Geração de Trajetórias no ROS 2

Walter Fetter Lages

fetter@ece.ufrgs.br

Universidade Federal do Rio Grande do Sul

Escola de Engenharia

Departamento de Sistemas Elétricos de Automação e Energia

ENG10052 Laboratório de Robótica

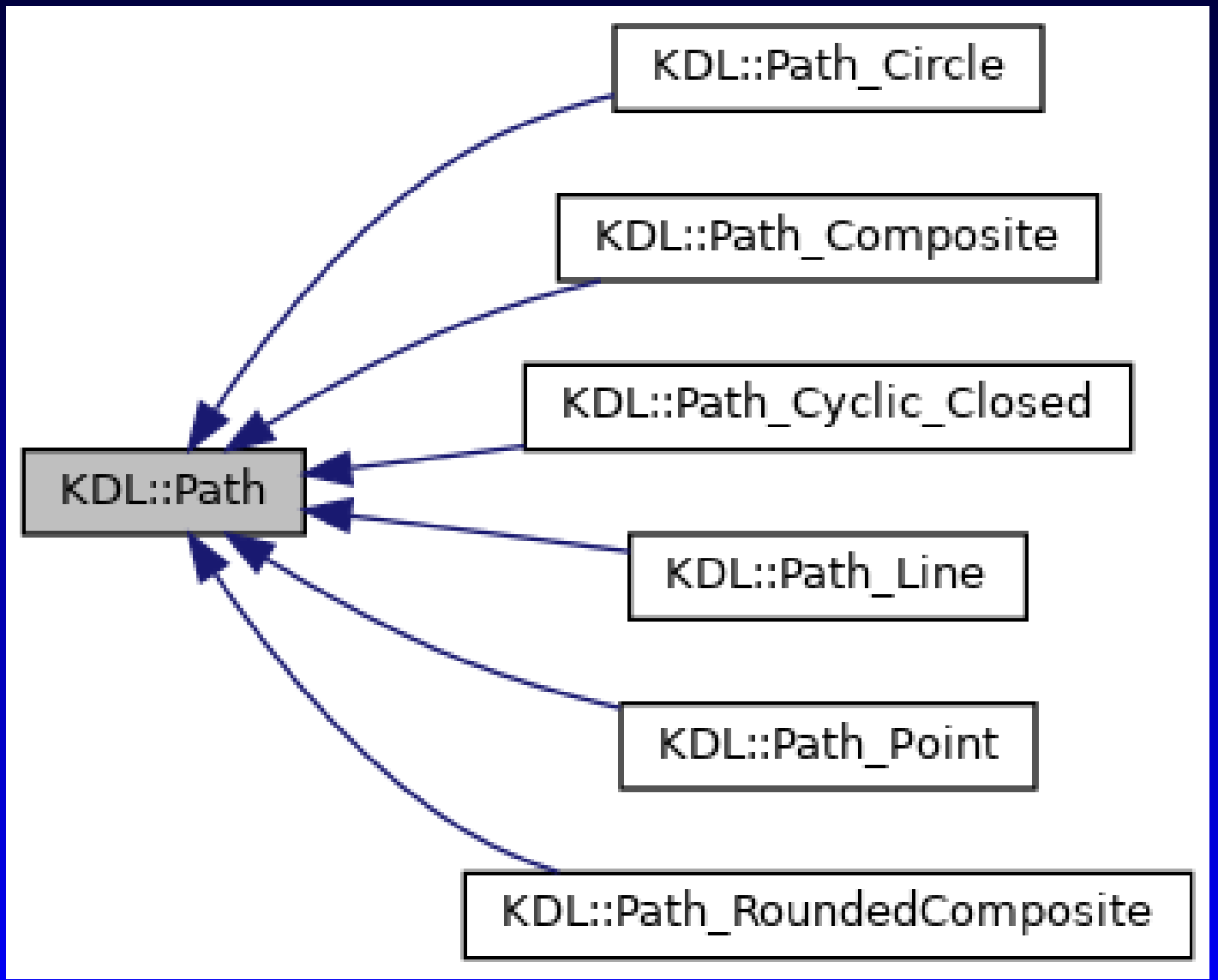
UFRGS
UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL

Introdução

- Implementada na biblioteca KDL (*Kinematics and Dynamics Library*)
- Implementa classes para especificar:
  - Caminhos
  - Perfis de velocidade
  - Trajetórias de *frames*
    - Trajetória no espaço cartesiano
  - Trajetórias de cadeias cinemáticas
    - Trajetória no espaço das juntas

Classe KDL::Path

- Representa um caminho no espaço cartesiano



UFRGS
UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL

Classe KDL::Path

```
class Path {
  public:
  enum IdentifierType { ID_LINE =1,ID_CIRCLE =2,
    ID_COMPOSITE =3, ID_ROUNDED_COMPOSITE =4,
    ID_POINT =5, ID_CYCLIC_CLOSED =6 };
  virtual double LengthToS(double length)=0;
  virtual double PathLength()=0;
  virtual Frame Pos(double s) const=0;
  virtual Twist Vel(double s,double sd) const=0;
  virtual Twist Acc(double s,double sd,double sdd) const=0;
  virtual void Write(std::ostream &os)=0;
  static Path *Read(std::istream &is);
  virtual Path *Clone()=0;
  virtual IdentifierType getIdentifier() const=0;
  virtual ~Path() {};
};
```

UFRGS
UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL

KDL::Path_Line

- Implementa um caminho em linha reta

```
class Path_Line:public Path
{
  public:
  Path_Line(const Frame &F_base_start,const Frame &F_base_end,
    RotationalInterpolation *orient,double eqradius,
    bool _aggregate=true);
  Path_Line(const Frame &F_base_start,const Twist &twist_in_base,
    RotationalInterpolation *orient,double eqradius,
    bool _aggregate=true);
  virtual ~Path_Line();
};
```

UFRGS
UNIVERSIDADE FEDERAL
DO RIO GRANDE DO SUL

KDL::Path_Point

- Implementa um caminho pontual

```
class Path_Point:public Path
{
  public:
  Path_Point(const Frame& F_base_start);
  virtual ~Path_Point();
};
```

KDL::Path_Circle

- Implementa um caminho em arco de circunferênca

```
class Path_Circle:public Path
{
  public:
  Path_Circle(const Frame &F_base_start,const Vector &
      V_base_center,
    const Vector &V_base_p,const Rotation &R_base_end,
    double alpha,RotationalInterpolation *otraj,
    double eqradius,bool _aggregate=true);
  virtual ~Path_Circle();
};
```

UFRGS
UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL

KDL::Path_Composite

- Permite a formação de caminhos compostos

```
class Path_Composite:public Path
{
  public:
  Path_Composite();
  void Add(Path *geom,bool aggregate=true);
  virtual int GetNrOfSegments();
  virtual Path *GetSegment(int i);
  virtual double GetLengthToEndOfSegment(int i);
  virtual void GetCurrentSegmentLocation(double s,int &
     segment_number,double &inner_s);
  virtual ~Path_Composite();
};
```

UFRGS
UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL

KDL::Path_Cyclic_Closed

- Permite a formação de caminhos fechados e repetitivos

```
class Path_Cyclic_Closed:public Path
{
  public:
  Path_Cyclic_Closed(Path *_geom,int _times,bool _aggregate=true);
  virtual ~Path_Cyclic_Closed();
};
```

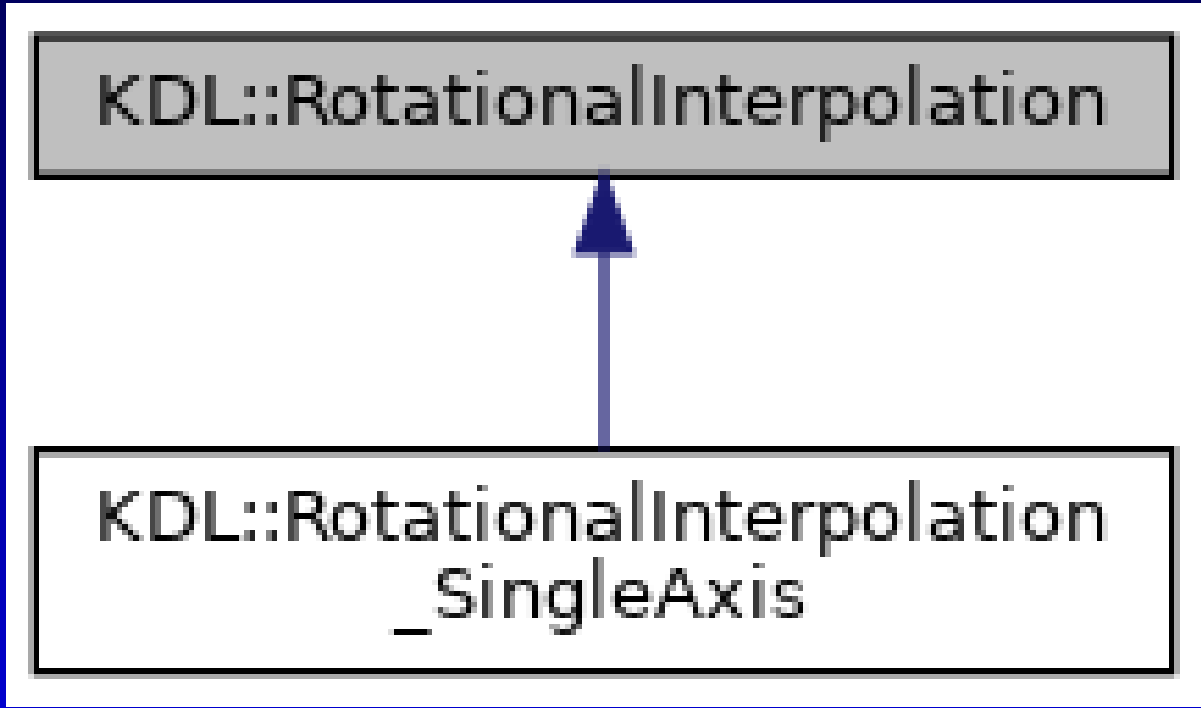
KDL::Path_RoundedComposite

- Caminho composto com transições suaves

```
class Path_RoundedComposite:public Path
{
  public:
  Path_RoundedComposite(double radius,double eqradius,
     RotationalInterpolation *orient,bool aggregate=true);
  void Add(const Frame &F_base_point);
  void Finish();
  virtual int GetNrOfSegments();
  virtual Path *GetSegment(int i);
  virtual double GetLengthToEndOfSegment(int i);
  virtual void GetCurrentSegmentLocation(double s,int &
     segment_number,double &inner_s);
  virtual ~Path_RoundedComposite();
};
```

KDL::RotationalInterpolation

- Especifica a parte rotacional de um caminho no espaço cartesiano



UFRGS
UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL

KDL::RotationalInterpolation

```
class RotationalInterpolation
{
  public:
  virtual void SetStartEnd(Rotation start,Rotation end)=0;
  virtual double Angle()=0;
  virtual Rotation Pos(double theta) const=0;
  virtual Vector Vel(double theta,double thetad) const=0;
  virtual Vector Acc(double theta,double thetad,double thetadd)
     const=0;
  virtual void Write(std::ostream &os) const=0;
  static RotationalInterpolation *Read(std::istream &is);
  virtual RotationalInterpolation *Clone() const=0;
  virtual ~RotationalInterpolation();
};
```

UFRGS
UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL

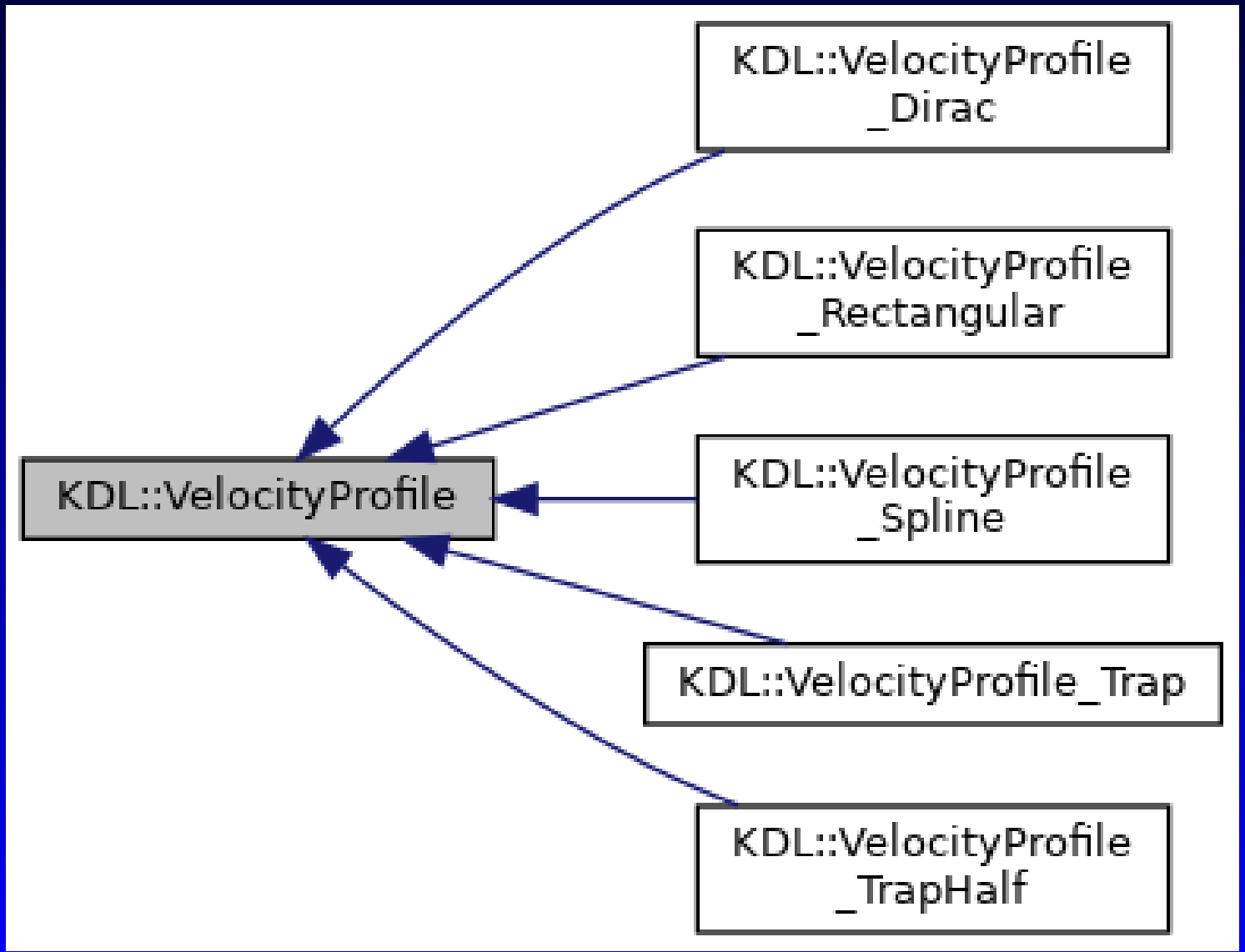
KDL::RotationalInterpolation_SingleAxis

- Interpolação através da rotação em torno de um único eixo

```
class RotationalInterpolation_SingleAxis: public
      RotationalInterpolation
{
  public:
  RotationalInterpolation_SingleAxis();
  virtual ~RotationalInterpolation_SingleAxis();
};
```

Classe `KDL::VelocityProfile`

- Representa um perfil de velocidade



UFRGS
UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL

KDL::VelocityProfile

```
class VelocityProfile
{
  public:
  virtual void SetProfile(double pos1,double pos2)=0;
  virtual void SetProfileDuration(double pos1,double pos2,double
      duration)=0;
  virtual double Duration() const=0;
  virtual double Pos(double time) const=0;
  virtual double Vel(double time) const=0;
  virtual double Acc(double time) const=0;
  virtual void Write(std::ostream &os) const=0;
  static VelocityProfile *Read(std::istream &is);
  virtual VelocityProfile *Clone() const=0;
  virtual ~VelocityProfile();
};
```

UFRGS
UNIVERSIDADE FEDERAL
DO RIO GRANDE DO SUL

KDL::VelocityProfile_Dirac

- Perfil de velocidade delta de Dirac

```
class VelocityProfile_Dirac:public VelocityProfile
{
  public:
  virtual ~VelocityProfile_Dirac();
};
```

UFRGS
UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL

KDL::VelocityProfile_Rectangular

- Perfil de velocidade retangular

```
class VelocityProfile_Rectangular:public VelocityProfile
{
  public:
  VelocityProfile_Rectangular(double _maxvel=0);
  void SetMax(double _maxvel);
  virtual ~VelocityProfile_Rectangular();
};
```

UFRGS
UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL

KDL::VelocityProfile_Spline

- Perfil de velocidade em *spline*
  - Posição é um polinômio de grau 5

```
class VelocityProfile_Spline:public VelocityProfile
{
  public:
  VelocityProfile_Spline();
  VelocityProfile_Spline(const VelocityProfile_Spline &p);
  virtual ~VelocityProfile_Spline();
  virtual void SetProfileDuration(double pos1, double vel1, double
     pos2, double vel2, double duration);
  virtual void SetProfileDuration(double pos1, double vel1, double
     acc1, double pos2, double vel2, double acc2, double duration);
};
```

UFRGS
UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL

KDL::VelocityProfile_Trap

- Perfil de velocidade trapezoidal

```
class VelocityProfile_Trap:public VelocityProfile
{
  public:
  VelocityProfile_Trap(double _maxvel=0,double _maxacc=0);
  virtual void SetProfileVelocity(double pos1,double pos2,double
     newvelocity);
  virtual void SetMax(double _maxvel,double _maxacc);
  virtual ~VelocityProfile_Trap();
};
```

UFRGS
UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL

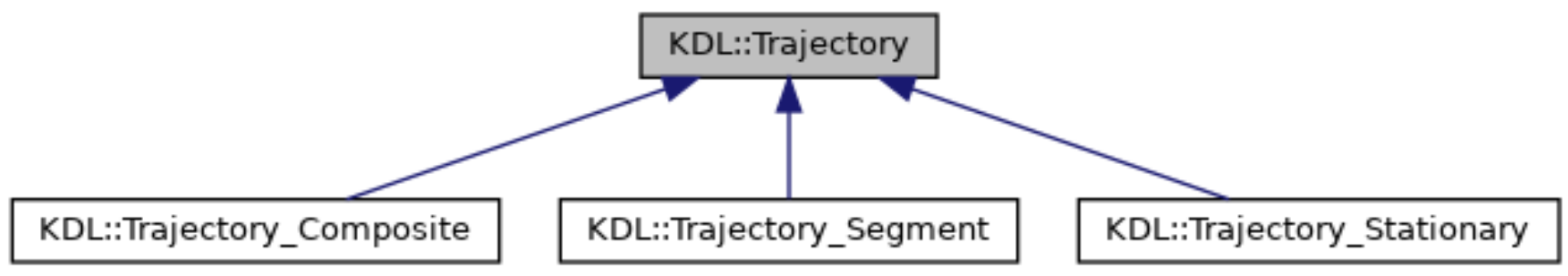
KDL::VelocityProfile_TrapHalf

- Perfil de velocidade trapezoidal pela metade
  - Só o início ou só o final

```
class VelocityProfile_TrapHalf:public VelocityProfile
{
  public:
  VelocityProfile_TrapHalf(double _maxvel=0,double _maxacc=0,
     bool _starting=true);
  void SetMax(double _maxvel,double _maxacc,bool _starting);
  virtual ~VelocityProfile_TrapHalf();
};
```

Classe `KDL::Trajectory`

- Implementa uma trajetória cartesiana



UFRGS
UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL

KDL::Trajectory

```
class Trajectory
{
  public:
  virtual double Duration() const=0;
  virtual Frame Pos(double time) const=0;
  virtual Twist Vel(double time) const=0;
  virtual Twist Acc(double time) const=0;
  virtual Trajectory *Clone() const=0;
  virtual void Write(std::ostream &os) const=0;
  static Trajectory *Read(std::istream &is);
  virtual ~Trajectory();
};
```

UFRGS
UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL

KDL::Trajectory_Composite

- Implementa uma trajetória cartesiana composta
- Os elementos adicionados com `Add()` são destruídos pelo destrutor da classe.
  - Devem ser alocados dinamicamente

```
class Trajectory_Composite:public Trajectory
{
  public:
  Trajectory_Composite();
  virtual void Add(Trajectory *elem);
  virtual void Destroy();
  virtual ~Trajectory_Composite();
};
```

UFRGS
UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL

KDL::Trajectory_Segment

- Implementa um segmento de trajetória cartesiana

```
class Trajectory_Segment:public Trajectory
{
  public:
  Trajectory_Segment(Path *_geom,VelocityProfile *_motprof,bool
      _aggregate=true);
  Trajectory_Segment(Path *_geom,VelocityProfile *_motprof,double
      duration,bool _aggregate=true);
  virtual Path *GetPath();
  virtual VelocityProfile *GetProfile();
  virtual ~Trajectory_Segment();
};
```

UFRGS
UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL

KDL::Trajectory_stationary

- "Trajetória" cartesiana com o robô parado

```
class Trajectory_Stationary:public Trajectory
{
  public:
  Trajectory_Stationary(double _duration,const Frame &_pos);
  virtual ~Trajectory_Stationary();
};
```

Exemplo - Trajetória Cartesiana

- Caminho especificado através de *via points*
  - $(0.61, 0, 0.1477)$, $(0.437, -0.424, 0.1477)$, $(0.238, -0.505, 0.1477)$, $(0.437, 0.424, 0.1477)$, $(0.238, 0.505, 0.1477)$, $(0.61, 0, 0.1477)$
- Perfil de velocidade trapezoidal
  - Velocidade máxima 0.1 $\mathrm{m/s}$
  - Aceleração 0.02 $\mathrm{m/s^2}$
- Trajetória fica parada 1 s no ponto final
- Discretização de 0.1 s
- Publica mensagem `geometry_msgs/msg/PoseStamped`

UFRGS
UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL

geometry_msgs/msg/PoseStamped

std_msgs/Header header

builtin_interfaces/Time stamp

int32 sec

uint32 nanosec

string frame_id

Pose pose

Point position

float64 x

float64 y

float64 z

Quaternion orientation

float64 x 0

float64 y 0

float64 z 0

float64 w 1

UFRGS
UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL

Pacote `eng10026_trajectories`

- Criar o pacote:

```
cd ~/colcon_ws/src
ros2 pkg create --build-type ament_cmake --dependencies
    rclcpp tf2_kdl geometry_msgs trajectory_msgs orocos_kdl
    --node-name pose_trajectory_publisher
    eng10026_trajectories
```

- `package.xml` deve ser editado para configurar os detalhes de documentação e incluir dependências

UFRGS
UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL

CMakeLists.txt

- Editar `CMakeLists.txt` para descomentar e ajustar as *tags*:

```
add_executable(pose_trajectory_publisher
  src/pose_trajectory_publisher.cpp)

ament_target_dependencies(pose_trajectory_publisher
  rclcpp tf2_kdl geometry_msgs trajectory_msgs orocos_kdl)

install(TARGETS pose_trajectory_publisher
  DESTINATION lib/${PROJECT_NAME})

install(DIRECTORY launch rviz
   DESTINATION share/${PROJECT_NAME})
```

Inclusão no Meta-Pacote

- O pacote `eng10026_trajectories` será incluido no meta-pacote `eng10026`
- Editar o arquivo `package.xml` do pacote `eng10026` e incluir

<run_depend>eng10026_trajectories</run_depend>

---

pose_trajectory_publisher.cpp

```
#include <rclcpp/rclcpp.hpp>

#include <kdl/path_roundedcomposite.hpp>
#include <kdl/rotational_interpolation_sa.hpp>
#include <kdl/velocityprofile_trap.hpp>
#include <kdl/trajectory_segment.hpp>
#include <kdl/trajectory_stationary.hpp>
#include <kdl/trajectory_composite.hpp>
#include <kdl/utilities/error.h>
#include <tf2_kdl/tf2_kdl.h>

#include <geometry_msgs/msg/pose_stamped.hpp>

using namespace KDL;
```

pose_trajectory_publisher.cpp

```
class PoseTrajectory: public rclcpp::Node
{
  public:
  PoseTrajectory(const char *name="trajectory_publisher");

  private:
  Trajectory_Composite trajectory_;
  double t0_;
  rclcpp::TimerBase::SharedPtr timer_;
  rclcpp::Publisher<geometry_msgs::msg::PoseStamped>::SharedPtr
      posePublisher_;

  void timerCB(void) const;
};
```

UFRGS
UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL

pose_trajectory_publisher.cpp

```
PoseTrajectory::PoseTrajectory(const char *name): Node(name)
{
  try
  {
    auto path=new Path_RoundedComposite(0.02,0.002,new
     RotationalInterpolation_SingleAxis());
    path->Add(Frame(Rotation::RPY(0,0,0),Vector(0.61,0,0.1477)));
    path->Add(Frame(Rotation::Quaternion(0,0,-0.375,0.926),Vector
     (0.437,-0.424,0.1477)));
    path->Add(Frame(Rotation::RotZ(-M_PI_2),Vector
     (0.238,-0.505,0.1477)));
    path->Add(Frame(Rotation::Quaternion(0,0,0.375,0.926),Vector
     (0.437,0.424,0.1477)));
    path->Add(Frame(Rotation::RotZ(M_PI_2*0),Vector
     (0.238,0.505,0.1477)));
```

pose_trajectory_publisher.cpp

```
    path->Add(Frame(Rotation::RPY(0,0,0),Vector(0.61,0,0.1477)));
    path->Finish();

    auto velocityProfile=new VelocityProfile_Trap(0.1,0.02);
    velocityProfile->SetProfile(0,path->PathLength());

    auto trajectorySegment=new Trajectory_Segment(path,
     velocityProfile);

    auto trajectoryStationary=new Trajectory_Stationary(1.0,Frame(
     Rotation::RPY(0,0,0),Vector(0.61,0,0.1477)));

    trajectory_.Add(trajectorySegment);
    trajectory_.Add(trajectoryStationary);
}
```

UFRGS
UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL

pose_trajectory_publisher.cpp

```
    catch(Error &error)
    {
      RCLCPP_ERROR_STREAM(get_logger(),"Error: " << error.
       Description() << std::endl);
      RCLCPP_ERROR_STREAM(get_logger(),"Type: " << error.
       GetType() << std::endl);
    }

    posePublisher_=create_publisher<geometry_msgs::msg::
       PoseStamped>("pose",10);

    t0_=now().seconds();
    using namespace std::chrono_literals;
    timer_=rclcpp::create_timer(this,this->get_clock(),100ms,std::bind
       (&PoseTrajectory::timerCB,this));
}
```

UFRGS
UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL

pose_trajectory_publisher.cpp

```
void PoseTrajectory::timerCB(void) const
{
  double t=fmin(now().seconds()−t0_,trajectory_.Duration());
  tf2::Stamped<KDL::Frame> pose(trajectory_.Pos(t),tf2::get_now(),"
      map");
  auto poseMsg=tf2::toMsg(pose);
  posePublisher_−>publish(poseMsg);
}

int main(int argc,char* argv[])
{
  rclcpp::init(argc,argv);
  rclcpp::spin(std::make_shared<PoseTrajectory>());
  rclcpp::shutdown();
  return 0;
}
```

Visualização como Pose

ros2 launch eng10026_trajectories display.launch.xml

Vídeo com a trajetória espacial

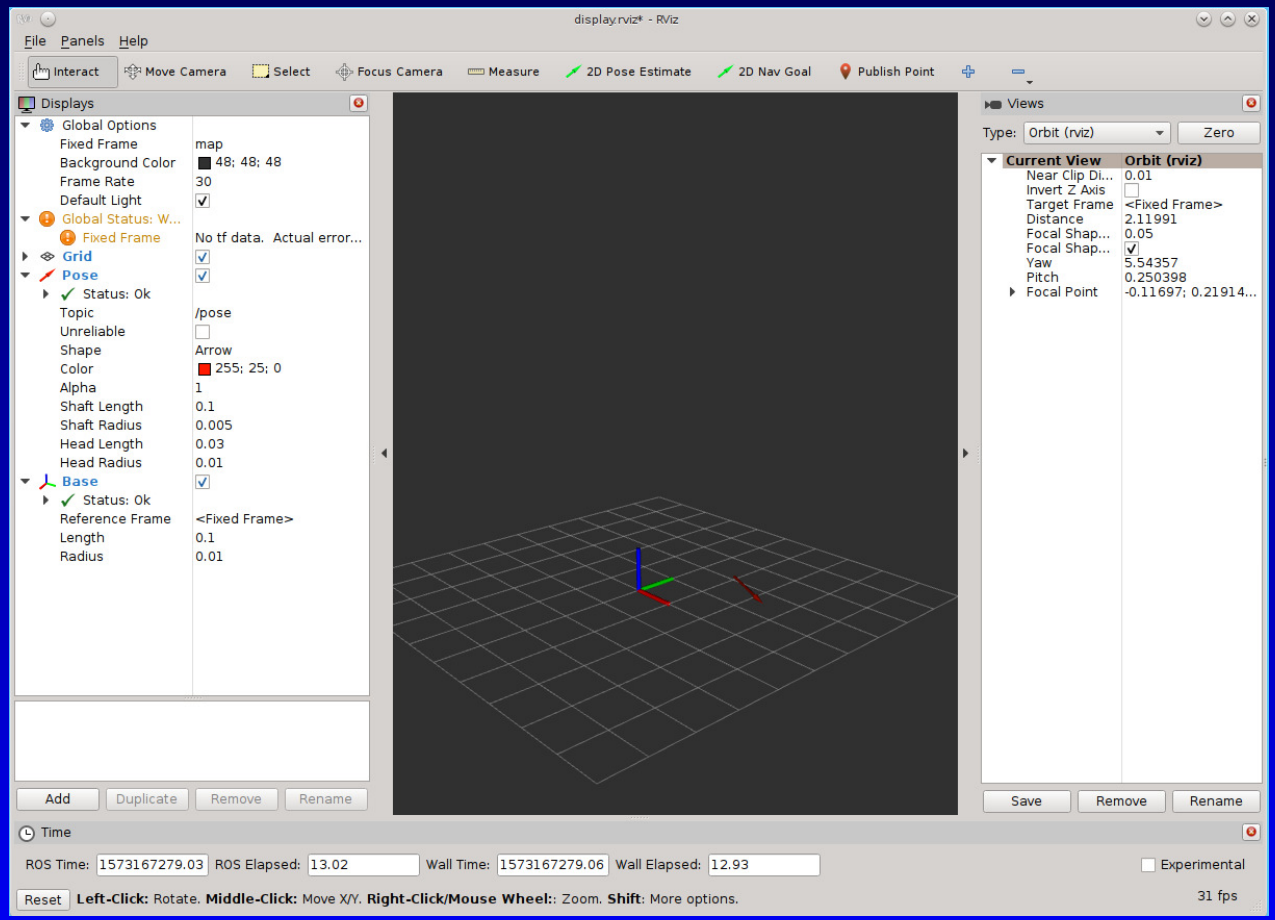
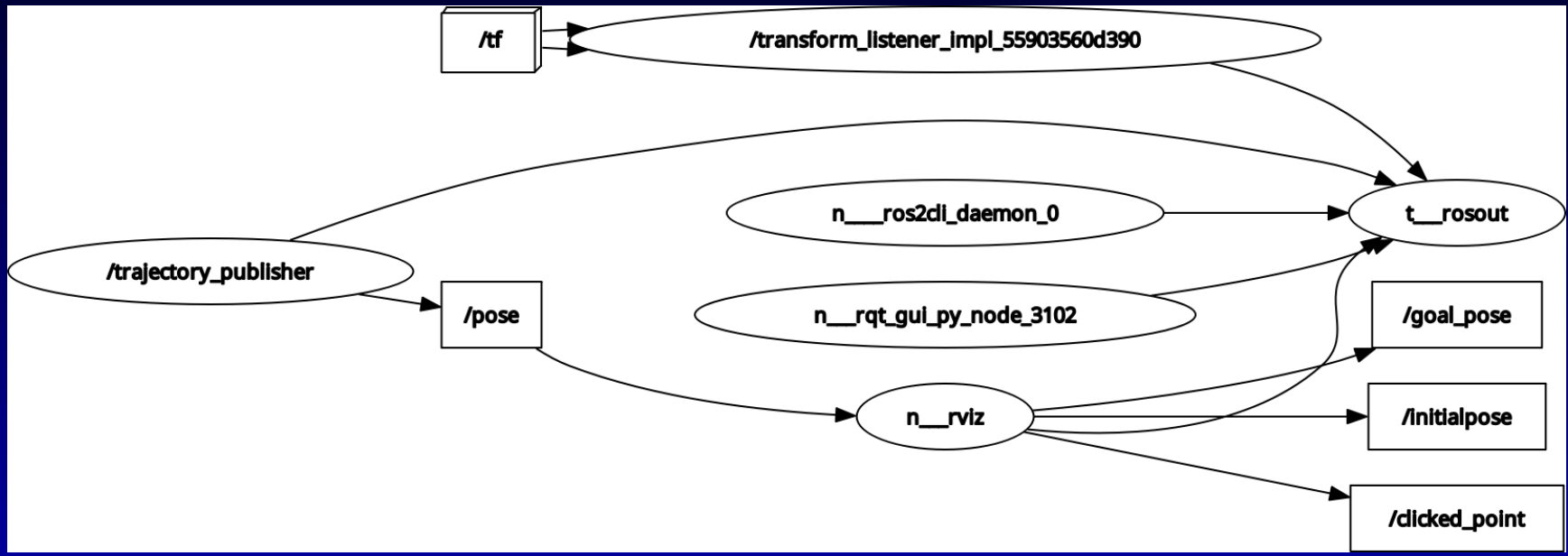
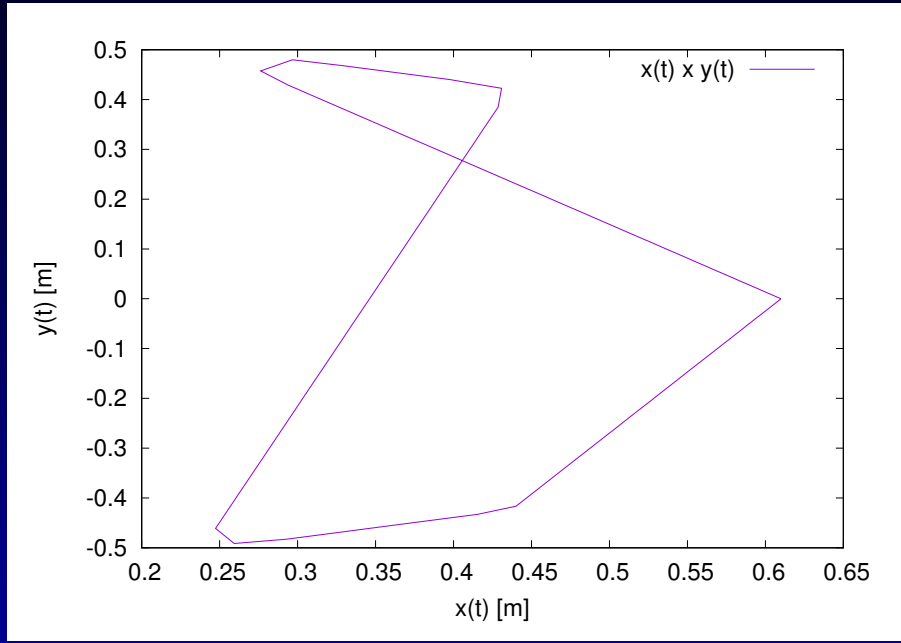


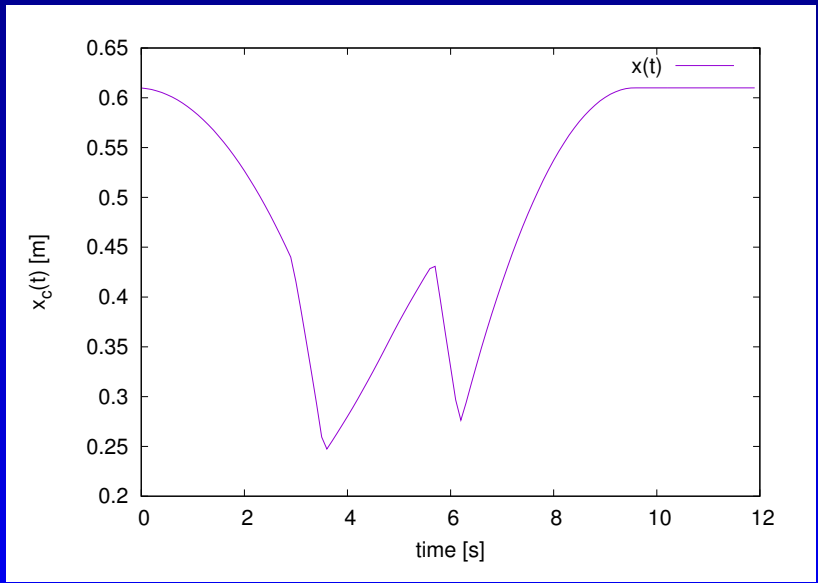
Gráfico de Computação

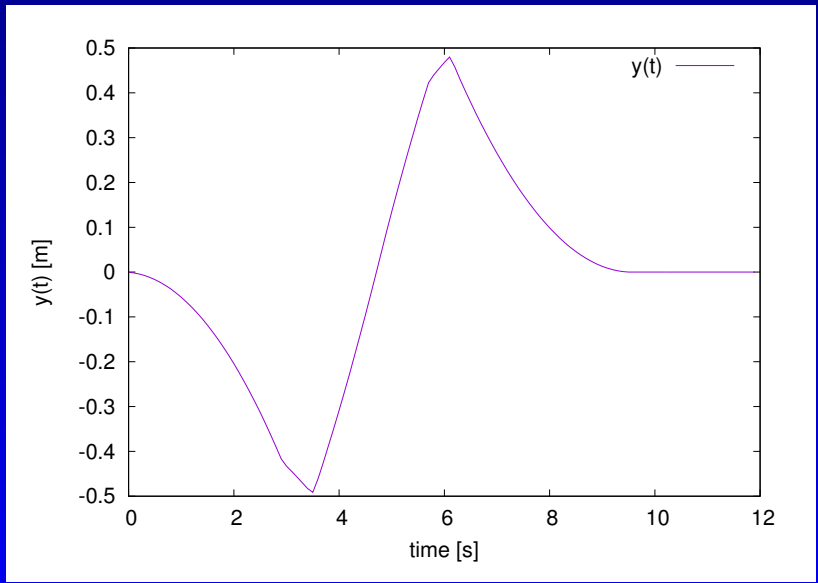


UFRGS
UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL

Trajetória







Visualização da Trajetória

- A visualização de `geometry_msgs/msg/PoseStamped` no rviz é através de um vetor
  - Não é muito adequada para vizualizar a orientação em 3D
    - Não permite representar rotação em torno do próprio eixo do vetor
  - É mais conveniente vizualizar um *frame*
  - É necessário publicar o *frame* no tópico `/tf`
  - É necessário transformar as mensagens `geometry_msgs/PoseStamped` em `tf2_msgs/msg/TFMessage`

tf2_msgs/msg/TFMessage

geometry_msgs/TransformStamped[] transforms

- std_msgs/Header header
  - builtin_interfaces/Time stamp
    - **int32** sec
    - **uint32** nanosec
  - **string** frame_id
- **string** child_frame_id
- Transform transform
  - Vector3 translation
    - **float64** x
    - **float64** y
    - **float64** z
  - Quaternion rotation
    - **float64** x 0
    - **float64** y 0
    - **float64** z 0
    - **float64** w 1

pose_stamped2tf.cpp

```
#include <rclcpp/rclcpp.hpp>

#include <geometry_msgs/msg/pose_stamped.hpp>
#include <tf2_msgs/msg/tf_message.hpp>

class PoseStampedToTf: public rclcpp::Node
{
  public:
  PoseStampedToTf(const char *node_name,const char *
      child_frame_id);
```

pose_stamped2tf.cpp

```
  private:
  rclcpp::Subscription<geometry_msgs::msg::PoseStamped>::
      SharedPtr poseSubscriber_;
  rclcpp::Publisher<tf2_msgs::msg::TFMessage>::SharedPtr
      tfPublisher_;
  std::string child_frame_id_;
  void poseStampedCB(const geometry_msgs::msg::PoseStamped::
      SharedPtr poseMsg) const;
};
```

pose_stamped2tf.cpp

```
PoseStampedToTf::PoseStampedToTf(const char *node_name,const
    char *child_frame_id): Node(node_name)
{
  child_frame_id_=child_frame_id;
  using std::placeholders::_1;
  poseSubscriber_=create_subscription<geometry_msgs::msg::
    PoseStamped>("pose",10,std::bind(&PoseStampedToTf::
    poseStampedCB,this,_1));
  tfPublisher_=create_publisher<tf2_msgs::msg::TFMessage>("/tf"
    ,10);
}
```

pose_stamped2tf.cpp

```
void PoseStampedToTf::poseStampedCB(const geometry_msgs::msg::
      PoseStamped::SharedPtr poseMsg) const
{
  tf2_msgs::msg::TFMessage tfMsg;

  tfMsg.transforms.resize(1);
  tfMsg.transforms[0].header.stamp=poseMsg->header.stamp;
  tfMsg.transforms[0].header.frame_id=poseMsg->header.frame_id;
  tfMsg.transforms[0].child_frame_id=child_frame_id_;
  tfMsg.transforms[0].transform.translation.x=poseMsg->pose.position.x;
  tfMsg.transforms[0].transform.translation.y=poseMsg->pose.position.y;
  tfMsg.transforms[0].transform.translation.z=poseMsg->pose.position.z;
  tfMsg.transforms[0].transform.rotation=poseMsg->pose.orientation;

      tfPublisher_->publish(tfMsg);
}
```

pose_stamped2tf.cpp

```
int main(int argc,char* argv[])
{
  rclcpp::init(argc,argv);

  if(argc < 2)
  {
    RCLCPP_ERROR_STREAM(rclcpp::get_logger("pose_stamped2tf"),"
      pose_stamped2tf: No child frame id.\n");
    return -1;
  }

  rclcpp::spin(std::make_shared<PoseStampedToTf>("pose_stamped2tf",argv
      [1]));

  return 0;
}
```

pose2tf.launch.xml

```
<launch>
  <arg name="child_id" default="trajectory"/>
  <node name="pose2tf" pkg="trajectory_conversions" exec="
     pose_stamped2tf" args="$(var child_id)"/>
</launch>
```

display.launch.xml

```
<launch>
  <node name="trajectory_publisher" pkg="eng10026_trajectories"
    exec="pose_trajectory_publisher"/>
  <include file="$(find-pkg-share eng10026_trajectories)/launch/
    pose2tf.launch.xml"/>
  <node name="rviz" pkg="rviz2" exec="rviz2" args="-d $(find-
    pkg-share eng10026_trajectories)/rviz/display.rviz"/>
</launch>
```

Instalação do Pacote

- Clonar e compilar o pacote `eng10026_trajectories`

```
cd ~/colcon_ws/src
git clone -b $ROS_DISTRO http://git.ece.ufrgs.br/eng10026/
    eng10026_trajectories
cd ..
colcon build --symlink-install
source $HOME/colcon_ws/install/setup.bash
```

Visualização como *Frame*

ros2 launch eng10026_trajectories display.launch.xml

Vídeo com a trajetória espacial

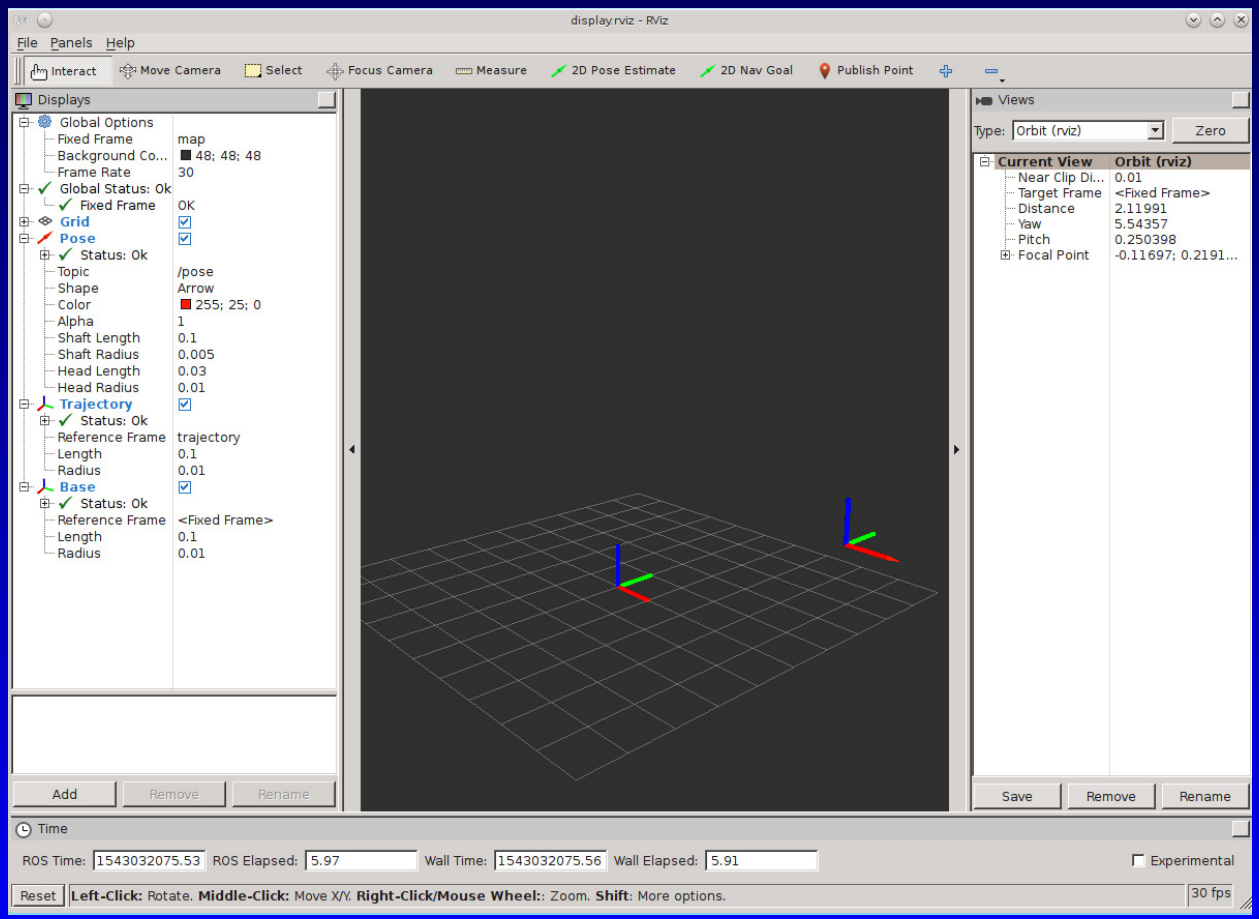
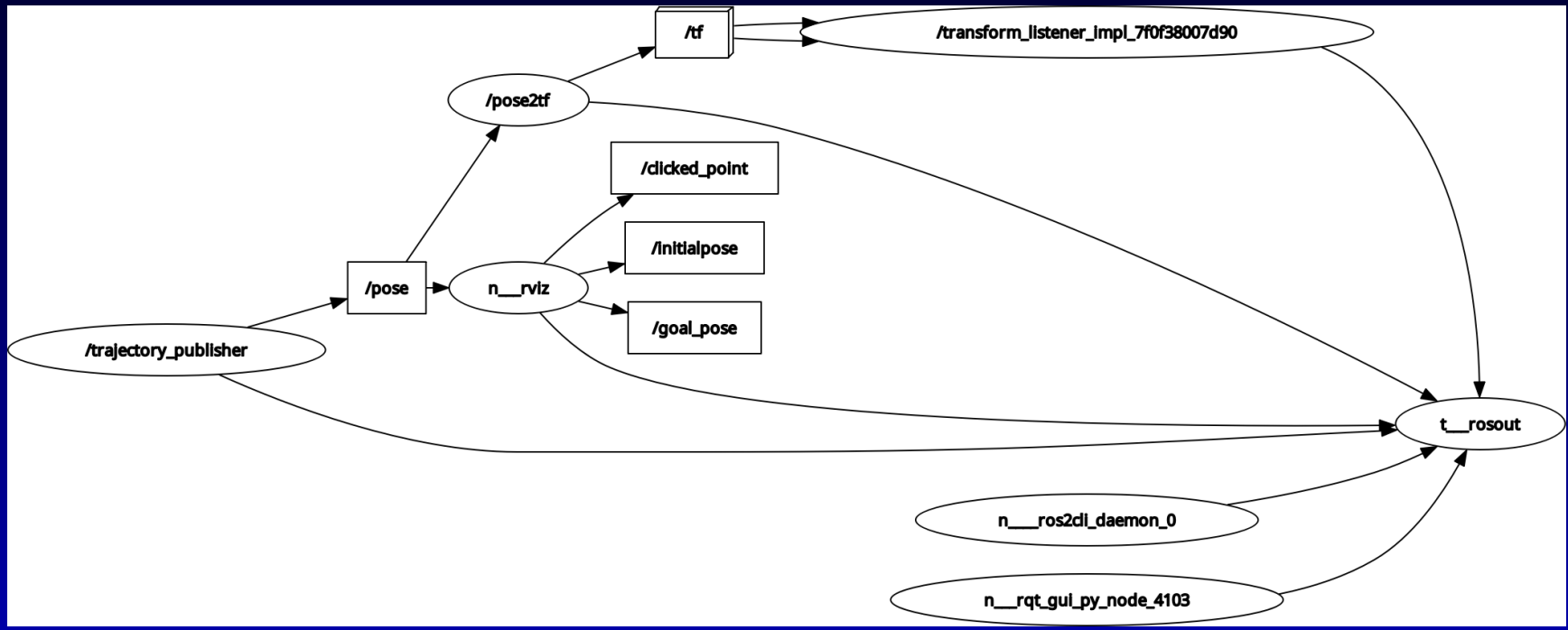


Gráfico de Computação



Mapeamento para o Espaço das Juntas

- O robô é acionado através das juntas
- As poses no espaço cartesiano devem ser mapeadas para o espaço das juntas pelo modelo cinemático inverso
- Posições das juntas publicadas através da mensagem `trajectory_msgs/msg/JointTrajectoryPoint`

---

float64[] positions

float64[] velocities

float64[] accelerations

float64[] effort

builtin_interfaces/Duration time_from_start

    int32 sec

    uint32 nanosec

Pacote `trajectory_conversions`

- Criar o pacote:

```
cd ~/colcon_ws/src
ros2 pkg create --build-type ament_cmake --dependencies
    rclcpp std_msgs geometry_msgs trajectory_msgs urdf
    tf2_kdl kdl_parser orocos_kdl --node-name
    pose_stamped2joint trajectory_conversions
```

- `package.xml` deve ser editado para configurar os detalhes de documentação e incluir dependências

UFRGS
UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL

CMakeLists.txt

- Editar `CMakeLists.txt` para descomentar e ajustar as *tags*:

```
add_executable(pose_stamped2joint
  src/pose_stamped2joint.cpp)

ament_target_dependencies(pose_stamped2joint
  rclcpp std_msgs geometry_msgs trajectory_msgs urdf tf2_kdl
      kdl_parser orocos_kdl)

install(TARGETS pose_stamped2joint
  DESTINATION lib/${PROJECT_NAME})
```

pose_stamped2joint.cpp

```
#include <rclcpp/rclcpp.hpp>
#include <geometry_msgs/msg/point_stamped.hpp>
#include <std_msgs/msg/string.hpp>
#include <trajectory_msgs/msg/joint_trajectory_point.hpp>
#include <kdl/chainiksolverpos_lma.hpp>
#include <tf2_kdl/tf2_kdl.h>
#include <kdl_parser/kdl_parser.hpp>

class Pose2Joint: public rclcpp::Node
{
  public:
  Pose2Joint(const std::string &name,const std::string &root,const std
      ::string &tip,const Eigen::Matrix<double,6,1> &L);
```

pose_stamped2joint.cpp

```
private:
rclcpp::Subscription<geometry_msgs::msg::PoseStamped>::SharedPtr
    poseSub_;
rclcpp::Publisher<trajectory_msgs::msg::JointTrajectoryPoint>::SharedPtr
    jointTrajPointPub_;

std::string robotDescription_;
KDL::Chain chain_;
std::unique_ptr<KDL::ChainIkSolverPos_LMA> ikSolverPos_;
KDL::JntArray q_;
    builtin_interfaces::msg::Time::UniquePtr t0_;

void poseCB(const geometry_msgs::msg::PoseStamped::SharedPtr pose);
void robotDescriptionCB(const std_msgs::msg::String::SharedPtr
    robotDescription);
};
```

pose_stamped2joint.cpp

```
Pose2Joint::Pose2Joint(const std::string &name,const std::string &root,const
      std::string &tip,const Eigen::Matrix<double,6,1> &L): Node(name), q_
      (0)
{
      using std::placeholders::_1;
      rclcpp::QoS qos(rclcpp::KeepLast(1));
      qos.transient_local();
      auto robotDescriptionSubscriber_=create_subscription<std_msgs::msg::
      String>("robot_description",qos,std::bind(&Pose2Joint::
      robotDescriptionCB,this,_1));
      while(robotDescription_.empty())
      {
            RCLCPP_WARN_STREAM_SKIPFIRST_THROTTLE(
      get_logger(),*get_clock(),1000,"Waiting for robot model on /
      robot_description.");
            rclcpp::spin_some(get_node_base_interface());
      }
```

pose_stamped2joint.cpp

```
KDL::Tree tree;
if(!kdl_parser::treeFromString(robotDescription_,tree))
    RCLCPP_ERROR_STREAM(get_logger(),"Failed to construct
    KDL tree.");
if(!tree.getChain(root,tip,chain_)) RCLCPP_ERROR_STREAM(
    get_logger(),"Failed to get chain from KDL tree.");
ikSolverPos_=std::make_unique<KDL::ChainIkSolverPos_LMA>(
    chain_,L);

q_.resize(chain_.getNrOfJoints());

jointTrajPointPub_=create_publisher<trajectory_msgs::msg::
    JointTrajectoryPoint>("joint_trajectory_point",10);
poseSub_=create_subscription<geometry_msgs::msg::PoseStamped
    >("/pose",10,std::bind(&Pose2Joint::poseCB,this,_1));
}
```

UFRGS
UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL

pose_stamped2joint.cpp

```
void Pose2Joint::poseCB(const geometry_msgs::msg::PoseStamped::
      SharedPtr poseStamped)
{
  tf2::Stamped<KDL::Frame> goalFrame;
  tf2::fromMsg(*poseStamped,goalFrame);

  int error=ikSolverPos_->CartToJnt(q_,goalFrame,q_);
  if(error != 0) RCLCPP_ERROR_STREAM(get_logger(),"Failed to
     compute invere kinematics: (" << error << ") "
        << ikSolverPos_->strError(error));

  trajectory_msgs::msg::JointTrajectoryPoint jointTrajPoint;
  jointTrajPoint.positions.resize(q_.rows());
  Eigen::VectorXd::Map(&jointTrajPoint.positions[0],jointTrajPoint.
     positions.size())=q_.data;
```

pose_stamped2joint.cpp

```
if(!t0_) t0_=std::make_unique<builtin_interfaces::msg::Time>(
    poseStamped->header.stamp);
if(poseStamped->header.stamp.nanosec >= t0_->nanosec)
{
  jointTrajPoint.time_from_start.nanosec=poseStamped->header.
    stamp.nanosec-t0_->nanosec;
  jointTrajPoint.time_from_start.sec=poseStamped->header.stamp.
    sec-t0_->sec;
}
else
{
  jointTrajPoint.time_from_start.nanosec=1000000000+
    poseStamped->header.stamp.nanosec-t0_->nanosec;
  jointTrajPoint.time_from_start.sec=poseStamped->header.stamp.
    sec-t0_->sec-1;
}
```

pose_stamped2joint.cpp

```
  jointTrajPointPub_->publish(jointTrajPoint);
}

void Pose2Joint::robotDescriptionCB(const std_msgs::msg::String::
      SharedPtr robotDescription)
{
      robotDescription_=robotDescription->data;
}
```

pose_stamped2joint.cpp

```
int main(int argc,char* argv[])
{
    rclcpp::init(argc,argv);
    if(argc < 3)
    {
        RCLCPP_ERROR_STREAM(rclcpp::get_logger("
    pose_stamped2joint"),"Please, provide a chain root and a chain tip");
        return -1;
    }

    Eigen::Matrix<double,6,1> L;
    L << 1.0 , 1.0 , 1.0, 0.01, 0.01, 0.01;
    for(int i=0;i < argc-3 && i < L.size();i++) L(i)=atof(argv[i+3]);

    rclcpp::spin(std::make_shared<Pose2Joint>("pose_stamped2joint",argv
    [1],argv[2],L));

    return 0;
```

trajectory.launch.xml

```
<launch>
  <include file="$(find-pkg-share eng10026_trajectories)/launch/
      display.launch.xml"/>
  <include file="$(find-pkg-share q2d_description)/launch/q2d.
      launch.xml"/>
  <node name="ik" pkg="trajectory_conversions" exec="
      pose_stamped2joint" args="origin_link tool_link 1 1 0 0 0 0"/>
</launch>
```

Instalação do Pacote

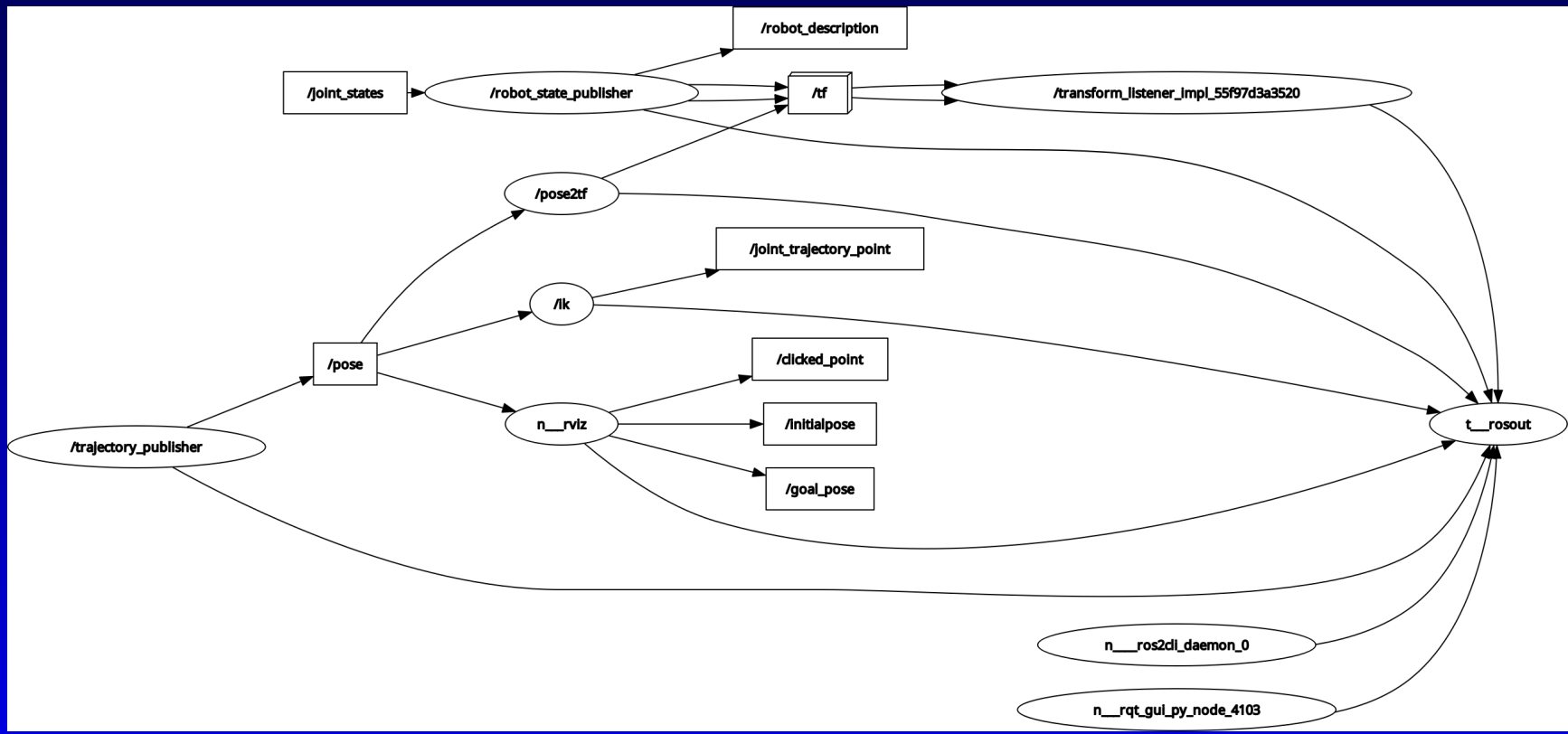
- Clonar e compilar o pacote `trajectory_conversions`

```
cd ~/colcon_ws/src
git clone -b $ROS_DISTRO http://git.ece.ufrgs.br/
      trajectory_conversions
cd ..
colcon build --symlink-install
source $HOME/colcon_ws/install/setup.bash
```

Execução

ros2 launch eng10026_trajectories trajectory.launch.xml

ros2 topic echo /joint_trajectory_point



Referência para o(s) Controlador(es)

- Pode ser necessário extrair do tópico `/joint_trajectory_point` as referências dos controladores
  - Se for utilizado controlador por torque calculado ou pid+gravidade, a referência já é do tipo `trajectory_msgs/JointTrajectoryPoint`
    - Não é necessário extrair a referência para cada junta
  - Se forem utilizados controladores independentes por junta, cada controlador tem uma referência do tipo `std_msg/msg/Float64`

---

float64 data

---

gazebo.launch.xml

```
<launch>
  <arg name="joint" default="false"/>
  <arg name="controller" default="pid_plus_gravity"/>
  <arg name="pause" default="true"/>
  <arg name="gui" default="true"/>
  <arg name="use_sim_time" default="true"/>

  <include file="$(find-pkg-share q2d_bringup)/launch/gazebo.launch.xml
    ">
    <arg name="controller" value="$(var controller)"/>
    <arg name="pause" value="$(var pause)"/>
    <arg name="gui" value="$(var gui)"/>
    <arg name="use_sim_time" value="$(var use_sim_time)"/>
  </include>
```

gazebo.launch.xml

```
<group if="$(eval '\'$(var controller)\' == \'pid\'')">
  <node name="shoulder_demux" pkg="topic_tools" exec="transform"
   args="/joint_trajectory_point /shoulder_controller/command std_msgs/
   Float64 'm.positions[0]'"/>
  <node name="elbow_demux" pkg="topic_tools" exec="transform" args
   ="/joint_trajectory_point /elbow_controller/command std_msgs/Float64
   'm.positions[1]'"/>
</group>

<node pkg="tf2_ros" exec="static_transform_publisher" name="
   q2d_origin_publisher" args="0 0 0 0 0 0 1 map origin_link"/>
```

gazebo.launch.xml

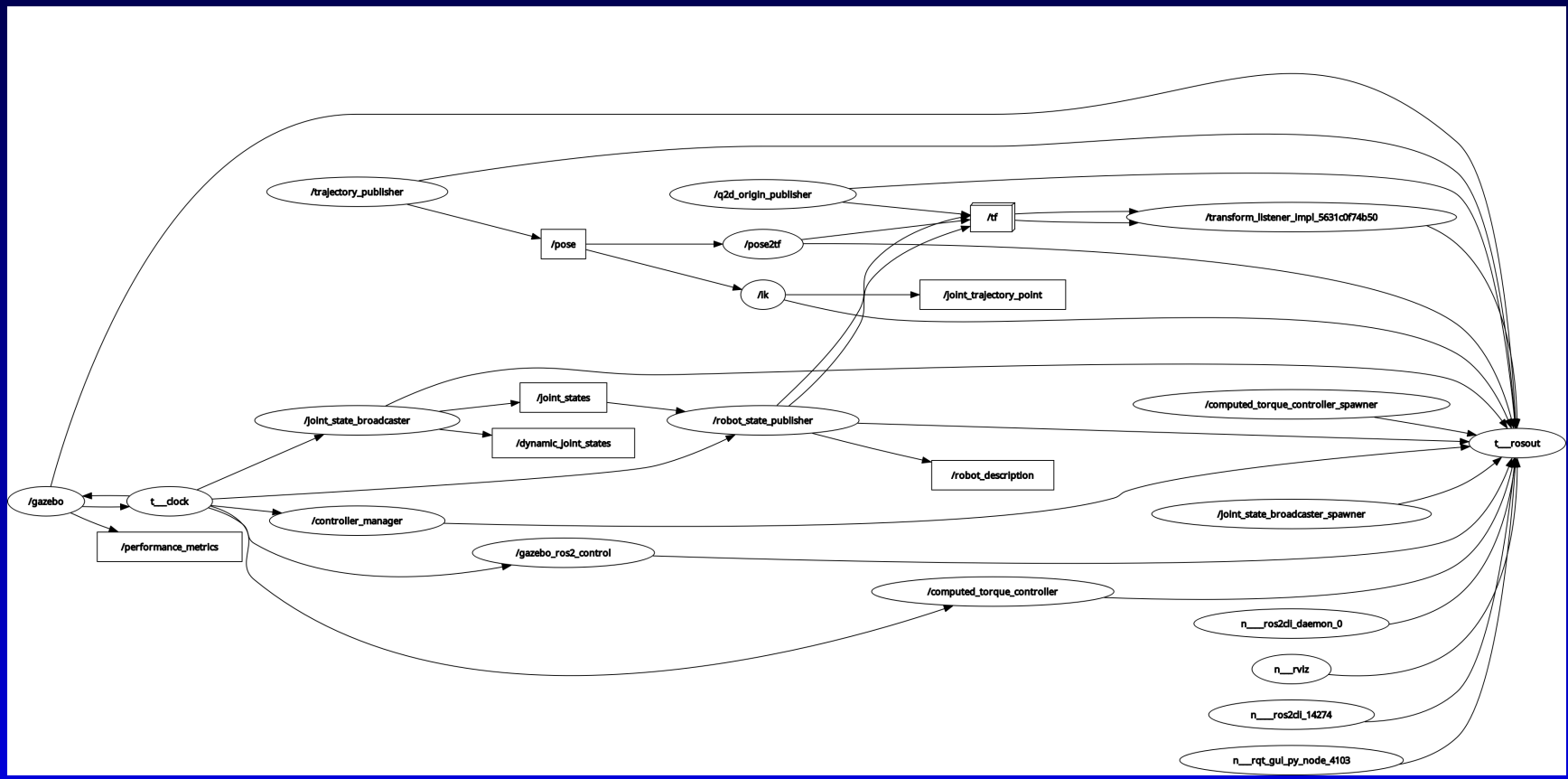
```
<group unless="$(var joint)">
  <node name="trajectory_publisher" pkg="eng10026_trajectories" exec=
   "pose_trajectory_publisher">
    <param name="use_sim_time" value="$(var use_sim_time)"/>
  </node>
  <node name="ik" pkg="trajectory_conversions" exec="
   pose_stamped2joint" args="origin_link tool_link 1 1 0 0 0 0">
    <remap from="joint_trajectory_point" to="command"/>
    <param name="use_sim_time" value="$(var use_sim_time)"/>
  </node>
  <include file="$(find-pkg-share eng10026_trajectories)/launch/pose2tf.
   launch.xml"/>
  <node name="rviz" pkg="rviz2" exec="rviz2" args="-d $(find-pkg-
   share eng10026_trajectories)/rviz/gazebo.rviz"/>
</group>
```

gazebo.launch.xml

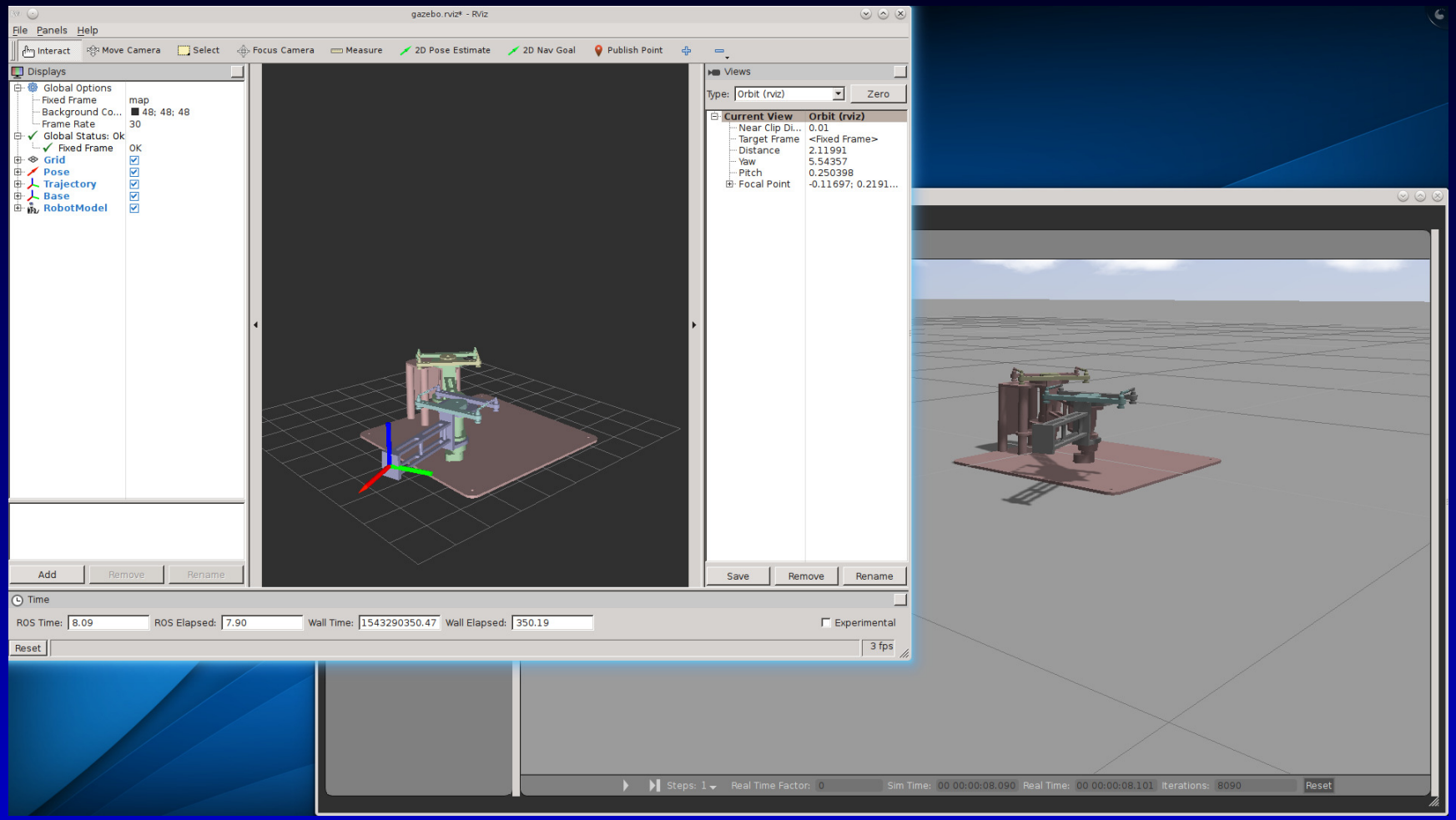
```
  <group if="$(var joint)">
    <node name="trajectory_publisher" pkg="eng10026_trajectories" exec=
     "joint_trajectory_publisher">
      <remap from="joint_trajectory_point" to="command"/>
      <param name="use_sim_time" value="$(var use_sim_time)"/>
    </node>
    <node name="rviz" pkg="rviz2" exec="rviz2" args="-d $(find-pkg-
     share eng10026_trajectories)/rviz/gazebo_joint.rviz"/>
  </group>
</launch>
```

Execução

ros2 launch eng10026_trajectories gazebo.launch.xml



Execução



- Flange não segue perfeitamente a trajetória
  - Pontos em que a trajetória não é factível
  - Desempenho do controlador

Trajetórias no Espaço das Juntas

- Usualmente o caminho é uma reta
- Pode-se usar diretamente a classe `KDL::VelocityProfile` e derivadas para gerar a trajetória de cada junta
- Movimento de $(0, 0)$ a $(\frac{\pi}{4}, -\frac{\pi}{4})$
- Tempo final=5 s
- Publica o tópico `trajectory_msgs/msg/JointTrajectoryPoint`

CMakeLists.txt

- Incluído no pacote `eng10026_trajectories`
- Editar `CMakeLists.txt` para adicionar as *tags* `add_executable` e `target_link_libraries:`

```
add_executable(joint_trajectory_publisher
  src/joint_trajectory_publisher.cpp)

ament_target_dependencies(joint_trajectory_publisher
  rclcpp tf2_kdl geometry_msgs trajectory_msgs orocos_kdl)

install(TARGETS joint_trajectory_publisher
  DESTINATION lib/${PROJECT_NAME})
```

UFRGS
UNIVERSIDADE FEDERAL
DO RIO GRANDE DO SUL

joint_trajectory_publisher.cpp

```
#include <rclcpp/rclcpp.hpp>
#include <kdl/velocityprofile_spline.hpp>
#include <trajectory_msgs/msg/joint_trajectory_point.hpp>

using namespace KDL;

class JointTrajectory: public rclcpp::Node
{
  public:
  JointTrajectory(std::vector<double> &p0,std::vector<double> &pf,
      double tf,const char *name="trajectory_publisher");
```

UFRGS
UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL

joint_trajectory_publisher.cpp

```
  private:
  std::vector<VelocityProfile_Spline> velocityProfile_;
  double t0_;
  rclcpp::TimerBase::SharedPtr timer_;
  rclcpp::Publisher<trajectory_msgs::msg::JointTrajectoryPoint>::
      SharedPtr jointPublisher_;

  void timerCB(void) const;
};
```

joint_trajectory_publisher.cpp

```
JointTrajectory::JointTrajectory(std::vector<double> &p0,std::vector<
      double> &pf,double tf,const char *name): Node(name)
{
  velocityProfile_.resize(min(p0.size(),pf.size()));
  for(unsigned int i=0;i < velocityProfile_.size();i++)
    velocityProfile_[i].SetProfileDuration(p0[i],0,0,pf[i],0,0,tf);

  jointPublisher_=create_publisher<trajectory_msgs::msg::
      JointTrajectoryPoint>("joint_trajectory_point",10);

  t0_=now().seconds();
  using namespace std::chrono_literals;
  timer_=rclcpp::create_timer(this,this->get_clock(),100ms,std::bind
      (&JointTrajectory::timerCB,this));
}
```

joint_trajectory_publisher.cpp

```
void JointTrajectory::timerCB(void) const
{
  double t=fmin(now().seconds()−t0_,velocityProfile_[0].Duration());

  trajectory_msgs::msg::JointTrajectoryPoint jointMsg;
  for(auto const &velocityProfile : velocityProfile_)
  {
    jointMsg.positions.push_back(velocityProfile.Pos(t));
    jointMsg.velocities.push_back(velocityProfile.Vel(t));
    jointMsg.accelerations.push_back(velocityProfile.Acc(t));
  }

  double sec;
  jointMsg.time_from_start.nanosec=modf(t,&sec)∗1e9;
  jointMsg.time_from_start.sec=sec;
```

joint_trajectory_publisher.cpp

```
  jointPublisher_->publish(jointMsg);
}

int main(int argc,char* argv[])
{
  rclcpp::init(argc,argv);

  std::vector<double> p0 {0, 0};
  std::vector<double> pf {M_PI_4,-M_PI_4};
  rclcpp::spin(std::make_shared<JointTrajectory>(p0,pf,5.0));

  rclcpp::shutdown();
  return 0;
}
```

Execução

ros2 launch eng10026_trajectories gazebo.launch joint:=**true**