# Outline “Tutorial MoveIt e Unity para manipulação robótica”

## Motivação

1. Moveit é um dos maiores frameworks de planejamento de movimento existente
    a. É usado em diversos robôs industriais e da academia
    b. Implementa diversos algoritmos de cinemática inversa, planejamento de trajetória etc
    c. Alta integração com ROS (mas pode ser executado stand-alone)
2. Unity é uma ferramenta para criação de jogos, mas que possui potencial promissor na robótica
    a. Permite criar simulações com alta riqueza visual
    b. Conta com uma alta quantidade de assets criados pela comunidade
    c. Compatibilidade com PhysX SDK 4 que conta com solver Temporal Gauss-Seidel para uma maior fidelidade da física
    d. Alta integração com outras ferramentas: ML-Agents, OpenAI Gym etc
    e. Não é uma questão de qual simulador é melhor, depende do caso de uso
        i. Estamos apenas adicionando mais uma ferramenta ao seu repertório
3. O objetivo deste tutorial é mostrar as principais etapas do processo de construção e configuração de uma simulação para um manipulador robótico
    a. Vamos partir de um arquivo CAD e chegar nisso (Exibir Unity e RVIZ)
    b. Apesar de estarmos utilizando um CAD de um manipulador comercial, o processo para um CAD desenvolvido por vcs é igual
    c. No tutorial vamos controlar um robô simulado, mas poderiamos (utilizando as mesmas ferramentas) controlar um robô físico

## Parte teórica

1. Apresentação dos conceitos básicos
    a. Manipulador

A estrutura mecânica do manipulador consiste numa sequência de corpos rígidos (links) interconectados por articulações (juntas). Um manipulador é caracterizado por um braço que garante mobilidade, um punho que dá destreza e um end-effector que performa a tarefa.

b. Graus de liberdade

Um grau de liberdade (*degree of freedom* ou DOF ) é uma das variáveis livres do sistema (mecânico, elétrico, térmico, etc) necessária para determinar o estado total do mesmo. No nosso caso, elas são manifestadas no formato de juntas. Os dois tipos essenciais de juntas são as de revolução e as prismáticas.

c. Links

Os "links" são as estruturas que conectam as juntas. Usualmente em manipuladores relativamente pequenos e lentos são considerados corpos rígidos e podem assumir os mais diferentes formatos.

d. Atuador

Atuadores são os dispositivos eletromecânicos responsáveis por causar uma mudança de estado em um dos graus de liberdade. Em outras palavras, são usualmente os motores elétricos ou pistões que movimentam as juntas.

Cada atuador possui uma curva de velocidade x torque, um limite de deslocamento e vários outros fatores que o projetista deve se atentar. Tipos diferentes de atuadores são recomendados para diferentes aplicações.

e. Cinemática Direta

Cinemática direta é o ato de calcular a posição e orientação do "end-effector" dado o valor de cada grau de liberdade (usualmente como ângulos e deslocamentos). Usualmente algo trivial de ser realizado para manipuladores seriais e mais complexo para paralelos.

f. Cinemática Inversa

Cinemática inversa é o ato de calcular o valor de cada grau de liberdade do sistema dado a posição e ângulo do "end-effector". Usualmente algo trivial de ser realizado para manipuladores paralelos e mais complexo para seriais.

g. Planejamento de trajetória

Ao utilizar um manipulador, usualmente deseja-se mover o *end-effector* do ponto "A" ao ponto "B", digamos. Contudo, em muitas aplicações, é necessário controlar também a posição do *end-effector;* e muitas vezes do manipulador inteiro; durante todo a trajetória entre "A" e "B". Isto é relevante, por exemplo, em ambientes restritos e cuja movimentação não controlada pode resultar em danos físicos.

A técnica tradicional para realizar este planejamento é discretizar a trajetória em vários pontos em posições desejadas e utilizar de cinemática inversa para calcular o valor de cada DOF em cada instante de tempo.

# Parte prática

2. Fusion 360
    a. Intro - Explicar um pouco sobre o Fusion 360
        i. Programa para Computer Assisted Design (CAD), com suporte a modelagem, simulação, integração com diversas ferramentas (EAGLE etc)
        ii. Gratuito para educadores (e para hobbistas com algumas restrições)
        iii. Fácil de usar
        iv. Como caso de estudo, utilizaremos o CAD do manipulador Kinova Gen 3 (disponível em https://www.kinovarobotics.com/sites/default/files/GEN3_7DOF_SPHERICAL_ARM_V11.zip)
    b. Meshes – Converter o CAD em um Mesh, para que possamos usar na simulação
        i. Mudar escala para metros (unidade padrão dos programas que vamos usar daqui pra frente)
        ii. Garantir que o projeto não está capturando histórico de edição (senão a aba **Mesh** não fica habilitada)
        iii. Usamos a ferramenta **Isolate** em cada componente (link) para facilitar a manipulação
        iv. Selecionamos o **Body** que vamos converter em **Mesh**
        v. Clique com o botão direito -> **BRep to Mesh**
        vi. Mesh criado. Clique com o botão direito -> **Find in Browser** (mais fácil do que ficar procurando o mesh na "mão")
        vii. Fazer isso para todos os links do robô
        viii. As vezes no processo de conversão, alguns componentes geram mais do que 1 Mesh. Para resolver isso basta combinar os Meshes gerados (Aba **Mesh-> Modify -> Merge Bodies**)
        ix. Feito isso teremos gerado o Mesh do manipulador
        x. Uma etapa importante do processo é garantir que todos os meshes possuam a mesma origem. Isso simplificará muito o processo de "montagem" dos links em simulação posteriormente

xi. Para isso ache o componente/mesh que possui a origem que vc deseja usar (geralmente a base). Devemos mover todos os outros meshes para a mesma pasta (Grupo) que esse mesh
xii. Clique com o botão direito e selecione **Move to Group** e em **Target Group** selecione o componente do passo anterior
xiii. Faça isso para o restante dos meshes. Ao final do processo todos deverão ter a mesma origem
xiv. Para exportar os objetos para formato **obj**, clique em cada mesh com o botão direito e selecione **Save As Obj**

c. Posição das juntas – Obtendo informação das juntas para configurar URDF e simulação do Unity
   i. Nessa etapa iremos obter a posição das juntas e seus eixos de rotação. Recomendo utilizar uma planilha para organizar as informações. Vamos preencher as coordenadas de mundo com as informações do Fusion e calcularemos as posições relativas das juntas através de uma simples subtração (posição relativa da junta i = posição absoluta da junta i - posição absoluta da junta i-1)
   ii. Primeira junta (fixa) possui coordenada (0,0,0) pois se encontra na origem. Representa a união da Base ao "mundo"
   iii. Segunda junta (revolução Base -> Shoulder1) possui coordenada na intersecção do eixo de rotação do Shoulder1 com a base.
   iv. Para determinar esse ponto, primeiro vamos criar o eixo clicando em **Construct -> Axis Through Cylinder/Cone/Torus** e clicando em um cilindro que seja concêntrico com o eixo
   v. Agora para determinar o ponto de intersecção, devemos clicar em **Construct -> Point at Edge and Plane** e clicar no eixo e na face da base.
   vi. O ponto criado será a origem da junta **Base_Shoulder1**
   vii. Utilizaremos a ferramenta **Measure** (Aba **Inspect -> Measure**) para determinar as coordenadas do ponto recém criado. Inserimos esses dados na nossa planilha
   viii. Devemos repetir esse procedimento para todas as juntas do nosso sistema
   ix. Algumas juntas não possuem faces planares e, portanto, não conseguimos usar a ferramenta **Construct -> Point at Edge and Plane**. Para solucionar isso devemos primeiro criar um plano que toca essa face através da ferramenta **Construct -> Offset Plane** com **distância 0** (importante!)
   x. Feito isso conseguimos usar a ferramenta **Construct -> Point at Edge and Plane**
   xi. Coletar as coordenadas de todas as juntas

3. Moveit

a. Criação do Xacro/URDF – Nessa etapa iremos gerar o URDF (unified robotic description file) para o nosso manipulador. Ele basicamente é uma forma padronizada de descrever as informações que coletamos na etapa anterior. Vamos gerar o URDF a partir de um arquivo XACRO, que nos permite algumas facilidades na hora de descrever o robô
    i. Eu geralmente gosto de criar um workspace do catkin para cada projeto, mas isso fica ao critério de cada um
    ii. Criaremos um pacote do ROS para armazenar os meshes e o URDF
        1. Dentro da ws de sua preferência: **catkin_create_pkg kinovagen3_description**
        2. Transferir todos os meshes para a pasta **kinovagen3_description/meshes (TAG 3.a.ii.2)**
    iii. Criar o arquivo **kinovagen3_description/urdf/kinovagen3.xacro**
        1. Adicionar um **link** de nome **base** e adicionar as tags: <visual>

```
<geometry>
  <mesh filename="package://kinovagen3_description/meshes/Base.obj"/>
</geometry>
<origin rpy="0 0 0" xyz="0 0 0"/>
</visual>
```

    iv. Criar um launch file para visualizar o arquivo Xacro. Copiar o launch file **display.launch** do pacote **urdf_tutorial**, substituindo a tag **model** por: <arg name="model" default="$(find kinovagen3_description)/urdf/kinovagen3.xacro"/>
    v. Comentar a tag **rvizconfig** e substituir a tag **rviz** por <node name="rviz" pkg="rviz" type="rviz" required="true" />
    vi. Iremos criar o arquivo de configuração do rviz:
        1. Rodar o arquivo display.launch
        2. Vc deverá ver uma mensagem de warning no Global Status. No momento ela não é importante.
        3. Digite **base** em **Global Options/Fixed Frame**
        4. Clique no botão **Add** e adicione a componente **rviz/RobotModel**
        5. Vc deverá ver a base do robô na cena
        6. Clique em **File/Save Config As** e salve a configuração como **kinovagen3_description/urdf/config.rviz**
        7. Mate o processo do rviz e sete a tag **rvizconfig** para: <arg name="rvizconfig" default="$(find kinovagen3_description)/urdf/config.rviz" /> e a tag **rviz** para: <node name="rviz" pkg="rviz" type="rviz" args="-d $(arg rvizconfig)" required="true" />

8. Ao rodar o arquivo display.launch novamente, vc deverá ver o modelo do robô na cena e a propriedade **Fixed Frame** com valor **base (TAG 3.a.vi.8)**

vii. Considerações para os **links**:
   1. Na tag **visual**, a opção **xyz** da tag **origin** de um link sempre será a posição absoluta (em coordenadas de mundo) negada da junta que o liga ao seu link pai. Por exemplo, o link **shoulder1** possui link pai **base**. Eles são ligados pela junta **base_shoulder1** que possui coordenadas de mundo (0, 0, 0.156). Logo o link **shoulder1** terá a seguinte tag **origin**:
      <origin rpy="0 0 0" xyz="0 0 -0.156"/>

      Outro exemplo: o link **halfarm1** é ligado ao link **shoulder1** através da junta **shoulder1_halfarm1**. Essa possui coordenadas de mundo (0, -0.005, 0.285). Logo o link **halfarm1** terá a seguinte tag **origin**:
      <origin rpy="0 0 0" xyz="0 0.005 -0.285"/>

viii. Considerações para as **joints**:
   1. Sua origem será sempre a sua posição relativa à ultima junta. Ex:
      base_shoulder1: <origin rpy="0 0 0" xyz="0 0 0.156"/>
      shoulder1_halfarm1: <origin rpy="0 0 0" xyz="0 -0.005 0.129"/>
   2. O eixo de rotação depende do design do manipulador, firmware etc. Como estamos utilizando um manipulador comercial, os eixos de rotação podem ser encontrados na página 48 do manual de referência: https://www.kinovarobotics.com/sites/default/files/UG-014_KINOVA_Gen3_Ultra_lightweight_robot-User_guide_EN_R01.pdf
   3. Os limites das juntas, velocidade e torque também podem ser encontrados no manual. Para esse tutorial apenas os limites de ângulo importam **(TAG 3.a.viii.3)**

b. Moveit setup assistant:
   i. Rodar no terminal:
      roslaunch moveit_setup_assistant setup_assistant.launch
   ii. “Create New Package” -> Selecionar arquivo Xacro
   iii. Configuraremos o nosso robô conforme o tutorial do próprio moveit: http://docs.ros.org/en/melodic/api/moveit_tutorials/html/doc/setup_assistant/setup_assistant_tutorial.html
   iv. Gerar self-collision matrix
   v. Criar virtual joint **virtual_joint** que liga **base** à world

vi. Criar Planning group para juntas virtual_joint, base_shoulder1, shoulder1_halfarm, ... , sphericalwirst2_eecoupler (nome: kinovagen3_arm, solver: kdl_kinematics_plugin/KDLKinematicsPlugin)
vii. Criar Planning group para link eecoupler (nome: ee). Adicionar apenas o link eecoupler
viii. Criar pose é opcional. Exemplo: todas as juntas na posição 0 -> nome: upright
ix. Criar end effector a partir do grupo "ee" (nome: kinovagen3_ee, base link: eecoupler)
x. Não há juntas passivas
xi. Criar ROS Control
    1. Nome: arm_position_controller
    2. Tipo: position_controllers/JointPositionController
    3. Planning group: kinovagen3_arm
xii. Simulation: Pular
xiii. 3D perception: Pular
xiv. Author Information: preencher
xv. Configuration files: criar pasta "kinovagen3_moveit_config" dentro do src da workspace catkin onde criamos o pacote kinovagen3_description e seleciona-la
xvi. Gerar os arquivos de configuração e rodar rospack profile para que o ROS carregue o pacote criado
xvii. Testar com:
roslaunch kinovagen3_moveit_config demo.launch
    **1.** Selecionar na aba Planning -> **Query -> Planning group --> kinovagen3_arm**
    **2.** Start State: <current>
    **3.** Goal State: <random valid>
    **4.** Clicar em **Plan and Execute (TAG 3.b.xvii.4)**

c. Controler:
i. Criar **kinovagen3_moveit_config/config/controllers.yaml** conforme modelo do site http://docs.ros.org/en/melodic/api/moveit_tutorials/html/doc/controller_configuration/controller_configuration_tutorial.html
ii. Editar **kinovagen3_moveit_config/config/ros_controllers.yaml** no campo **controller_list** e adicionar o conteúdo de **controllers.yaml**
iii. Criar uma cópia de **kinovagen3_moveit_config/launch/demo.launch** e renomea-la para **controller_demo.launch**
    1. Remover nó **joint_state_publisher**

2. Setar **fake_execution** como **false (TAG 3.c.iii.2)**

4. Unity
    a. Carregar meshes
        i. Abrir o Unity Hub e criar um projeto novo
        ii. Inserir um plano, adicionar um rigid body, desativar gravidade e habilitar kinematic
        **iii.** Clicar com o botão direito nos assets > **Import New Asset** e selecionar todos os **.obj** criados na etapa do Fusion
        **iv.** Arrastar um objeto de cada vez para a cena
        v. Selecionar todos dentro da cena e setar sua posição para (0, 0, 0). O braço deverá ficar "montado", orientado com o eixo Z (ficará dentro do plano)
        vi. Ainda com todos os objetos selecionados, setar rotação no eixo X como –90. O braço agora deverá ficar alinhado com o eixo Y
        vii. Por fim, com os objetos selecionados, clique em **Unpack Prefab Completelly**. Apesar desse passo não ser obrigatório, ajuda a limpar um pouco os objetos. Expanda todos os objetos e clique nos seus filhos e arraste para a raiz do explorador de objetos.
        viii. Delete os objetos pai. Nesta etapa deve restar apenas os objetos que possuem Mesh de fato
        ix. Crie um **Game Object** vazio na posição (0, 0, 0), renomeio como **Arm** e adicione todos os links como filhos dele
        x. Selecione a base e adicione uma fixed joint, setando como connected body o plano criado
        xi. Selecione os links restantes e adicione uma hinge joint
        **xii.** Nesse ponto é interessante que os links estejam em ordem na aba **Hierarchy** (Base, Shoulder1, Halfarm1, Halfarm2, Forearm1, Sphericalwrist1, Sphericalwrist2, EECoupler). A próxima etapa ficará mais fácil **(TAG 4.a.xii)**
    b. Configuração das hinge joints:
        i. Abra a planilha que criamos no fusion
        ii. Começando pelo shoulder1, adicione o link anterior como **Connected Body**. Nesse caso seria o link **Base**.
        iii. Para configuração da posição da ancora, iremos colocar a **posição de mundo das joints**. Note que as coordenadas do **Anchor** estão no sistema de coordenada do link em que essa junta se encontra. Se tudo ocorreu como esperado até aqui, todos os links devem estar orientados com mesmo Y e Z do URDF (use o RViz para visualizar) e com X invertido. Como todos os links possuem X=0, não precisaremos mudar as coordenadas ao configurar as hinge joints

iv. Colocar a visualização como **Wireframe** ajuda a observar como a ancora está orientada/posicionada (o vetor laranja)
v. Note que essa hinge joint liga a Base ao Shoulder1. Portanto devemos buscar na tabela informações sobre a junta Base_Shoulder1.
vi. No campo **Axis**, vc deve colocar 1 com o eixo em que vc deseja alinhar esse vetor. Para essa junta o eixo é o Z. Note que o Unity usa **a regra da mão esquerda. Portanto os eixos de rotação são negados quando comparados ao xacro/urdf**
**vii.** Faça isso para todas as juntas **(TAG 4.b.vii)**
viii. Selecione todos os links menos a base e marque a opção **Use Spring**, setando:
    1. **Spring** = 500
    2. **Damper** = 250
    3. **Target Position** = 0
ix. Essa opção é basicamente um controlador PD, onde P é **Spring** e D é o **Damper**. Idealmente esses valores precisam ser ajustados mais precisamente, mas para esse tutorial esses valores bastam
x. Outro elemento que devemos preencher com cuidado é a massa de cada link. Retirado da página 138 do manual: https://www.kinovarobotics.com/sites/default/files/UG-014_KINOVA_Gen3_Ultra_lightweight_robot-User_guide_EN_R01.pdf
    a. Base: 1.697kg
    b. Shoulder1: 1.377kg
    c. Halfarm1: 1.164kg
    d. Halfarm2: 1.164kg
    e. Forearm1: 0.93kg
    f. SphericalWrist1: 0.678kg
    g. SphericalWrist2: 0.678kg
    h. Vision(EECoupler): 0.500kg
xi. Selecione todos os links e adicione um Mesh Collider com a opção convex habilitada
xii. Clique em **Play** para começar a simulação e com os links que possuem hinge joints selecionados coloque a target position como 45. O braço deverá se movimentar
xiii. Perceba que no início do movimento o braço parece balançar um pouco, parecendo até meio “mole”. Isso é bem perceptível na junta Base_Shoulder1 ao setar o ângulo das juntas de 45 para 0

**xiv.** Isso ocorre por causa do tipo de solver utilizado pela engine física. Podemos melhorar muito o desempenho trocando o solver para o **Temporal Gauss Seidel** em **Edit -> Physics -> Solver Type: Temporal Gauss Seidel**

**xv.** Para obter um desempenho melhor podemos balancear a massa de cada link. Mas para o nosso tutorial, o desempenho atual é suficiente **(TAG 4.b.xv)**

**c.** Instalando ROS#, e criando um publisher para publicar o ângulo das juntas:

**i.** Primeiramente vamos criar um script que lê as informações de cada junta

1. Ele deve ser responsável por setar a posição da junta, ler valores como velocidade, aceleração etc, e indicar se a junta atingiu sua posição alvo
2. É importante para abstrair algumas rotinas/informações pertinentes a cada junta. Isso deixa o código mais organizado e auto-contido (facil de testar, debugar etc)
3. Nome **ManipulatorJoint (TAG 4.c.i.3)**

ii. Agora vamos criar o script que gerencia (controla) o manipulador, como um todo

1. Nome: **ArmController**
2. Deve armazenar uma referência para cada junta, bem como o nome de cada junta
3. Deve ser capaz de receber um array de posições e setar o manipulador nessa posição
4. Deve ser capaz de fornecer informações de uma junta a partir do seu nome
5. Deve ser capaz de informar se o braço chegou na posição alvo **(TAG 4.c.ii.5)**

iii. Para as próximas etapas, vamos instalar o RosSharp no projeto conforme descrito no link https://github.com/siemens/ros-sharp/wiki/User_Inst_Unity3DOnWindows

1. Basicamente copiar a pasta RosSharp para a pasta Assets do seu projeto e se certificar que o Unity está usando o runtime 4.0 **(TAG 4.c.iii.1)**

iv. Adicionar script **RosConnector** ao braço. Esse script fará a conexão com o ROS através do pacote **rosbridge**

v. Finalmente criaremos o script que de fato criará um publicará no tópico /joint_states

1. Nome: **PublishJointStates**

2. O script deverá publicar uma mensagem do tipo JointState (http://docs.ros.org/en/melodic/api/sensor_msgs/html/msg/JointState.html) no tópico /joint_states
3. Essa mensagem contém nome, posição, velocidade e torque das juntas

**vi.** Iniciar **roscore** e **rosbridge (roslaunch rosbridge_server rosbridge_websocket.launch)**

vii. Iniciar demo de visualização no rviz (**roslaunch kinovagen3_moveit_config controller_demo.launch**)

viii. Especificar ip da máquina e porta 9090 no campo **Ros Bridge Server Url** do script **RosConnector**

**ix.** Setar posição das juntas no Unity e observer que o braço se move no Rviz **(TAG 4.c.ix)**

d. Criando um action-server (http://wiki.ros.org/actionlib, http://wiki.ros.org/actionlib/DetailedDescription ) para requisições do tipo FollowJointTrajectory

i. Primeiramente, devemos criar as actions e mensagens que vamos utilizar

1. Crie duas Pastas, **Actions** e **Messages**, na pasta **Assets** do seu projeto
2. Crie os arquivos **Actions/FollowJointTrajectory.action** e **Messages/JointTolerance.msg**
3. Popule esses arquivos com os conteúdos de http://docs.ros.org/en/api/control_msgs/html/action/FollowJointTrajectory.html e http://docs.ros.org/en/diamondback/api/control_msgs/html/msg/JointTolerance.html respectivamente
4. Clique em **RosBridgeClient>Auto Generate Messages>Single Message** e selecione o arquivo **JointTolerance.msg**
    a. Preencha a opção **ROS Package Name** com **control_msgs**
5. Clique em **RosBridgeClient>Auto Generate Actions >Single Action** e selecione o arquivo **FollowJointTrajectory.action**
    a. Preencha a opção **ROS Package Name** com **control_msgs**
6. Os arquivos de saída podem ser encontrados na pasta **RosSharpMessages>Control**
7. Criar dois scripts, **UnityKinovaGen3ActionSever** que representará o script unity que será atribuído a um objeto e instanciará o **KinovaGen3ActionSever** que é a implementação do ActionServer de fato **(TAG 4.d.i.7)**