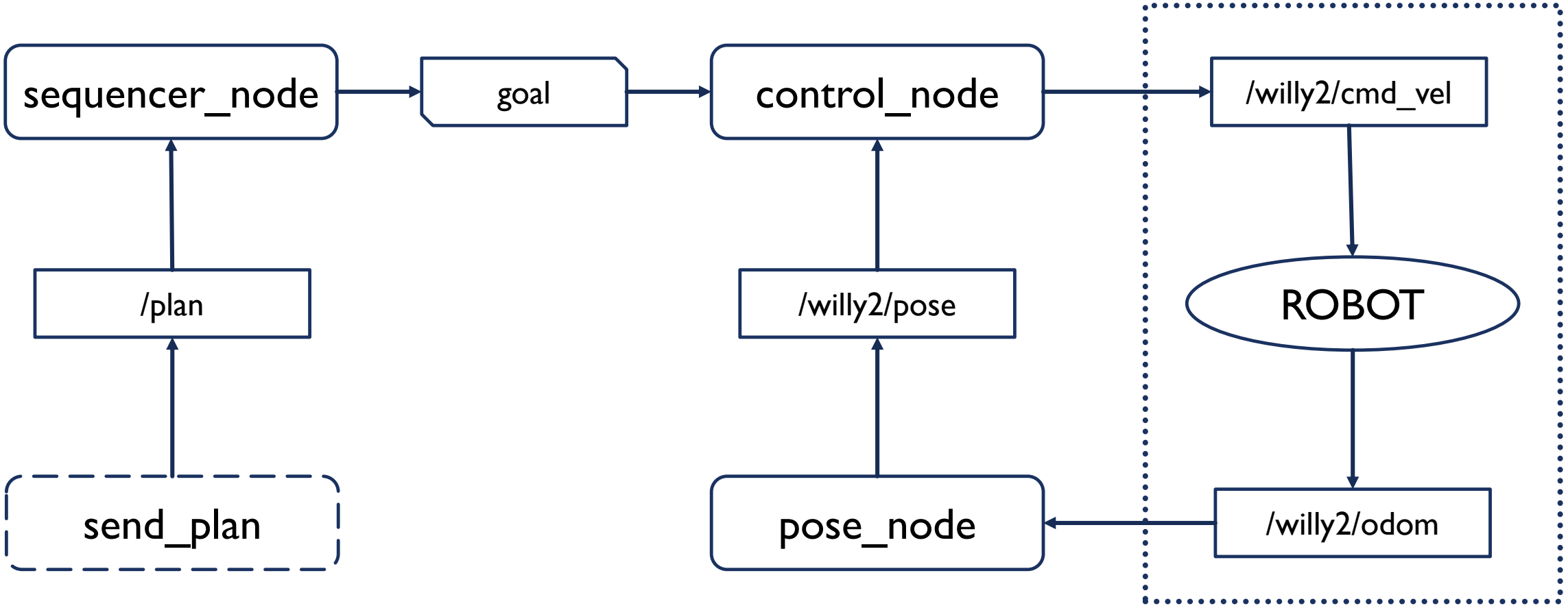
ROS ARCHITECTURE: AN EXAMPLE

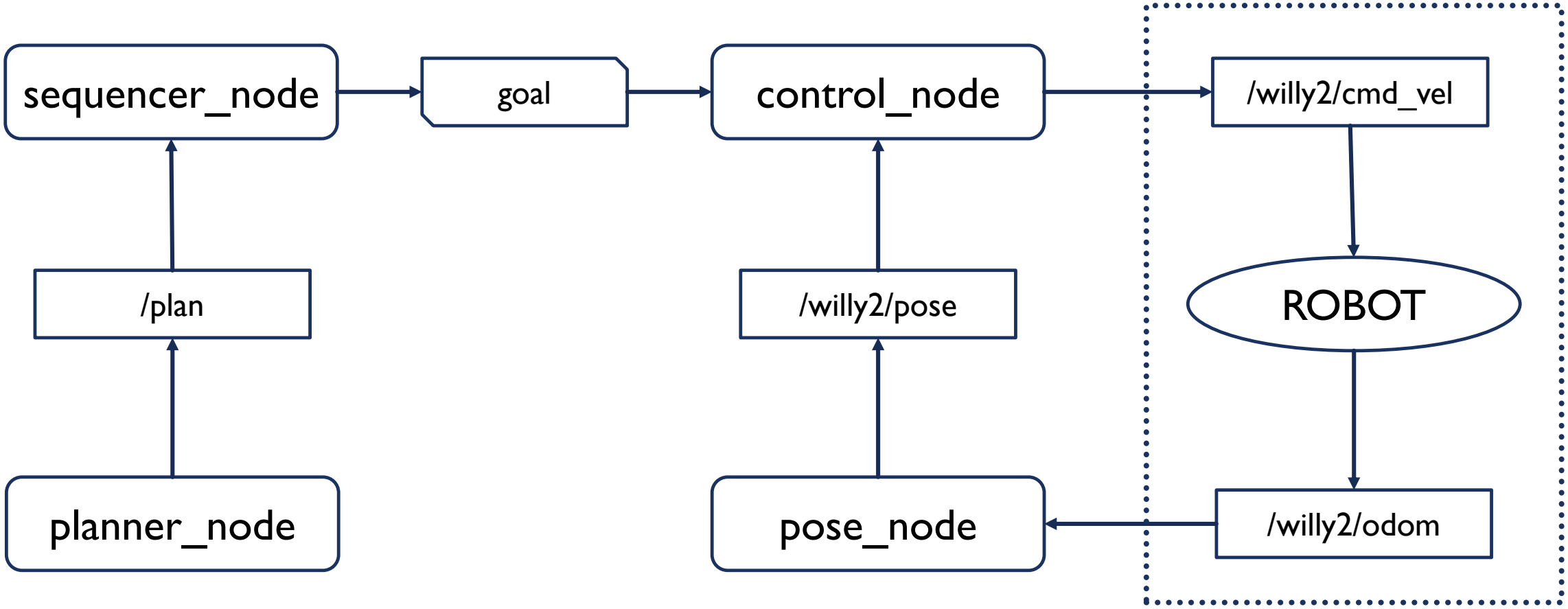
ROBOTICS

POLITECNICO
MILANO 1863

GENERAL ARCHITECTURE



GENERAL ARCHITECTURE



SEQUENCER_NODE

Input: a plan (as a sequence of poses) provided by some higher level system

Output: a local goal (as a single pose) provided by request using a service

Logic: Receive a complete plan and provides sub-goal in sequence

POSE_NODE

Input: odometry (as linear and angular velocity) provided by the robot

Output: pose of the robot (position and orientation) in the global reference frame

Assumptions: The robot can only move forward or rotate, not both at the same time

Logic: At each execution loop of the node, integrate the odometry to calculate the new position of the robot

POSE_NODE

```
//Attributes
ros::NodeHandler Handle;
double x, y, yaw;

//In the Prepare() method
if (!Handle.getParam(ros::this_node::getName()+"/x", x)) return false;
if (!Handle.getParam(ros::this_node::getName()+"/y", y)) return false;
if (!Handle.getParam(ros::this_node::getName()+"/yaw", yaw)) return false;
```

POSE_NODE

```
if(t < 0) { t = msg->header.stamp.toSec(); return; }
float v = msg->twist.twist.linear.x;
float w = msg->twist.twist.angular.z;
double dt = msg->header.stamp.toSec() - t;
x = x + v*cos(yaw)*dt;
y = y + v*sin(yaw)*dt;
yaw = yaw + w*dt;
t = msg->header.stamp.toSec();
```

POSE_NODE

```
geometry_msgs::PoseStamped out;
out.header = msg->header;
out.header.frame_id = "/base_link";
//quaternion magic out.pose.orientation <- yawToQuaternion(yaw);
out.pose.position.x = x;
out.pose.position.y = y;
out.pose.position.z = 0.0;
posePub.publish(out);
```

CONTROL_NODE

Input: Local goal (requested via service) and current robot pose (position and orientation)

Output: velocity command (as linear or angular velocity) in the robot reference frame

Constraint: The robot can only move forward or rotate, not both at the same time

Logic: Request a new local goal, align the robot with the goal and reach the goal

CONTROL_NODE

```
if(!gotPose) {
    diffdrive::GetGoal srv;
    srv.request.go = true;
    if(goalCl.call(srv)) {
        goal = srv.response.goal;
        gotPose = true; position = false; orientation = false;
    } else { return; }
}
```

CONTROL_NODE

```
if(!orientation) {
    double gy = goal.position.y; double gx = goal.position.x;
    double th = atan2(gy - msg->pose.position.y, gx - msg->pose.position.x);
    //quaternion magic double yaw <- yawFromQuaternion(msg->pose.orientation)
    if(fabs(yaw - th) < 0.005)
        orientation = true;
    else
        out.angular.z = 0.03;
}
```

CONTROL_NODE

```
if(!position && orientation) {
    double gy = goal.position.y;       double gx = goal.position.x;
    double py = msg->pose.position.y; double px = msg->pose.position.x;
    double d2 = pow(xx - px, 2) + pow(yy - px, 2);
    //control magic out.linear.x <- DistanceToSpeed(d2);
    if(d2 < 0.2*0.2) { out.linear.x = 0.0; position = true; }
}
if(position && orientation)
    gotPose = false;
cmdPub.publish(out);
```

ROS LAUNCH FILE

```
<launch>
    <node pkg="diffdrive" type="control_node" name="control_node" />
    <node pkg="diffdrive" type="pose_node" name="pose_node">
        <param name="x" value="0.0"/>
        <param name="y" value="0.0"/>
        <param name="yaw" value="0.0"/>
    </node>
    <node pkg="diffdrive" type="sequencer_node" name="sequencer_node" />
</launch>
```

EXTRA JUICE

Nodes started using the launch file have no low level output, add this:

```
<node pkg="pack" type="mNode" name="mNode" output="screen" />
```

Launch files have a hierarchical structure, you can include other launch files:

```
<launch>
    <include file="$(find pack)/launch/bunch_of_nodes.launch" />
    <include file="$(find pack)/launch/heap_of_nodes.launch" />
</launch>
```

It is possible to include parameters using a file:

```
<rosparam file="$(find pack)/config/param.yaml" command="load"/>
```

RUNNING EVERYTHING

1. Start the ROS framework

   ```
   roscore
   ```

2. Run gazebo as a ROS node

   ```
   rosrun gazebo_ros gazebo […]/test_world.sdf
   ```

3. Add the model in gazebo

4. Start the ROS nodes with the launch file

   ```
   roslaunch diffdrive diffdrive.launch
   ```

5. Run send_plan node to start the execution

   ```
   rosrun diffdrive send_plan
   ```

RUNNING EVERYTHING

1. Start the ROS framework

   ```
   roscore
   ```

2. Run gazebo as a ROS node

   ```
   rosrun gazebo_ros gazebo […]/test_world.sdf
   ```

3. Add the model in gazebo
4. Start the ROS nodes with the launch file

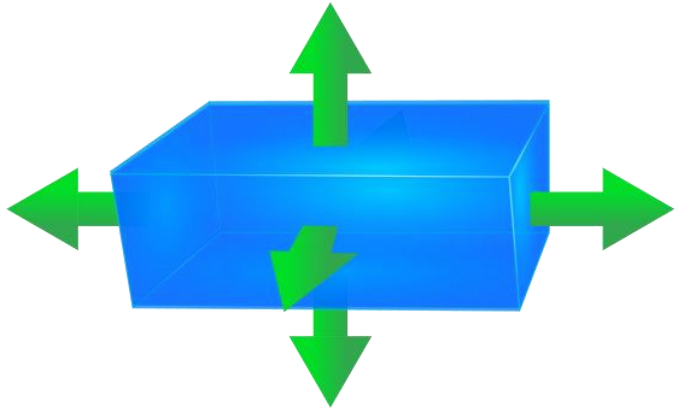
   ```
   roslaunch diffdrive diffdrive.launch
   ```

5. Run send_plan node to start the execution

   ```
   rosrun diffdrive send_plan
   ```

Use apt-get and install:
`ros-jade-gazebo7-ros-pkgs`

EULER ANGLES AND QUATERNIONS

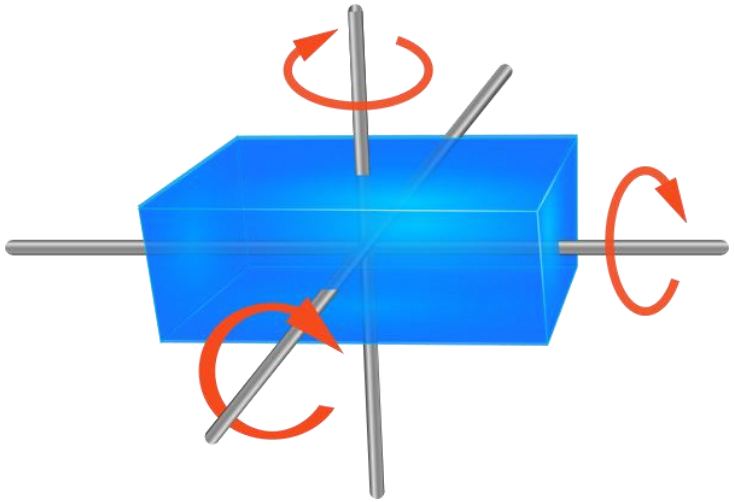


6 Degrees of freedom: 3 coordinates for the position, 3 coordinates for the orientation

Position defined by x, y and z

How it is possible to define the orientation of an object with 6 DoF?

- Euler angles
- Trait-Bryan angles
- Quaternions
- Rotation matrices



EULER/TRAIT-BRYAN ANGLES

Orientation defined as a sequence of rotation around three axes

Euler: first and last axis are the same (z-x-z, x-y-x, y-z-y,…)

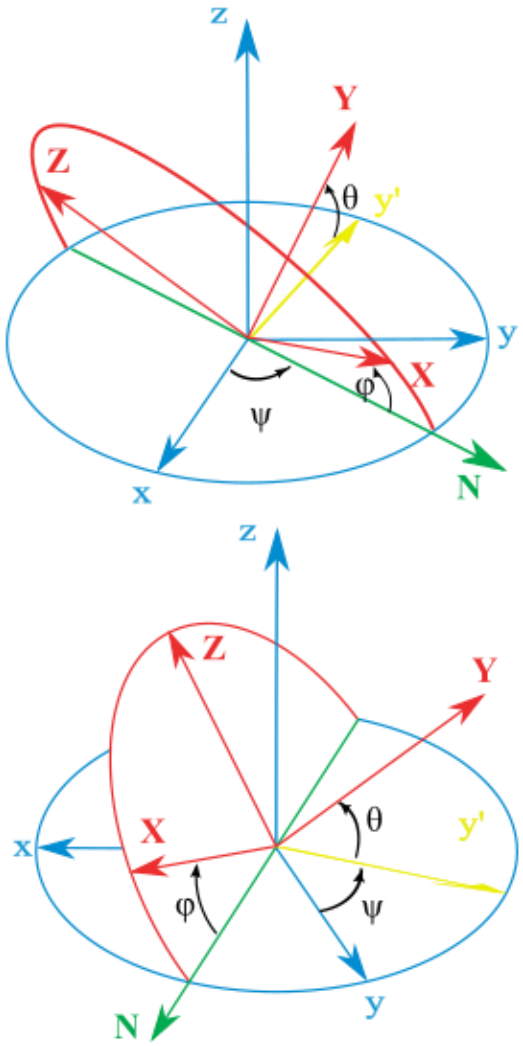
Trait-Bryan: three different axes (x-y-z, z-y-x, y-z-x,…)

Trait-Bryan (z-y-x) are the most used and are known as:

Yaw, pitch and roll

Heading, elevation and bank

If you are working on a straight surface with a ground vehicle, you are mostly interested in the rotation around the z axis



EULER/TRAIT-BRYAN ANGLES

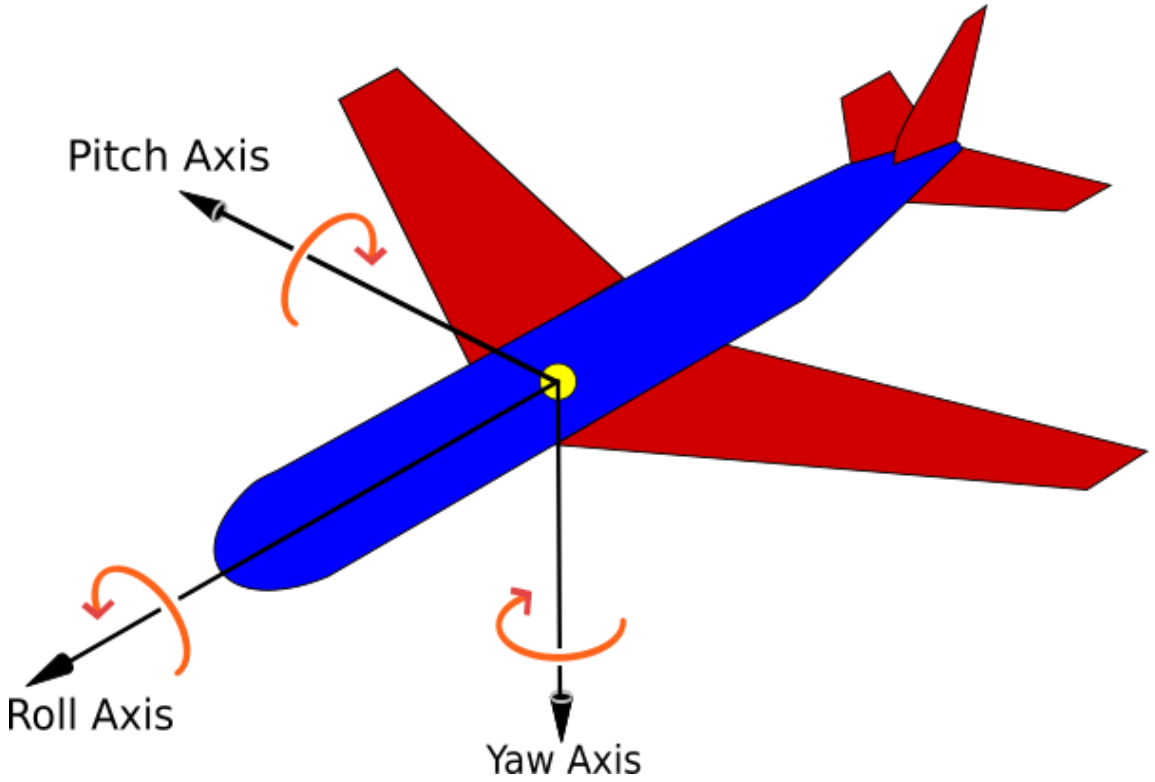
Orientation defined as a sequence of rotation around three axes

- **Euler**: first and last axis are the same (z-x-z, x-y-x, y-z-y,…)
- **Trait-Bryan**: three different axes (x-y-z, z-y-x, y-z-x,…)

Trait-Bryan (z-y-x) are the most used and are known as:

- Yaw, pitch and roll
- Heading, elevation and bank

If you are working on a straight surface with a ground vehicle, you are mostly interested in the rotation around the z axis



QUATERNIONS

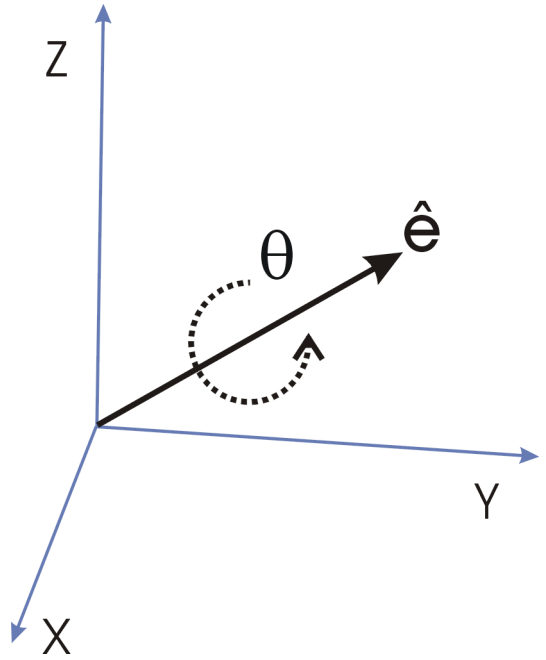
Four values used to define a vector in the 3D (x, y, z) space and a rotation around that axis (w).

Can be used to describe an orientation (rotation w.r.t. a fixed point) or angular movements.

Rotation around the z axis: <0; 0; 0,3826; 0,9238>

A single quaternion is enough to define the orientation of an object in a 6 DoF system

Quaternion algebra can be used to combine rotate object in space



CONVERSIONS

From RPY to quaternion

$$\begin{matrix}\phi \\ \vartheta \\ \psi\end{matrix} = \begin{matrix}roll/2 \\ pitch/2 \\ yaw/2\end{matrix}$$

$$\begin{matrix}x \\ y \\ z \\ w\end{matrix} = \begin{matrix}\sin\phi\cos\vartheta\cos\psi - \cos\phi\sin\vartheta\sin\psi \\ \cos\phi\sin\vartheta\cos\psi + \sin\phi\cos\vartheta\sin\psi \\ \cos\phi\cos\vartheta\sin\psi - \sin\phi\sin\vartheta\cos\psi \\ \cos\phi\cos\vartheta\cos\psi + \sin\phi\sin\vartheta\sin\psi\end{matrix}$$

From quaternion to RPY

$$\begin{matrix}roll \\ pitch \\ yaw\end{matrix} = \begin{matrix}\text{atan}_2(2(wx + yz), 1 - 2(x^2 + y^2) \\ \text{asin}(2(wy - zx)) \\ \text{atan}_2(2(wz + xy), 1 - 2(y^2 + z^2)\end{matrix}$$

ROS AND GAZEBO

ROBOTICS

POLITECNICO
MILANO 1863

HEADERS

```
#include "ros/ros.h"
// All possible messages/services
#include "std_msgs/Float32.h"
// Custom callback queue
#include "ros/callback_queue.h"
#include "ros/subscribe_options.h"
```

MEMBERS

```
// A node use for ROS transport
private: std::unique_ptr<ros::NodeHandle> rosNode;

// A ROS subscriber and publisher
private: ros::Subscriber rosSub;
private: ros::Publisher rosPub;
```

MEMBERS

```
// A ROS callbackqueue that helps process messages
private: ros::CallbackQueue rosQueue;
// A thread the keeps running the rosQueue
private: std::thread rosQueueThread;

// Gazebo connection to manage the publisher
private: event::ConnectionPtr updateConnection;
```

LOAD

```
// Initialize ros, if it has not already bee initialized
if (!ros::isInitialized())
   ros::init(0, NULL, "gazebo_client",
         ros::init_options::NoSigintHandler);
// Create the ROS node
rosNode.reset(new ros::NodeHandle("gazebo_client"));
```

LOAD

```
// Create a named topic, and subscribe to it
ros::SubscribeOptions so =
      ros::SubscribeOptions::create<std_msgs::Float32>("/sub", 1,
      boost::bind(&mPlugin::OnRosMsg, this, _1), ros::VoidPtr(),
      &rosQueue);
rosSub = rosNode->subscribe(so);

// Create a named topic, advertise it
rosPub = rosNode->advertise<std_msgs::Float32>("/pub", 1);
```

LOAD

```
// Spin up the queue helper thread
rosQueueThread =
      std::thread(std::bind(&mPlugin::QueueThread, this));

// Listen on world updates
updateConnection = event::Events::ConnectWorldUpdateBegin(
      boost::bind(&mPlugin::OnUpdate, this, _1));
```

METHODS

```
// Handle an incoming message from ROS
public: void OnRosMsg(const std_msgs::Float32ConstPtr &msg) {
   this->SetVelocity(msg->data);
}

// Publish message at each world update
private: void OnUpdate(sensors::mSensorPtr sensor) {
   this->GetValue(sensor)
}
```

METHODS

```
// ROS helper function that processes messages
private: void QueueThread() {
   static const double timeout = 0.1;
   while (rosNode->ok()) {
      rosQueue.callAvailable(ros::WallDuration(timeout));
   }
}
```

CMAKELISTS

```
cmake_minimum_required(VERSION 2.8 FATAL_ERROR)

#add ros references
find_package(roscpp REQUIRED)
find_package(std_msgs REQUIRED)
include_directories(${roscpp_INCLUDE_DIRS})
include_directories(${std_msgs_INCLUDE_DIRS})

#include roscpp libraries for the linker
target_link_libraries(m_plugin ${GAZEBO_libraries} ${roscpp_LIBRARIES})
```