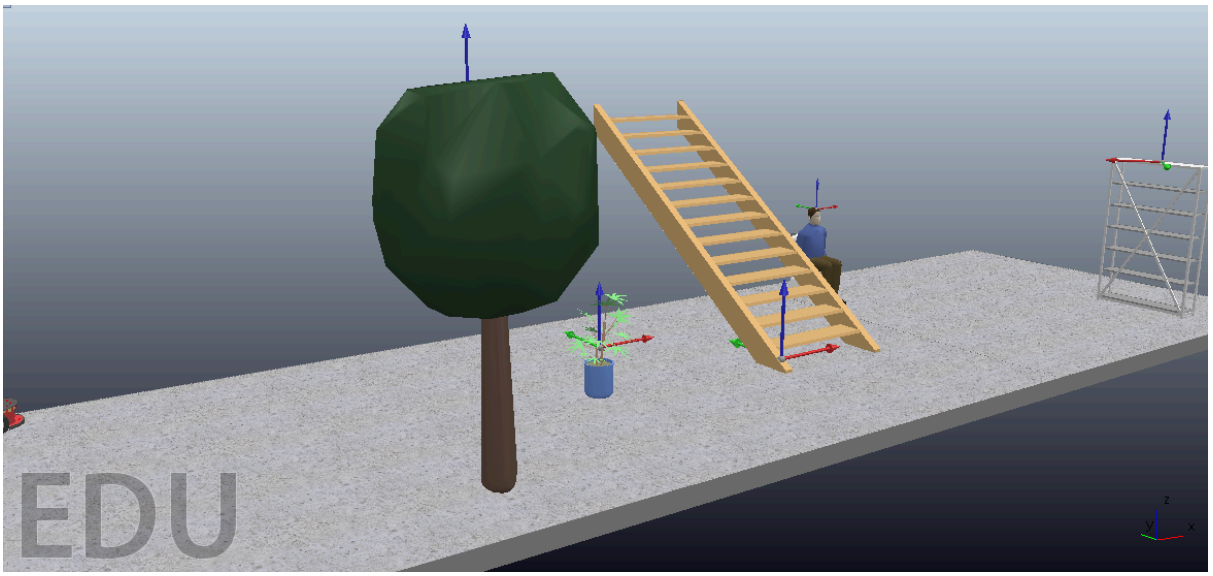


Fig 2 - Elementos distribuídos no cenário

3.2 - Representação do Sistema de Coordenadas e Transformações

Primeiramente, foram adicionados os “reference frames” no cenário e afixado em cada um dos cinco objetos da análise.
A convenção adotada para representação das coordenadas locais foi:

Convenção para sistema de coordenadas geral e sistema de coordenadas locais

- World coordinate frame (WCF)
- Robot coordinate frame (RCF)
- Indoor Plant coordinate frame (ICF) ou (IPCF)
- Stairs coordinate frame (SCF)
- Bookcase frame (BCF)
- Person on Toilet coordinate frame (PCF)
- Tree coordinate frame (TCF)

FAZER UMA OUTRA EM 2D

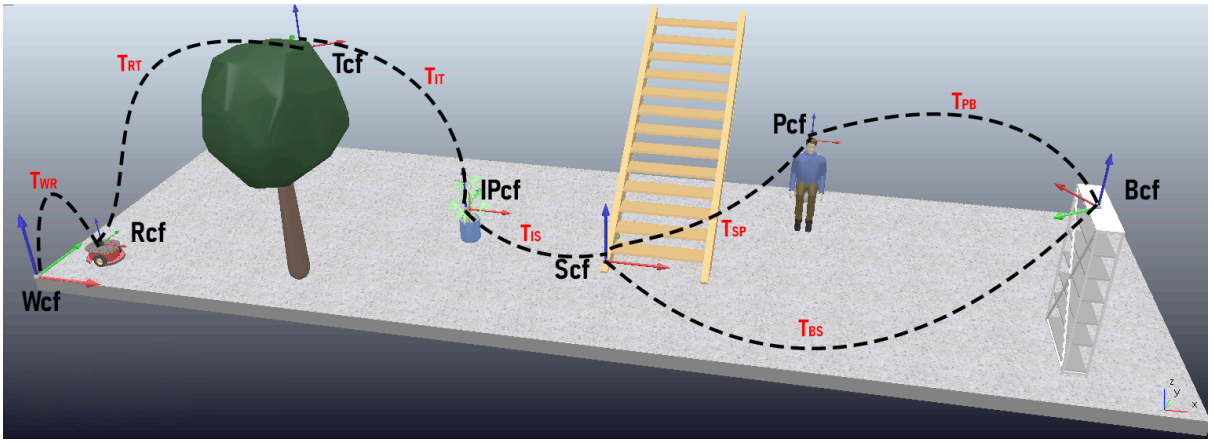


Fig 3 - Representação dos sistemas de coordenadas e transformações entre eles

3.3 - Representação do Sistema de Coordenadas e Transformações

Para que nosso robô saia de sua posição inicial para ir até uma outra posição, precisamos saber onde nosso robô está. Isso é definido em relação a algum outro ponto de coordenadas, o mesmo vale para os outros elementos do cenário.

Primeiramente, foram definidas as posições e orientações de cada elemento foram recuperadas utilizando-se dos recursos da RemoteAPI.

Após isso, foram definidas as posições, rotações e cores para representação dos plots em diferentes dicionários para facilitar os loops feitos para a construção das matrizes posteriormente.

```
    # Posições e orientações dos pontos a até f (você precisa atribuir os valores corretos)
    positions = {
        'Robot': robot_position,
        'Tree': tree_position,
        'IndoorPlant': plant_position,
        'Stairs': stairs_position,
        'PersonOnToilet': vase_position,
        'Bookcase': rack_position
    }
```

```
    rotations = {
        'Robot': Rz(robot_orientation[2]),  # Assume que a orientação
é dada em relação ao eixo z
        'Tree': Rz(tree_orientation[2]),
        'IndoorPlant': Rz(plant_orientation[2]),
        'Stairs': Rz(stairs_orientation[2]),
        'PersonOnToilet': Rz(vase_orientation[2]),
        'Bookcase': Rz(rack_orientation[2])
    }

    # Cores correspondentes para cada ponto
    point_colors = {
        'Robot': 'b',
        'Tree': 'g',
        'IndoorPlant': 'r',
        'Stairs': 'c',
        'PersonOnToilet': 'm',
        'Bookcase': 'y'
    }
```

As matrizes de transformação homogêneas foram definidas para cada ponto em relação ao robô através de um loop que utiliza o dicionário de rotação para atribuir à parte superior esquerda (3x3) da matriz 4x4 e o de posição para a coluna direita (3ª coluna) da matriz de transformação. Obedecendo assim a definição:

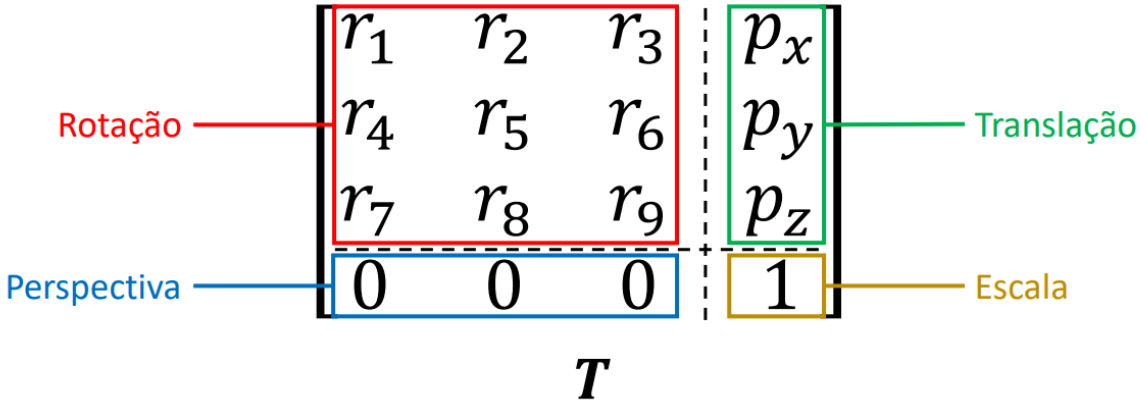


Fig 4 - Matriz de transformação homogênea

Para facilitar a manipulação na plotagem, as matrizes de transformação foram adicionadas no dicionário `transformations_rel_to_robot`.

```
# Construindo as matrizes de transformação homogêneas para cada ponto
em relação ao robô
    transformations_rel_to_robot = {}
    for point, position in positions.items():
```

```
            rotation = rotations[point]
            transformation = np.eye(4)
            transformation[:3, :3] = rotation
            transformation[:3, 3] = position
            transformations_rel_to_robot[point] = transformation
```

Para melhor visualização na plotagem do gráfico, o robô foi movido para a origem e os objetos foram “movidos” de acordo com a posição do robô.

Depois disso, utilizando matplotlib e funções adaptadas diretamente da aula 5 foi possível a plotagem da os referenciais e relacionamentos entre os elementos do cenário e o referencial local (PioneerP3DX).

Para plotar os vetores de transformação entre os pontos faço um loop externo que percorre cada ponto de origem (src_point) e sua respectiva transformação (src_transformation). O loop interno percorre cada ponto de destino (dst_point) e sua respectiva transformação (dst_transformation). Dentro do loop interno faço uma condição para que não sejam plotados vetores de transformação e vetores de posição para o mesmo ponto.

```
# Plotando os vetores de transformação de um ponto para o outro e
vetores de posição
for src_point, src_transformation in
transformations_rel_to_robot.items():
    for dst_point, dst_transformation in
transformations_rel_to_robot.items():
        if src_point != dst_point:
            plot_vector(src_transformation[:2, 3],
dst_transformation[:2, 3])
        else:
            plt.quiver(*robot_position[:2], *(src_transformation[:2, 3]
- robot_position[:2]), color=point_colors[src_point], angles='xy',
scale_units='xy', scale=1)
```

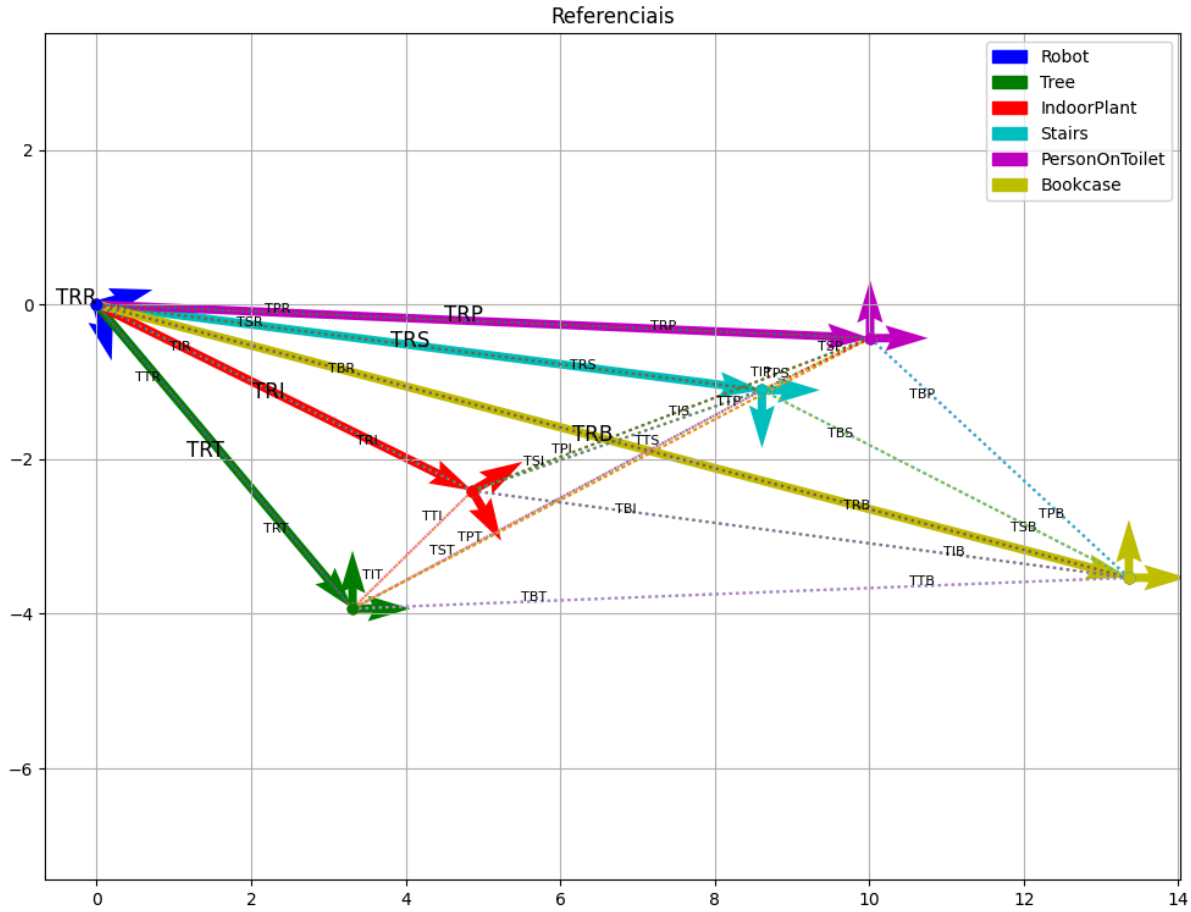


Fig 5 - Plotagem da os referenciais e relacionamentos entre os elementos do cenário e o referencial local

3.4 - Verificando a implementação Anterior

Colocando o robô em três posições diferentes, podemos verificar se os respectivos plots de referenciais e relacionamentos se comportam semelhantes ao primeiro exemplo no item anterior. Sendo assim, abaixo segue as coordenadas do Pioneer 3DX (ponto usado como referencial) e as imagens dos cenários e dos plots.

```
1 - Posição de Robot no coppelia: [-0.7616605229925083,
1.083573487885349, 0.13864973436883632]
Orientação de Robot no coppelia: [-7.82908952975391e-05,
-0.002904717944659432, -1.3096476337979985]
```

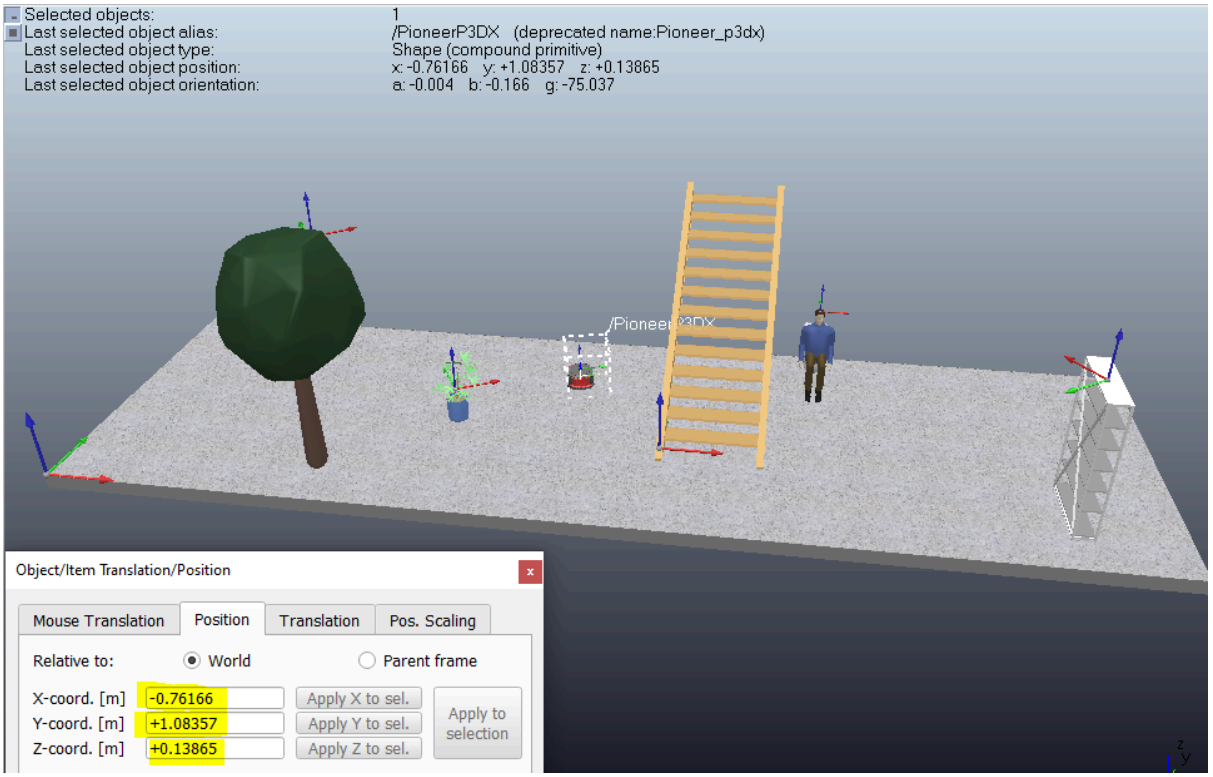


Fig 6 - Cenário com as posições do Pioneer destacadas 1

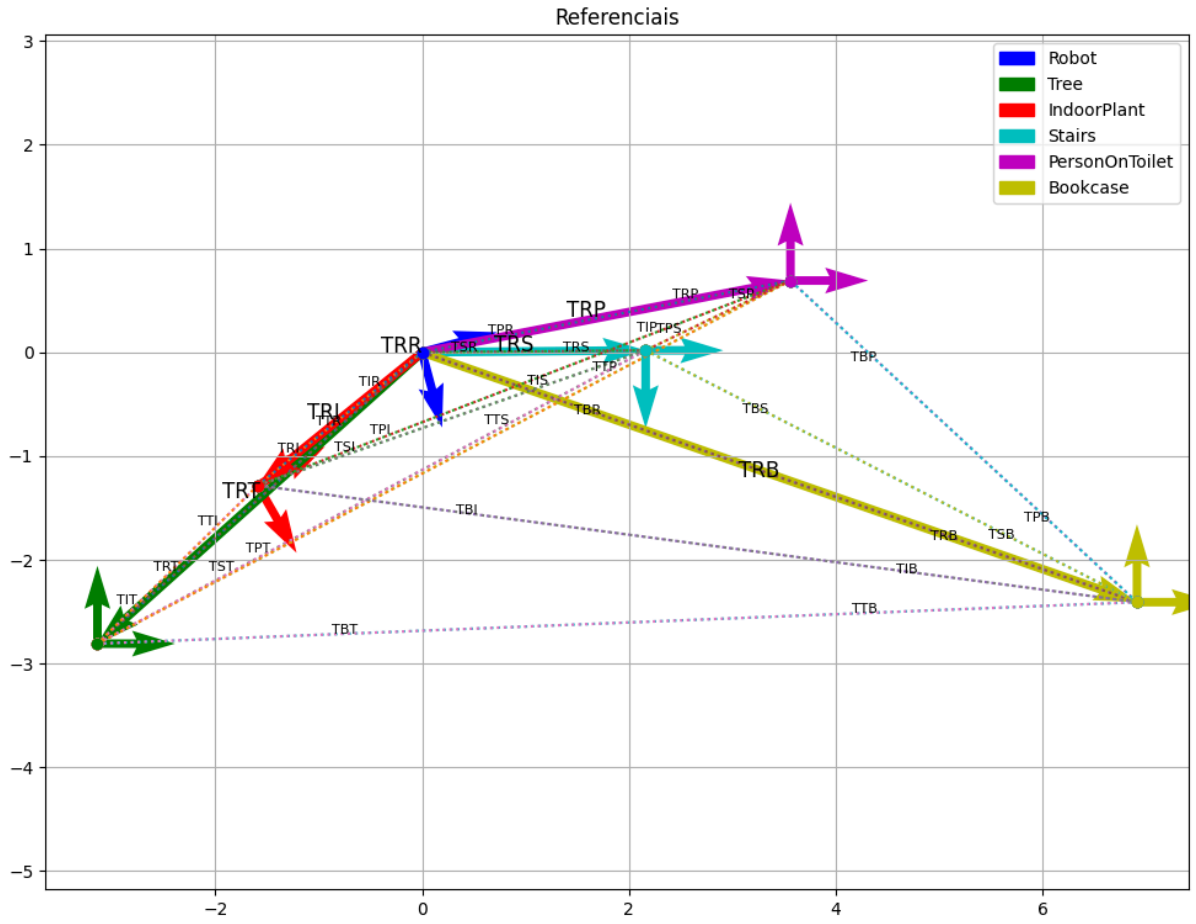


Fig 7 - Plotagem dos referenciais e relacionamentos com a origem (PioneerP3DX) em diferentes posições 1

2 - Posição de Robot no coppelia: [4.538339477007493, -0.3664265121146512, 0.13864973436883632]
Orientação de Robot no coppelia: [-7.82908952975391e-05, -0.002904717944659432, -1.3096476337979985]

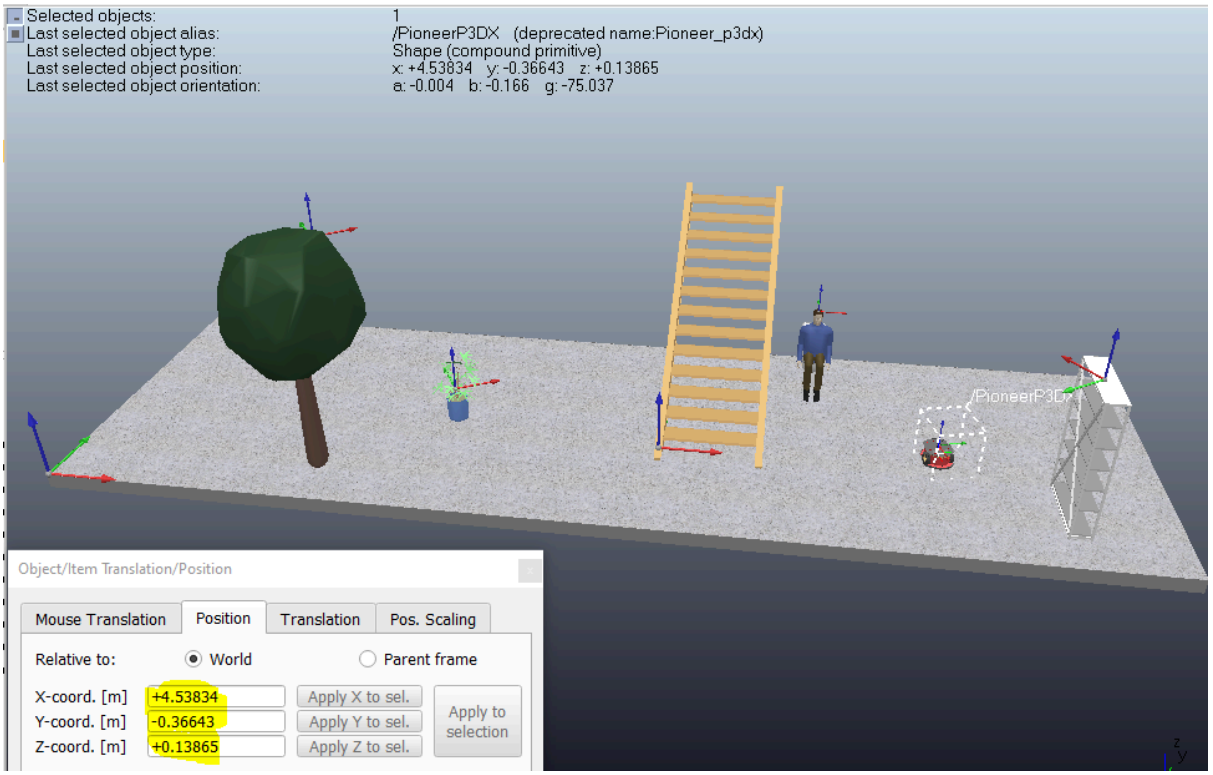


Fig 8 - Cenário com as posições do Pioneer destacadas 2

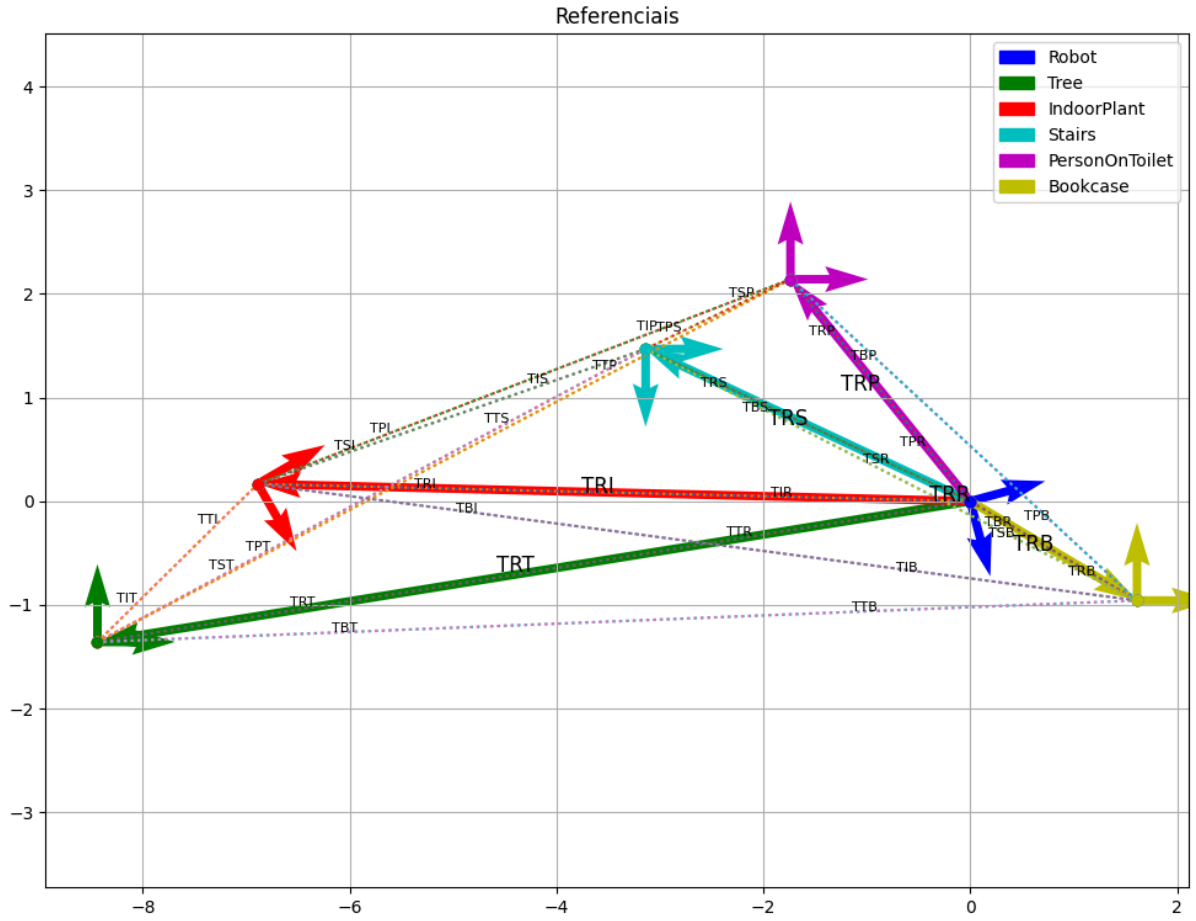


Fig 9 - Plotagem dos referenciais e relacionamentos com a origem (PioneerP3DX) em diferentes posições 2

```
3 - Posição de Robot no coppelia: [0.738339477007492,
-1.5664265121146506, 0.13864973436883632]
Orientação de Robot no coppelia: [-7.829089529753607e-05,
-0.0029047179446594347, 1.657412094592146]
```

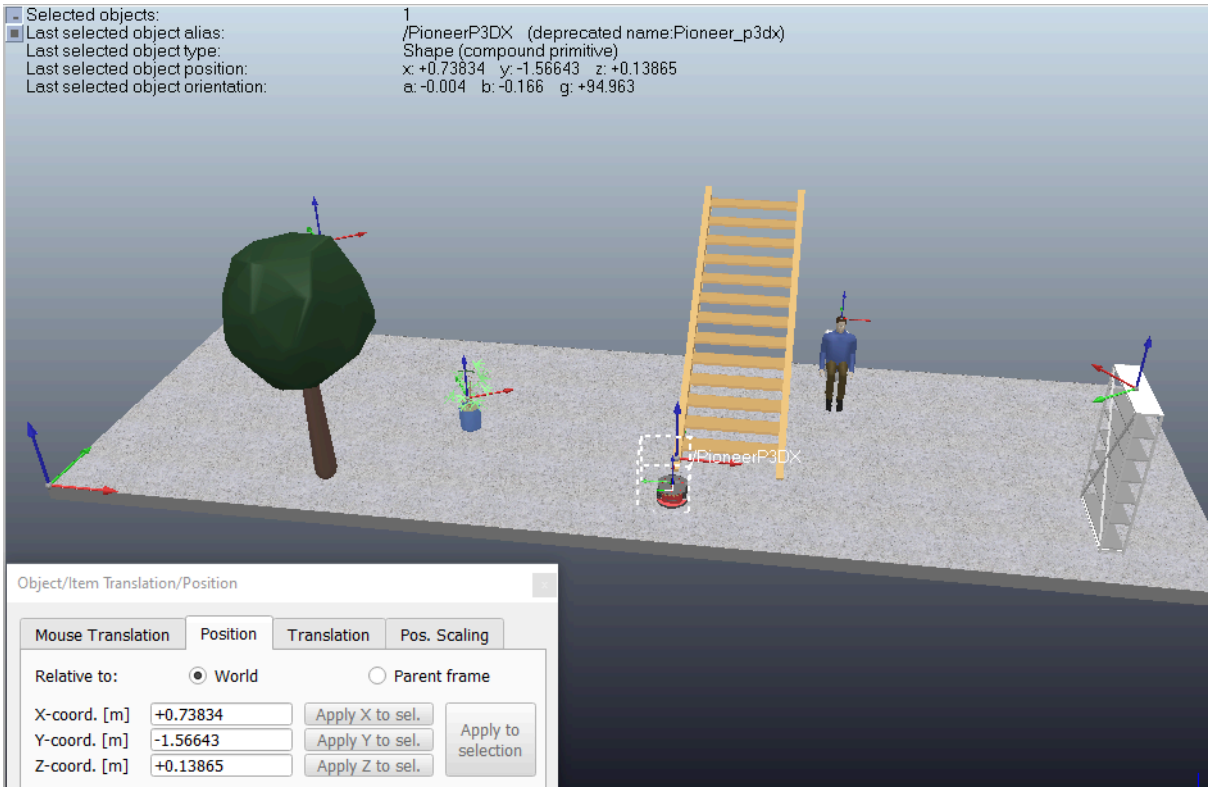


Fig 10 - Cenário com as posições do Pioneer destacadas 3

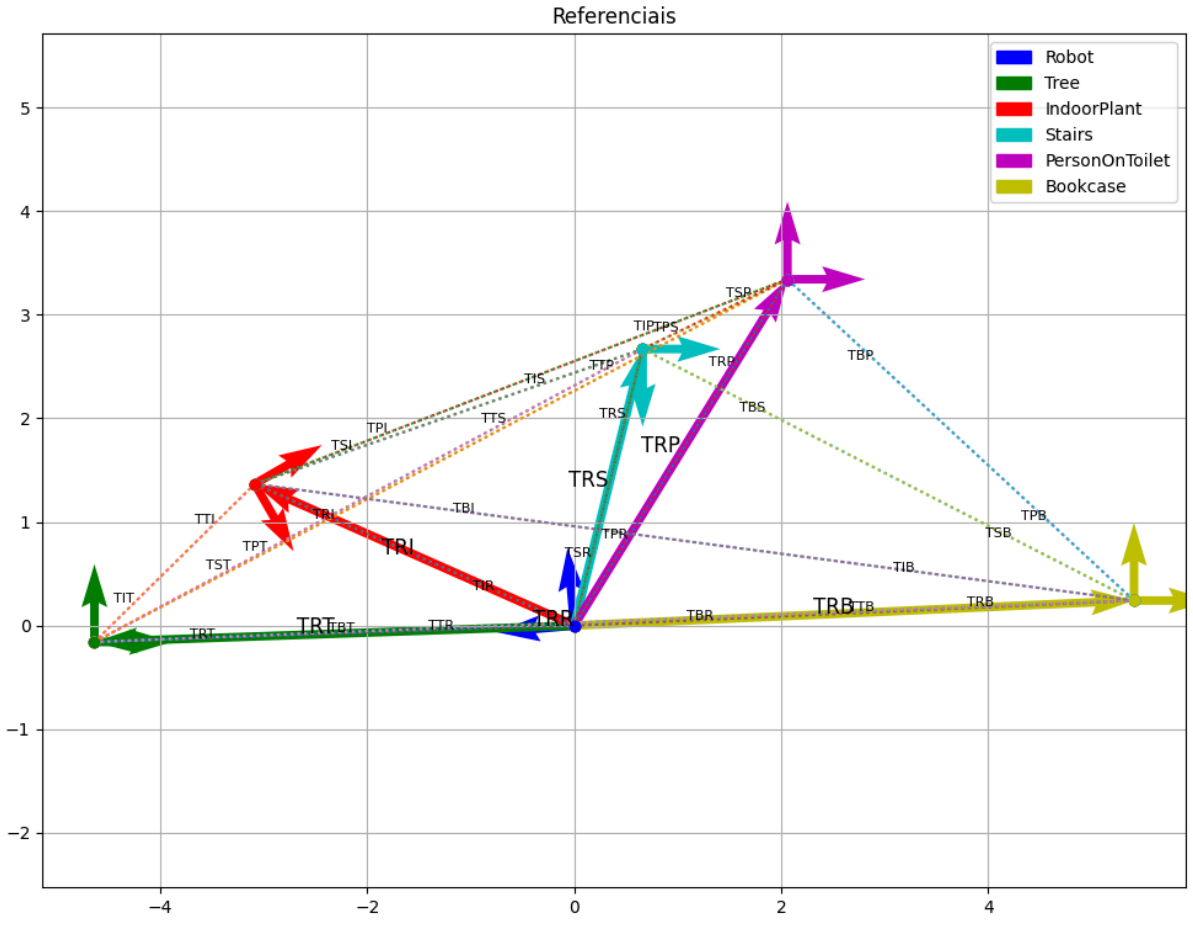


Fig 11 - Plotagem dos referenciais e relacionamentos com a origem (PioneerP3DX) em diferentes posições 3

3.5 - Leituras do Laser

Primeiro, foi adicionado a cena o robô com laser utilizado na aula 3 (Ferramental (Python e CoppeliaSim)). Seguindo o enunciado, foram recuperadas as matrizes de transformação Trl (laser → robô) e Twr (robô → mundo).

```
    # Handle para o ROBÔ
    laser_robot = sim.getObject('/Pioneer_p3dx')

    #Handle para o LASER
    laser = sim.getObject('/Pioneer_p3dx/fastHokuyo')

    # Matriz de transformação Trl (laser → robô)
    Trl = sim.getObjectMatrix(laser, laser_robot) # Matriz de
transformação do laser em relação ao robô
    Trl = np.hstack([Trl, [0, 0, 0, 1]])
    print("Matriz de transformação do laser em relação ao robô: " +
str(Trl))

    # Matriz de transformação Twr (robô → mundo)
    Twr = sim.getObjectMatrix(laser_robot, -1) # Matriz de
transformação do robô em relação ao mundo
    Twr = np.hstack([Twr, [0, 0, 0, 1]])
    print("Matriz de transformação do robô em relação ao mundo: " +
str(Twr))
```

Com isso, foi modificado o script da aula 3 para que funcione na nova API e que os pontos sejam plotados no referencial global, de acordo com a posição atual do robô.

```
def draw_laser_data_global(laser_data, robot_position,
robot_orientation, max_sensor_range=5):
    # Adicionar legenda usando os objetos de patch e os rótulos dos
pontos
    fig = plt.figure(figsize=(12, 9), dpi=100)
    ax = fig.add_subplot(111, aspect='equal')

    for i in range(len(laser_data)):
        ang, dist = laser_data[i]

        if (max_sensor_range - dist) > 0.1:
            # Transformando as coordenadas do laser para o referencial
global
            x_laser = dist * np.cos(ang)
            y_laser = dist * np.sin(ang)
```

```
            x_global = robot_position[0] + x_laser *
math.cos(robot_position[2]) - y_laser * math.sin(robot_position[2])
            y_global = robot_position[1] + x_laser *
math.sin(robot_position[2]) + y_laser * math.cos(robot_position[2])
            c = 'r' if ang < 0 else 'b'
            ax.plot(x_global, y_global, 'o', color=c)

    #plotando o robô como um ponto preto
    ax.plot(robot_position[0], robot_position[1], 'ko')

    #colocando uma legenda para o robô
    ax.text(robot_position[0], robot_position[1], 'Robo', fontsize=12,
ha='right')
    ax.grid()

    # Ajuste dos limites dos eixos para cobrir todo o cenário e os
pontos observados pelo scanner
    min_x = min(robot_position[0], min([x_global for x_global, _ in
laser_data]))
    max_x = max(robot_position[0], max([x_global for x_global, _ in
laser_data]))
    min_y = min(robot_position[1], min([y_global for _, y_global in
laser_data]))
    max_y = max(robot_position[1], max([y_global for _, y_global in
laser_data]))

    # Adicionando uma margem de segurança de 1 metro ao redor dos
pontos
    margin = 1
    ax.set_xlim([min_x - margin, max_x + margin])
    ax.set_ylim([min_y - margin, max_y + margin])

    plt.show()
# Conectando-se ao CoppeliaSim
# Run a simulation in asynchronous mode:
clientID = sim.startSimulation()

if clientID != -1:
    print("Connected to remote API server")
    # Posição do robô
    laser_robot_position = get_object_position(sim, '/Pioneer_p3dx')
    print("Posição do robô: " + str(laser_robot_position))
    # Orientação do robô
```

```
    laser_robot_orientation = get_object_orientation(sim,
'/Pioneer_p3dx')

    # Handle para os dados do LASER
    laser_range_data = "hokuyo_range_data"
    laser_angle_data = "hokuyo_angle_data"

    # Prosseguindo com a leitura dos dados
    raw_range_data, raw_angle_data = readSensorData(clientID,
laser_range_data, laser_angle_data)
    laser_data = np.array([raw_angle_data, raw_range_data]).T

    # Plotando os dados do sensor
    if laser_range_data is not None and laser_angle_data is not None:
        draw_laser_data_global(laser_data, laser_robot_position,
laser_robot_orientation)
```

Para verificação dos resultados, serão feitos os plots de três diferentes posições.

```
1 - Posição do robô: [-0.3496627942431832, -0.9790304302348692,
0.13865410333096223]
```

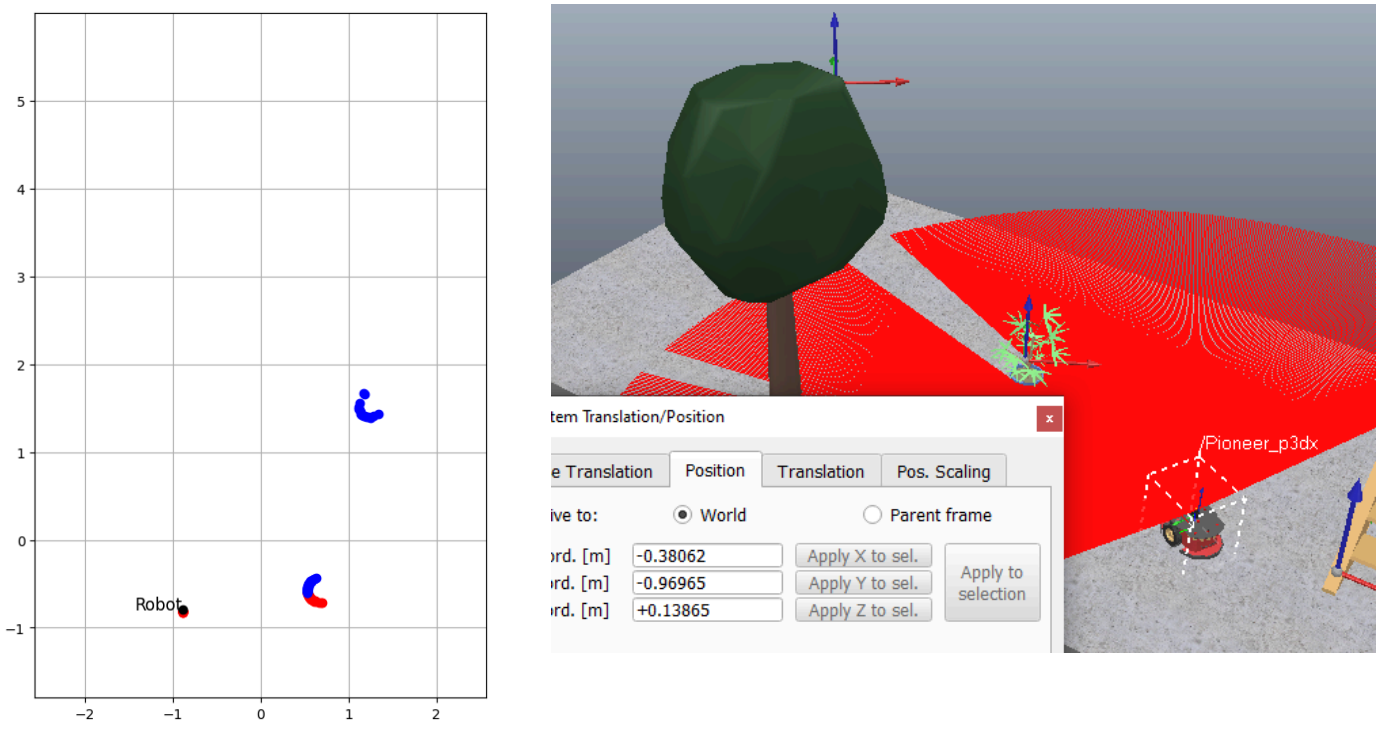
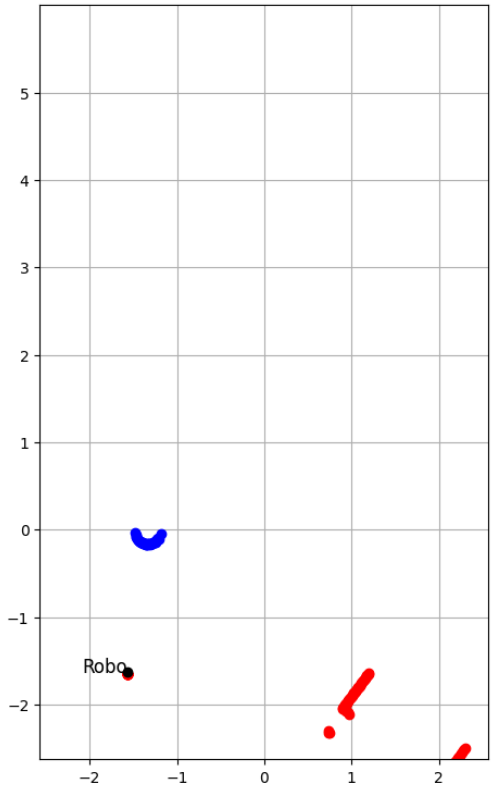


Fig 12 e 13 - Plotagem dos pontos no referencial global, de acordo com a posição atual do robô.

```
2 - Posição do robô: [-1.5719371578940122, -1.6264748443758719,
0.13865251722964786]
```



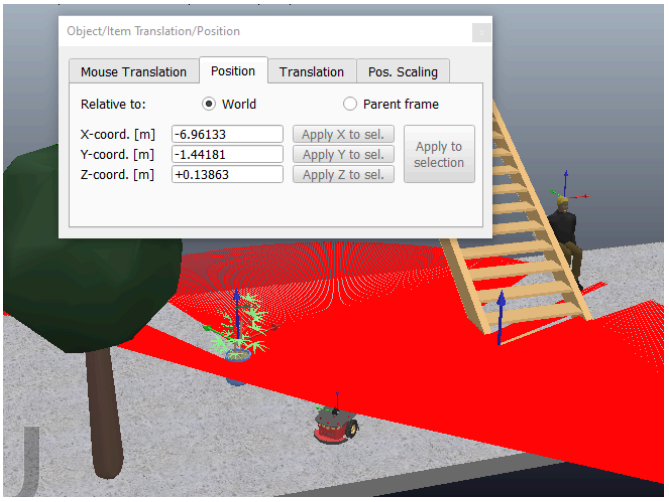
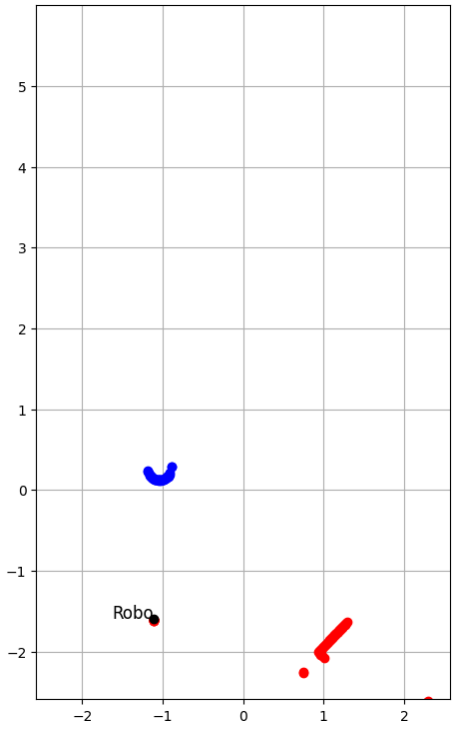


Fig 14 e 15 - Plotagem dos pontos no referencial global, de acordo com a posição atual do robô.

```
3 - Posição do robô: [-1.1144229685910476, -1.5914193556404905,
0.1386497339768365]
```



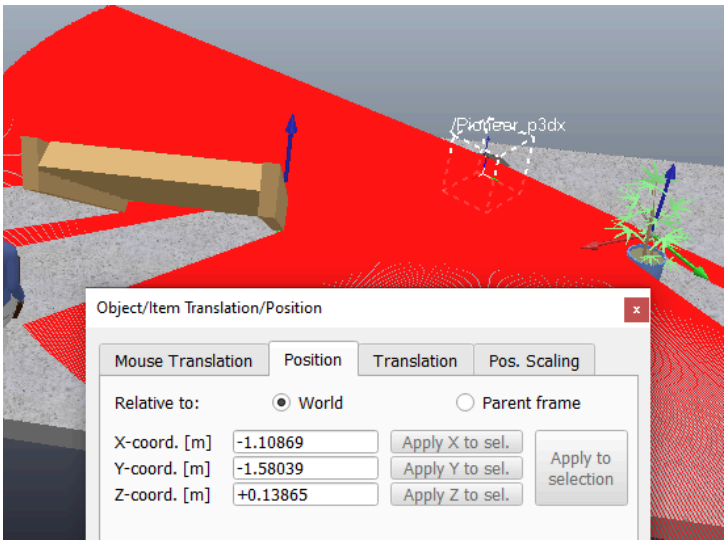


Fig 16 e 17 - Plotagem dos pontos no referencial global, de acordo com a posição atual do robô.

3.6 - Sequência de Posições

Por fim, para uma navegação básica, foi recuperado da API os handles para as rodas e setado uma velocidade em suas rodas para que o robô possa se locomover. Além disso, foi utilizado uma biblioteca chamada “keyboard”, para locomoção no mapa.

```
    # Handle para o ROBÔ
    laser_robot = sim.getObject('/Pioneer_p3dx')

    # Handle para as juntas das RODAS
    motorLeft = sim.getObject('/Pioneer_p3dx_leftMotor')
    motorRight = sim.getObject('/Pioneer_p3dx_rightMotor')

    # Armazenar a posição inicial do robô e dos pontos
    initial_robot_position = get_object_position(sim, '/Pioneer_p3dx')
    robot_trajectory.append(initial_robot_position)
    # definir o loop for por 10 segundos
    start_time = time.time()

    while time.time() - start_time < 10:
        if keyboard.is_pressed('w'):
            sim.setJointTargetVelocity(motorLeft, 2)
            sim.setJointTargetVelocity(motorRight, 2)
        elif keyboard.is_pressed('s'):
            sim.setJointTargetVelocity(motorLeft, -2)
            sim.setJointTargetVelocity(motorRight, -2)
        elif keyboard.is_pressed('a'):
            sim.setJointTargetVelocity(motorLeft, -2)
            sim.setJointTargetVelocity(motorRight, 2)
        elif keyboard.is_pressed('d'):
            sim.setJointTargetVelocity(motorLeft, 2)
            sim.setJointTargetVelocity(motorRight, -2)
        else:
            sim.setJointTargetVelocity(motorLeft, 0)
            sim.setJointTargetVelocity(motorRight, 0)
```

Para fazer o plot incremental também foi necessário alterar a função para plotagem das posições captadas pelo laser para o referencial global. A função agora itera sobre os dados do laser e, para cada ponto, verifica se a distância está dentro do alcance do sensor. Se estiver, a função transforma as coordenadas do laser da estrutura local do robô para a estrutura global utilizando uma matriz de rotação e traça o ponto nos eixos indicados.

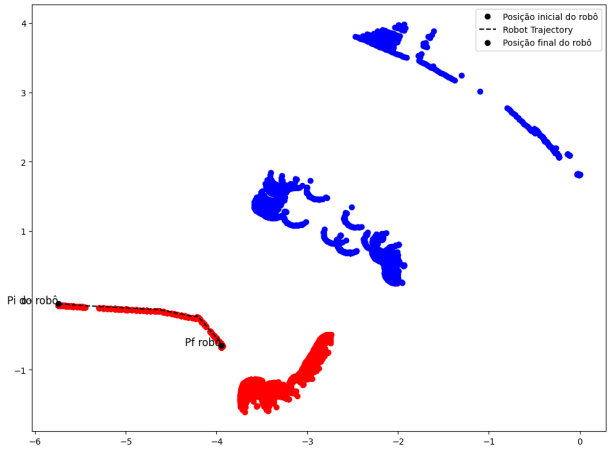
Outra necessidade foi armazenar as posições de todos os pontos de acordo com a movimentação do robô. Para isso, foi criado os seguintes dicionários:

```
# Armazenar as posições do robô e de todos os pontos ao longo da
navegação
robot_trajectory = []
```

```
points_trajectories = {point: [] for point in positions}  # Dicionário para armazenar as trajetórias de todos os pontos
```

Com isso é feito o plot indicando o deslocamento do robô de maneira mais detalhada e o deslocamento dos outros pontos utilizando a função de *draw_laser_data_global_positions*.

```
            # Armazenar a posição atual do robô
            current_robot_position = get_object_position(sim, '/Pioneer_p3dx')
            robot_trajectory.append(current_robot_position)

            # Prosseguindo com a leitura dos dados
            raw_range_data, raw_angle_data = readSensorData(clientID, laser_range_data, laser_angle_data)
            laser_data = np.array([raw_angle_data, raw_range_data]).T

            # Plotar os dados do sensor a cada iteração
            draw_laser_data_global_positions(laser_data, current_robot_position, laser_orientation, plt.gca())

        # Plotar um ponto como a posição inicial do robô
        plt.plot(initial_robot_position[0], initial_robot_position[1], 'o', color='black', label='Posição inicial do robô')
        plt.text(initial_robot_position[0], initial_robot_position[1], 'Pi do robô', fontsize=12, ha='right')

        # Plotar o caminho executado pelo robô
        robot_trajectory = np.array(robot_trajectory)
        plt.plot(robot_trajectory[:, 0], robot_trajectory[:, 1], linestyle='--', color='black', label='Robot Trajectory')

        # Plotar o último ponto do robô
        plt.plot(current_robot_position[0], current_robot_position[1], 'o', color='black', label='Posição final do robô')
        plt.text(current_robot_position[0], current_robot_position[1], 'Pf robô', fontsize=12, ha='right')

        plt.legend()
```



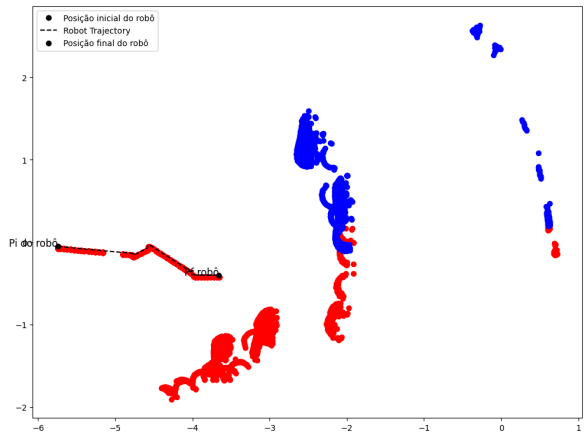


Fig 18 e 19 - Plotagem dos pontos no referencial global de acordo com a movimentação do robô

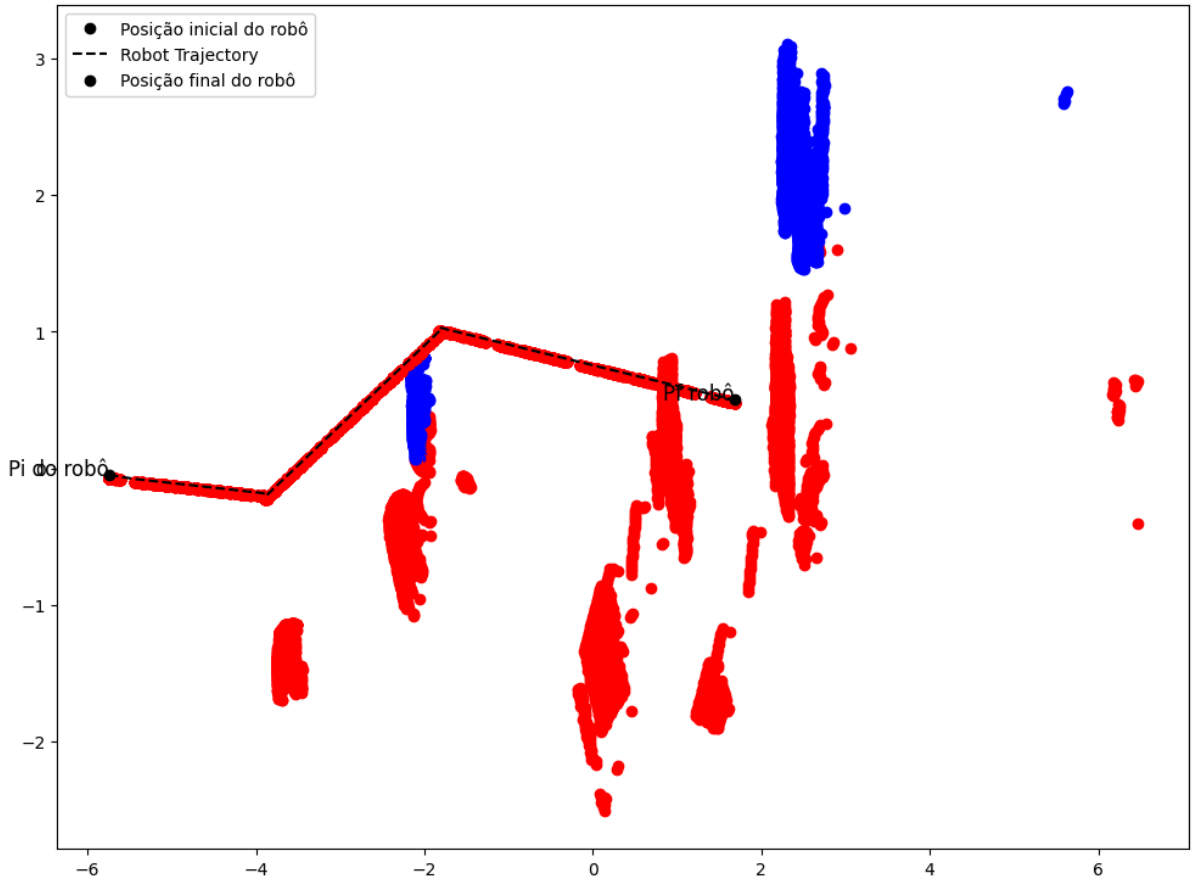


Fig 20 - Plotagem dos pontos no referencial global de acordo com a movimentação do robô realizada com 30s

4 - Conclusões

Por fim, o trabalho prático de robótica móvel desempenhou o principal papel de familiarização com o ferramental do curso. O aspecto teórico foi evidenciada pelo conteúdo abordado, destacando os conceitos de descrições espaciais e transformações homogêneas e compostas de maneira adequada. Em resumo, o trabalho prático foi de suma importância para a consolidação dos conceitos apresentados até o momento, bem como para sua aplicação prática por meio do uso do software do Coppelia.

5 - Bibliografia

Prof. Douglas G. Macharet - Robótica Móvel - Ferramental

Notações para Sistema de Coordenadas Geral - compas_fab - Coordinate frames

Documentação zmqRemoteApi - regular API reference

Prof. Douglas G. Macharet - Robótica Móvel - Descrição espacial e Transformações rígidas

Robótica con Python - Robótica con Python y CoppeliaSim - YouTube