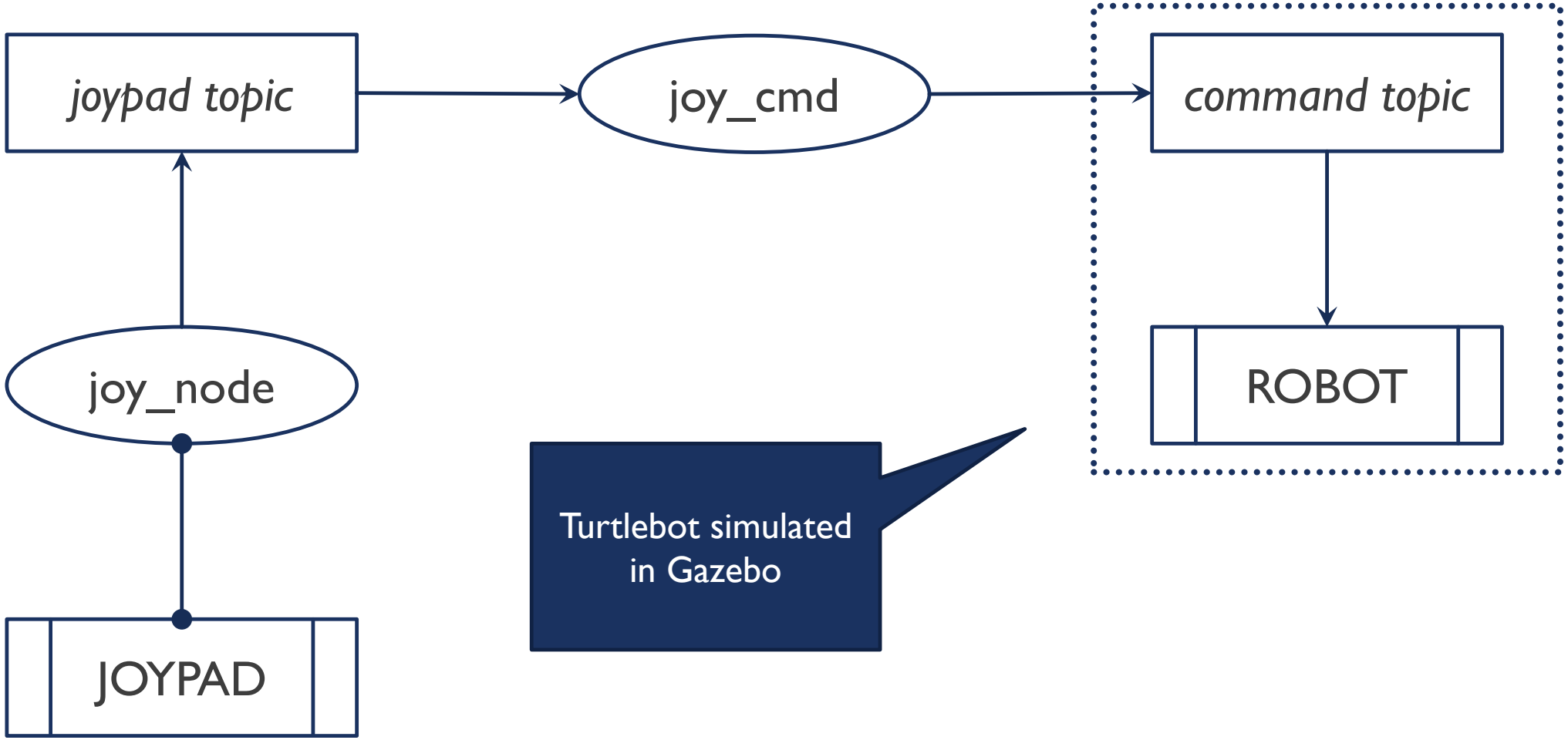
ROS ARCHITECTURE

ROBOTICS

POLITECNICO
MILANO 1863

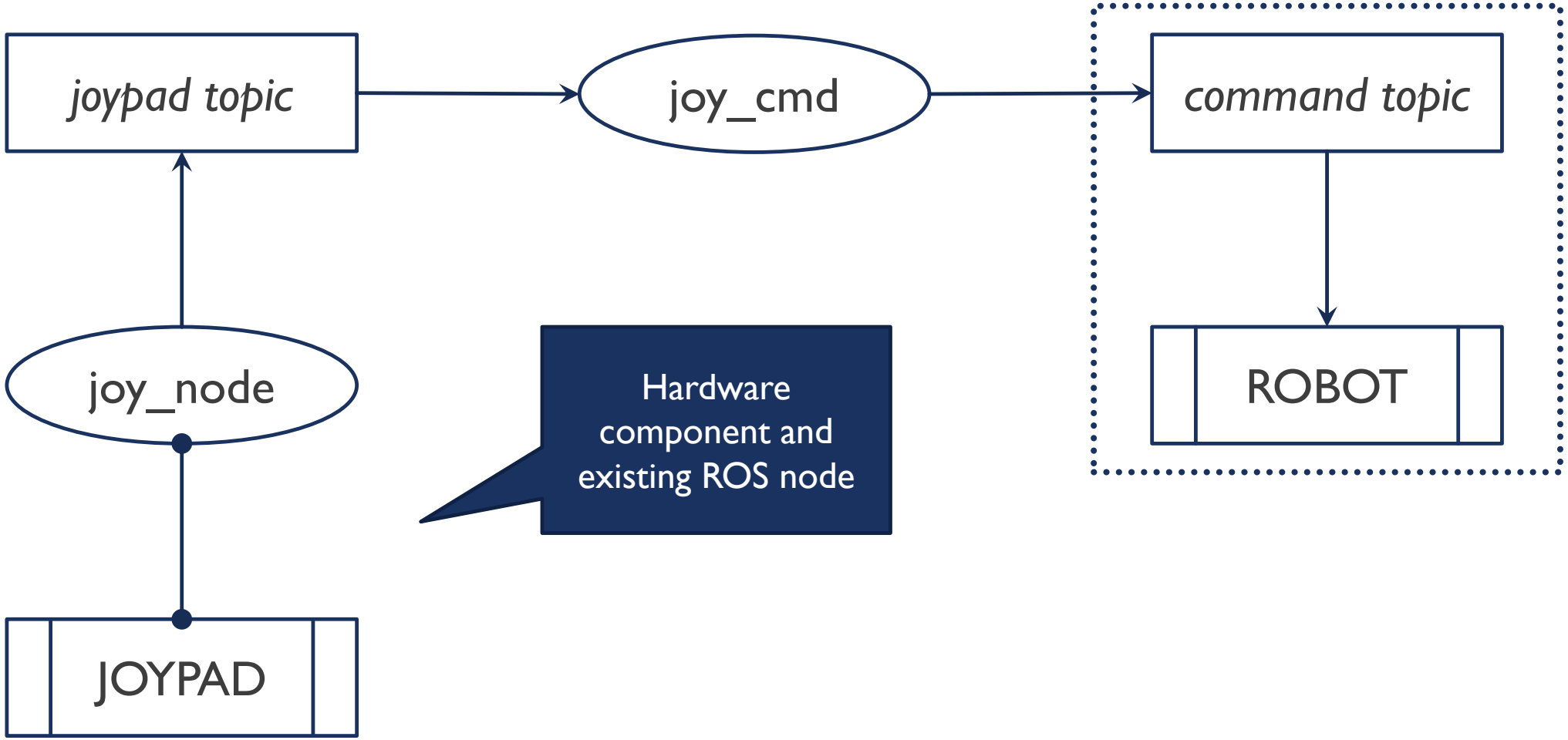
ROBOT TELEOPERATION

goo.gl/DBwhhC



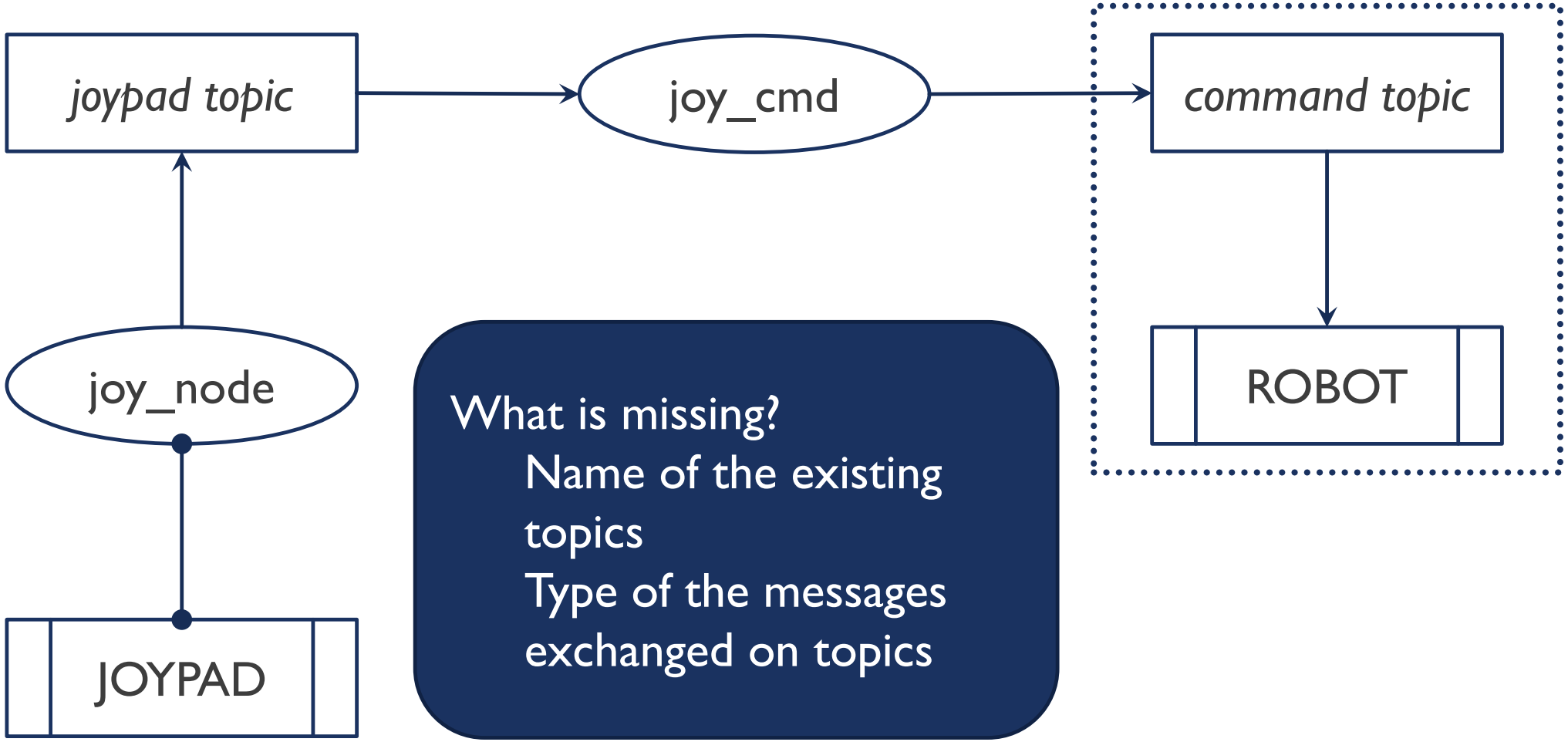
ROBOT TELEOPERATION

goo.gl/DBwhhC



ROBOT TELEOPERATION

goo.gl/DBwhhC

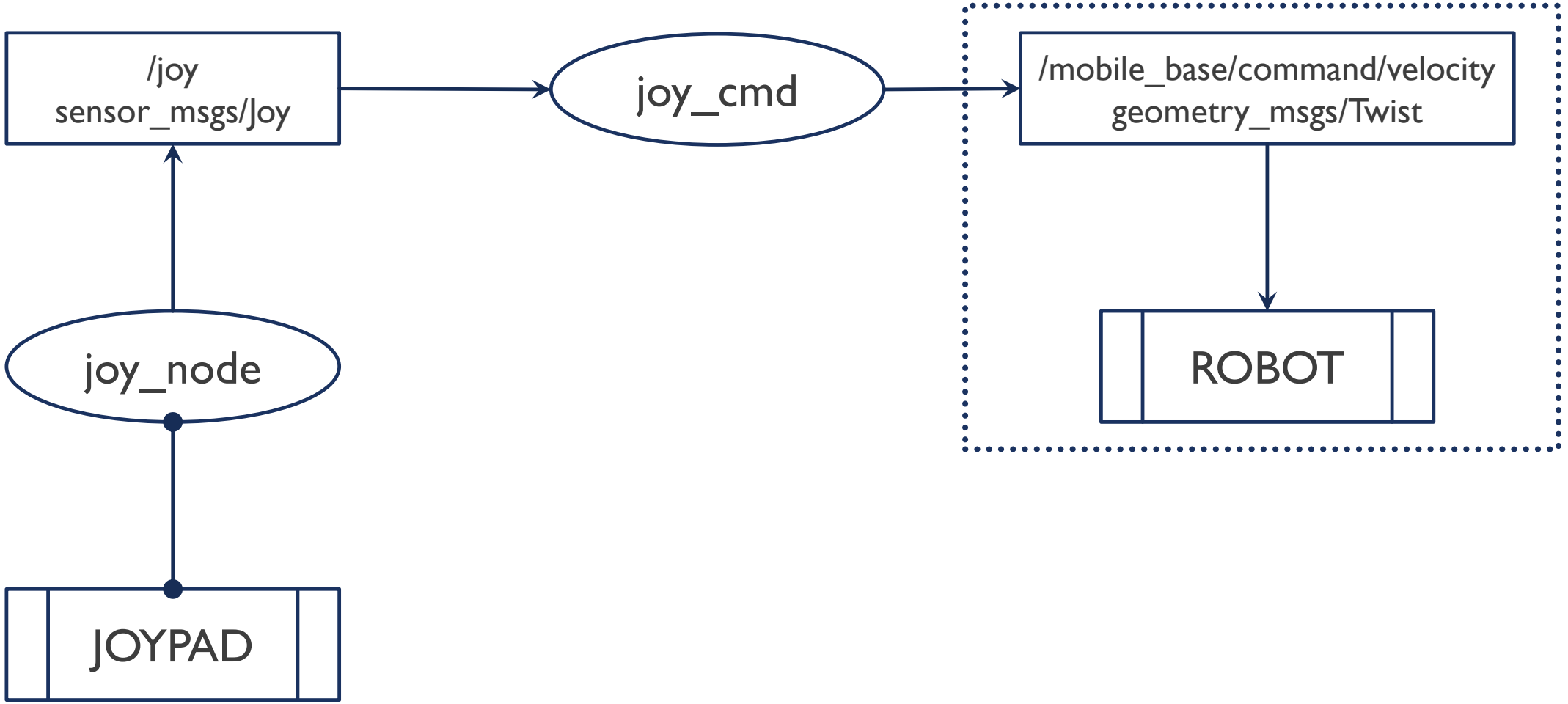


What is missing?

- Name of the existing topics
- Type of the messages exchanged on topics

ROBOT TELEOPERATION

goo.gl/DBwhhC



JOY_CMD IMPLEMENTATION

goo.gl/DBwhhC

What are the required elements?

JOY_CMD IMPLEMENTATION

goo.gl/DBwhhC

What are the required elements?

- Main loop (Spin vs SpinOnce)
- Subscriber (read from /joy)
- Publisher (publish on /cmd_vel)

JOY_CMD IMPLEMENTATION

goo.gl/DBwhhC

What are the required elements?

- Main loop (Spin vs SpinOnce)
- Subscriber (read from /joy)
- Publisher (publish on /cmd_vel)
- Timers? Parameters? Client/Server?

JOY_CMD IMPLEMENTATION

goo.gl/DBwhhC

Execution logic

Process the input received from the joypad and convert it in a velocity command for the robot

CMAKELISTS.TXT

goo.gl/DBwhhC

```
cmake_minimum_required(VERSION 2.8.3)
project(examples)
find_package(catkin REQUIRED COMPONENTS roscpp joy_msgs geometry_msgs)
catkin_package()

include_directories(include ${catkin_INCLUDE_DIRS})
add_executable(joy_cmd src/joy_cmd.cpp)
target_link_libraries(joy_cmd ${catkin_LIBRARIES})
```

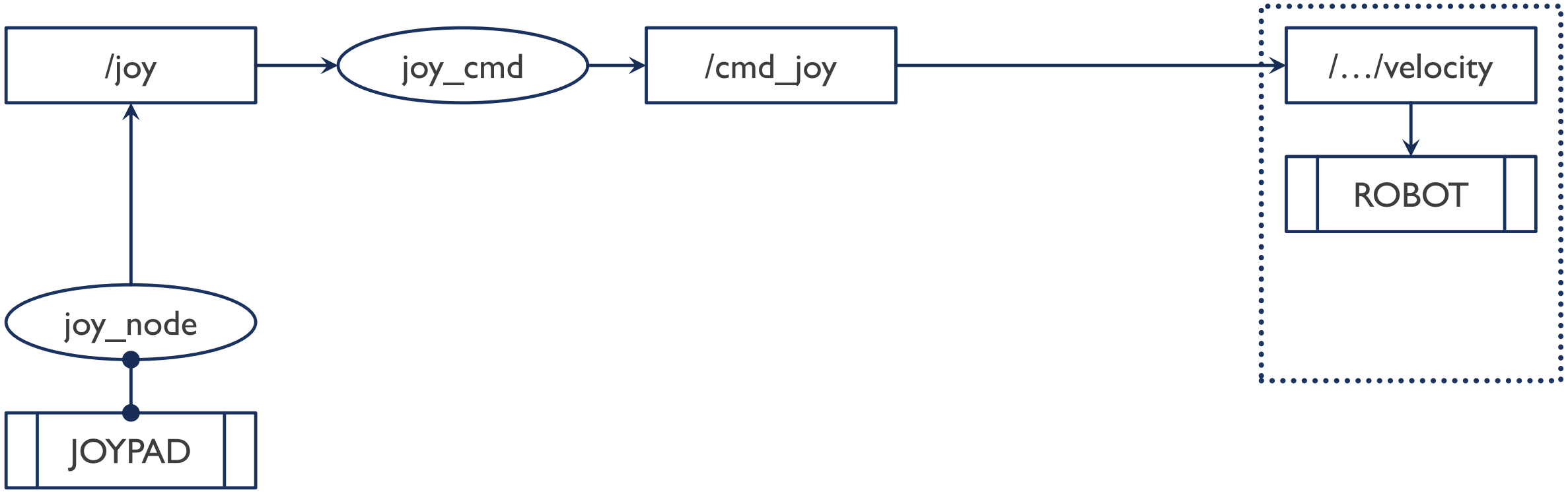
JOY_CMD IMPLEMENTATION

goo.gl/DBwhhC

Let's see the code!

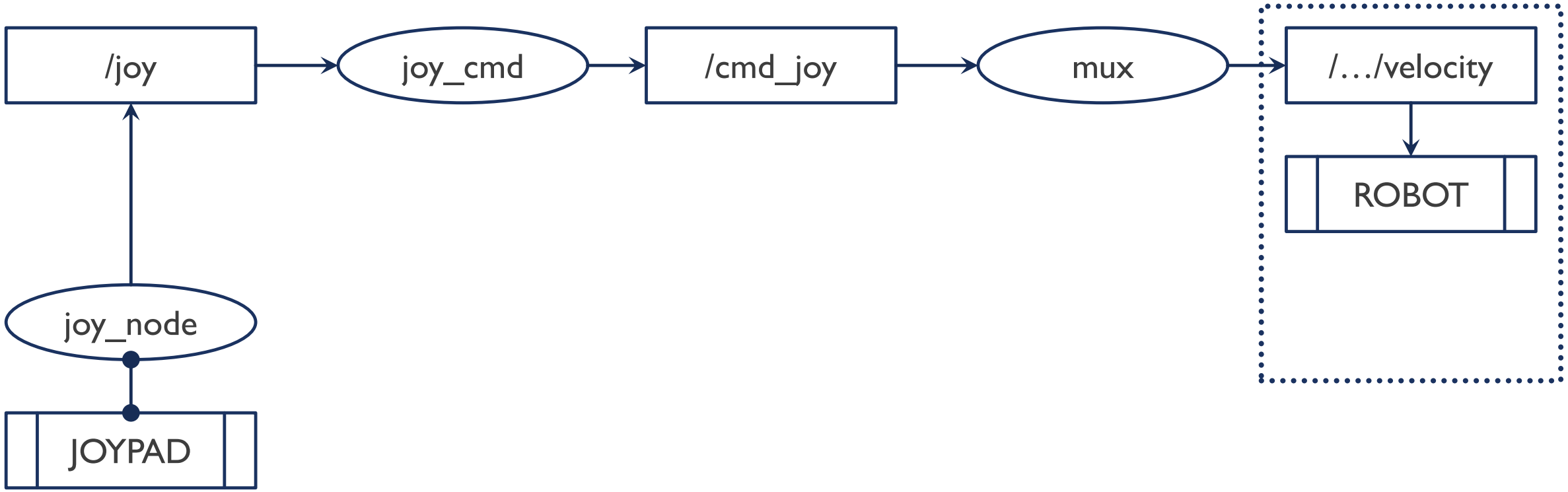
EXTENDING THE ARCHITECTURE

goo.gl/DBwhhC



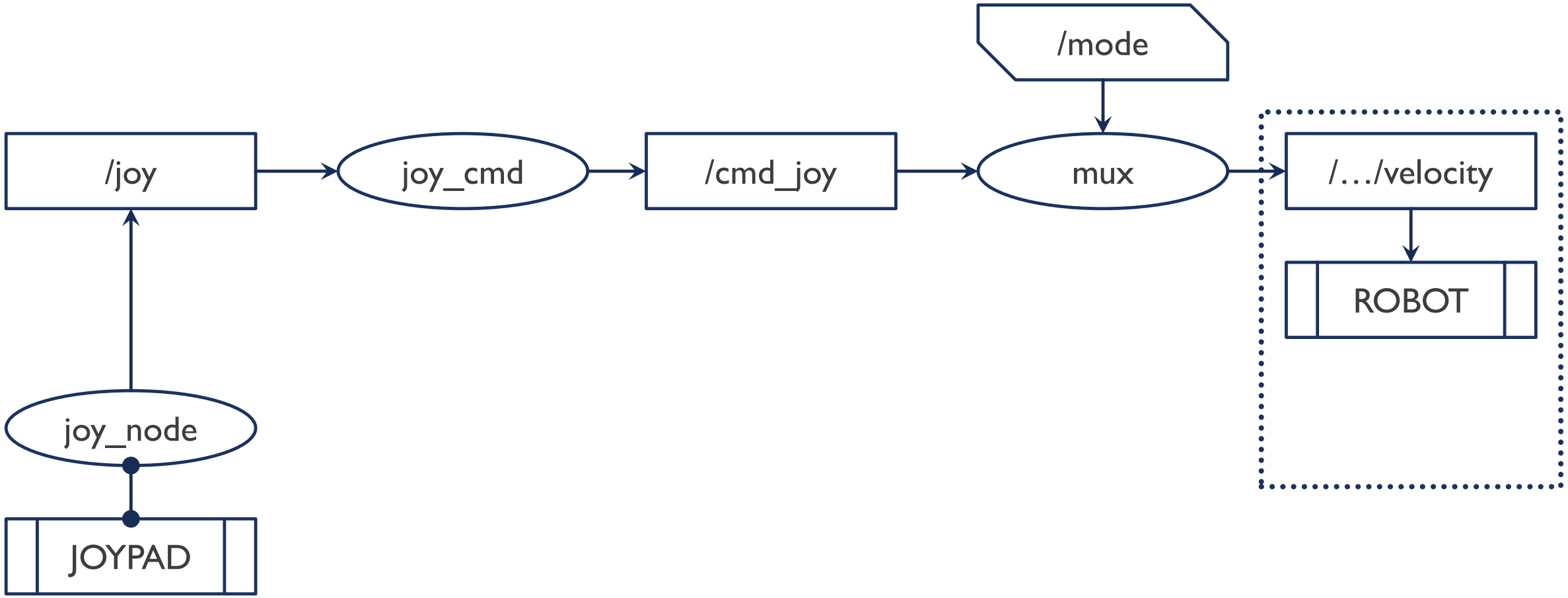
EXTENDING THE ARCHITECTURE

goo.gl/DBwhhC



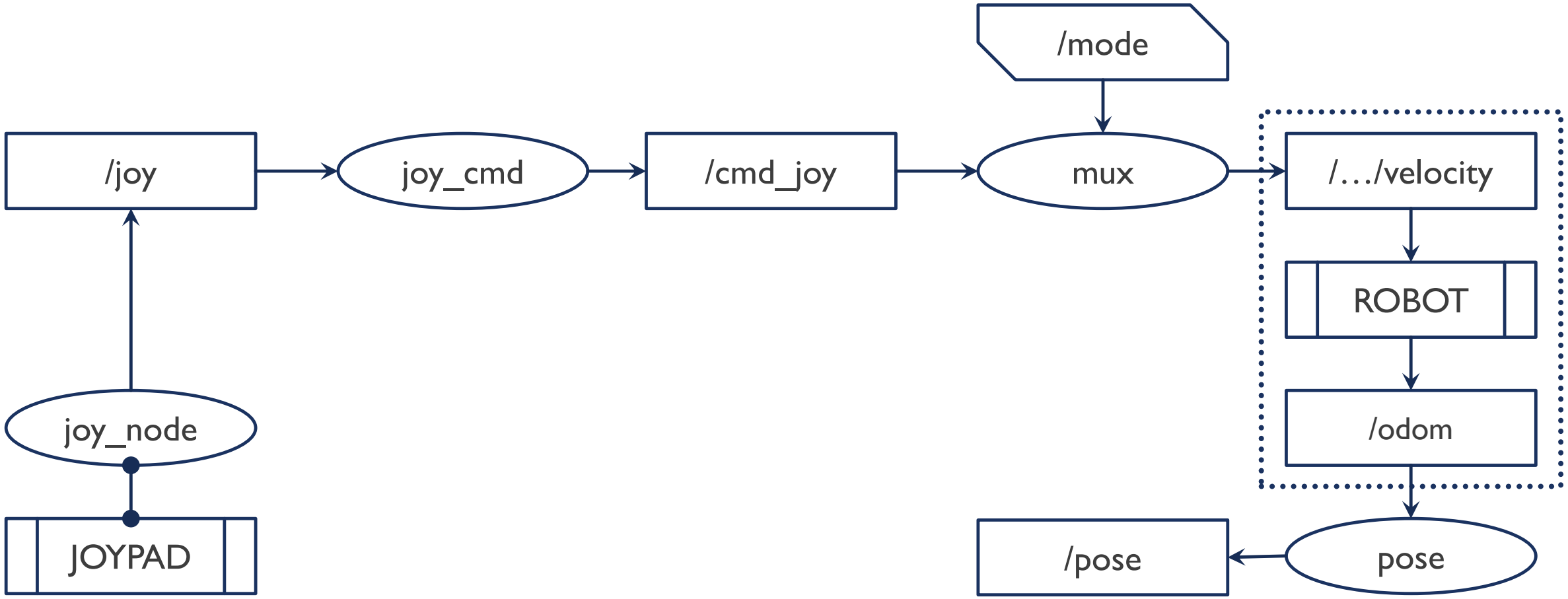
EXTENDING THE ARCHITECTURE

goo.gl/DBwhhC



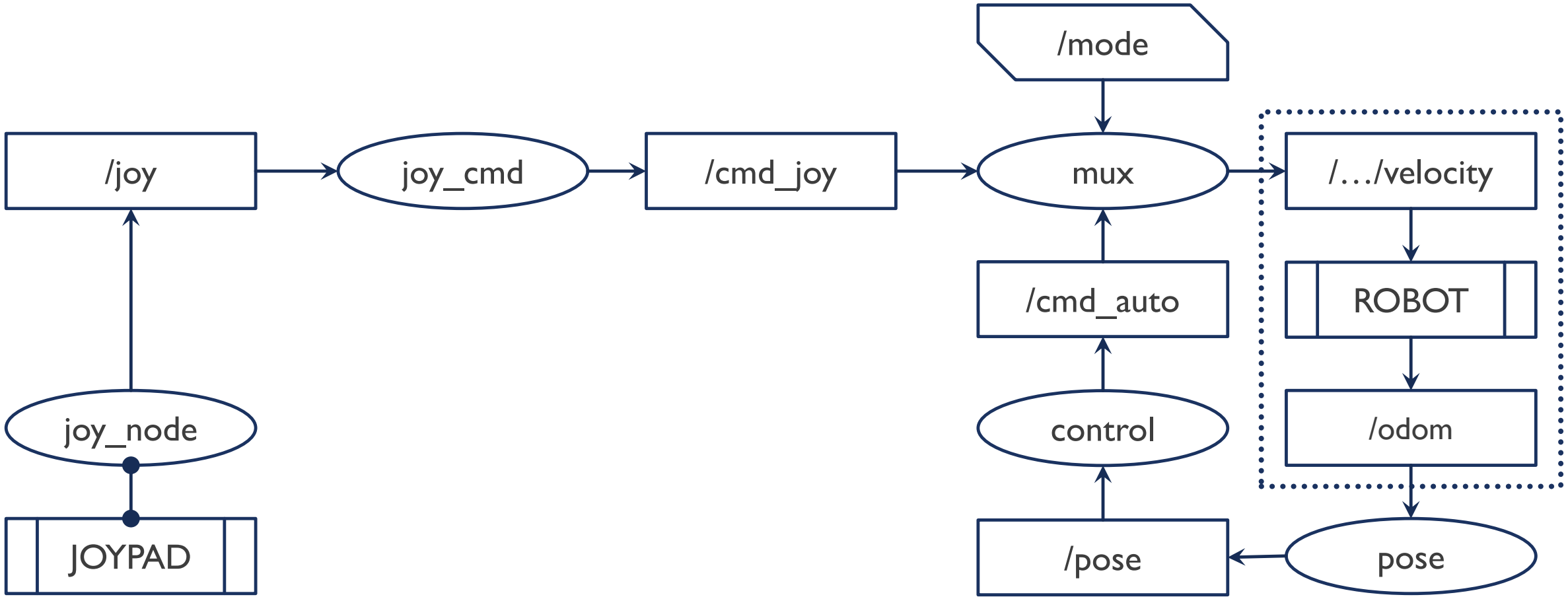
EXTENDING THE ARCHITECTURE

goo.gl/DBwhhC



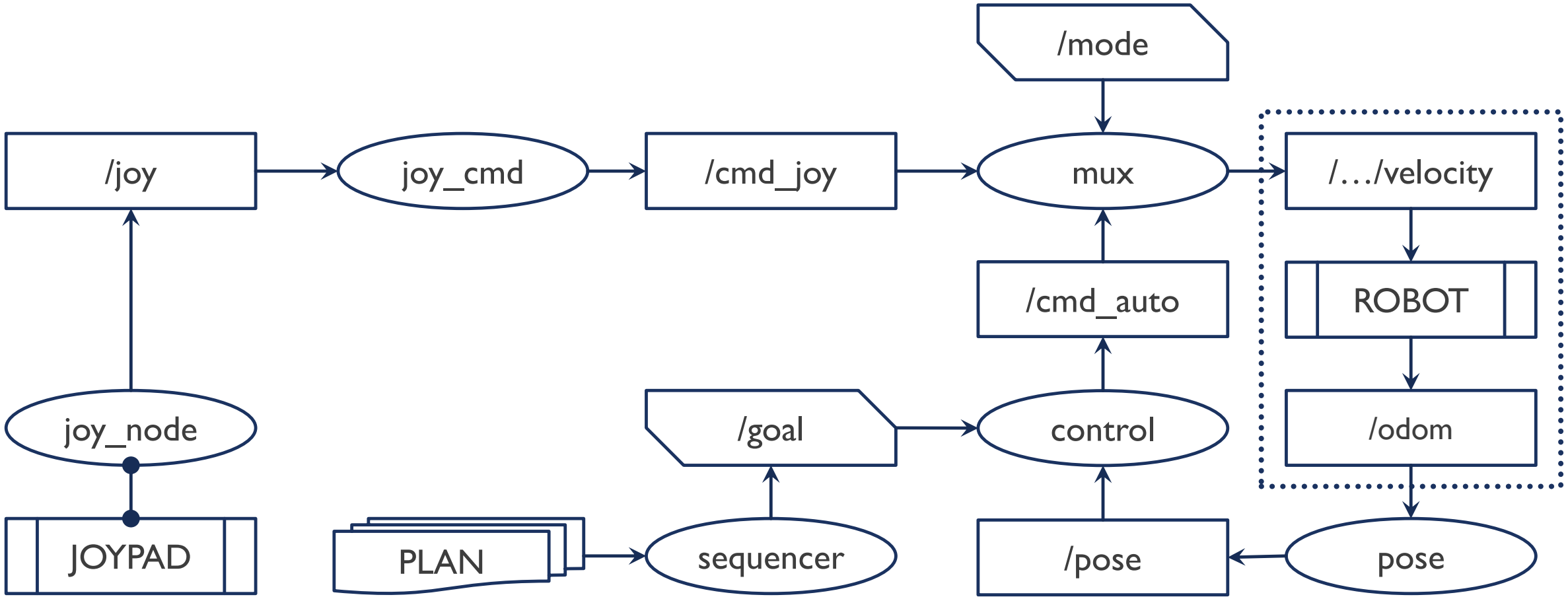
EXTENDING THE ARCHITECTURE

goo.gl/DBwhhC



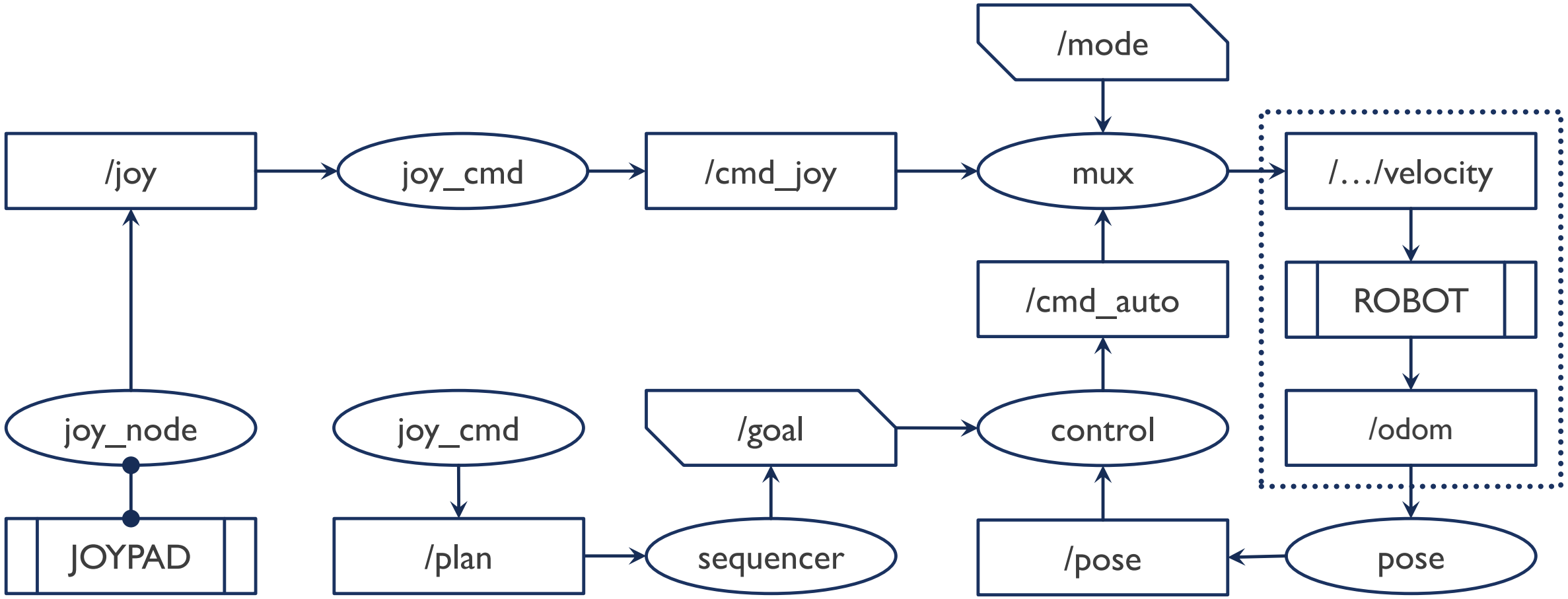
EXTENDING THE ARCHITECTURE

goo.gl/DBwhhC



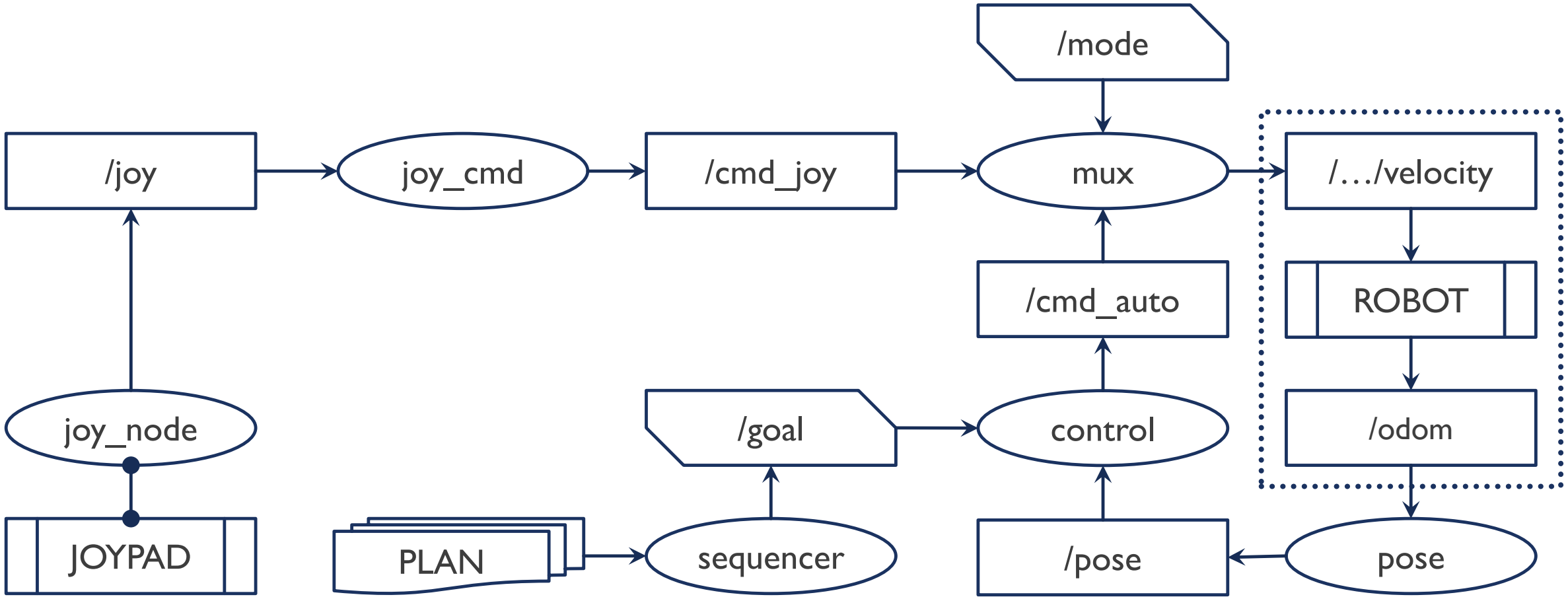
EXTENDING THE ARCHITECTURE

goo.gl/DBwhhC



EXTENDING THE ARCHITECTURE

goo.gl/DBwhhC



MUX

goo.gl/DBwhhC

Input: Two speed topic and a service with a mode (AUTO or MANUAL)

Output: The input of one of the two topic re-published

Logic: Select one of the two topic based on the value of the service

POSE

goo.gl/DBwhhC

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

goo.gl/DBwhhC

```
//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

goo.gl/DBwhhC

```
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

goo.gl/DBwhhC

```
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

goo.gl/DBwhhC

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

goo.gl/DBwhhC

```
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

goo.gl/DBwhhC

```
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;
}
```

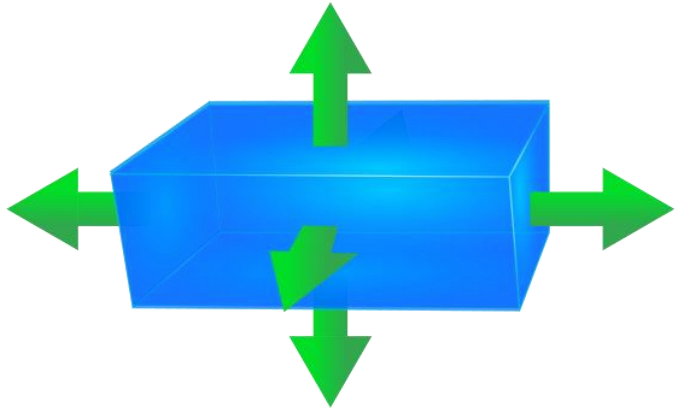
CONTROL

goo.gl/DBwhhC

```
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);
```

EULER ANGLES AND QUATERNIONS

goo.gl/DBwhhC

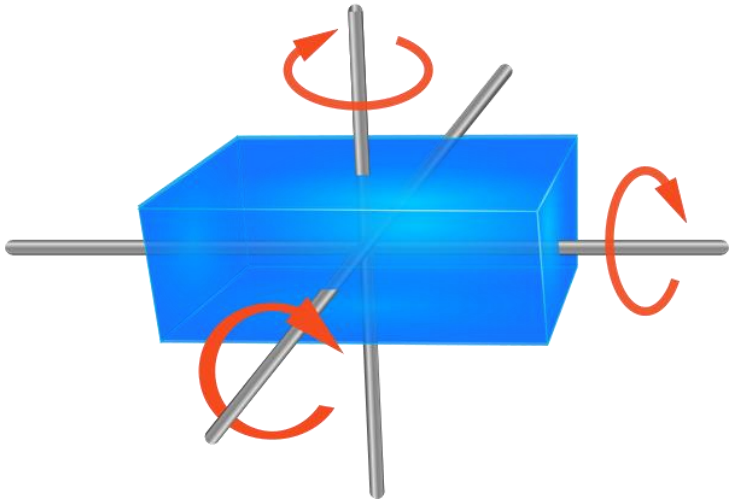


6 Degrees of freedom: 3 coordinates for the position, 3 coordinates for the orientation

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

goo.gl/DBwhhC

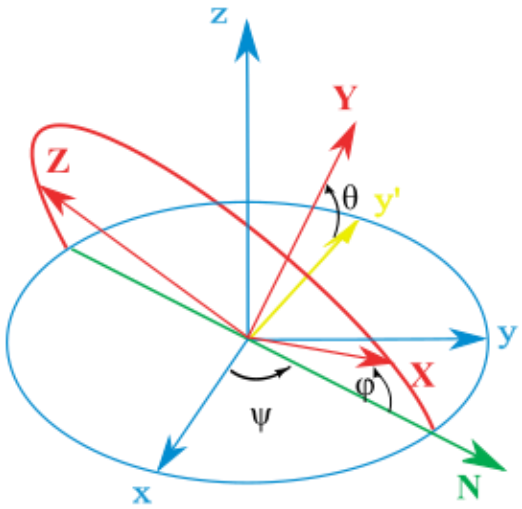
Orientation defined as a sequence of rotation around three axes

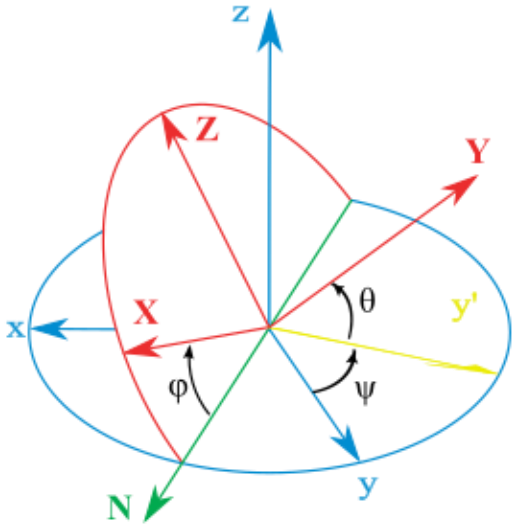
- **Euler**: first and last axis are the same (z-x-z, x-y-x, y-z-y,…)
- **Trait-Bryan**: three different axes (x-y-z, z-y-x, y-z-x,…)

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

goo.gl/DBwhhC

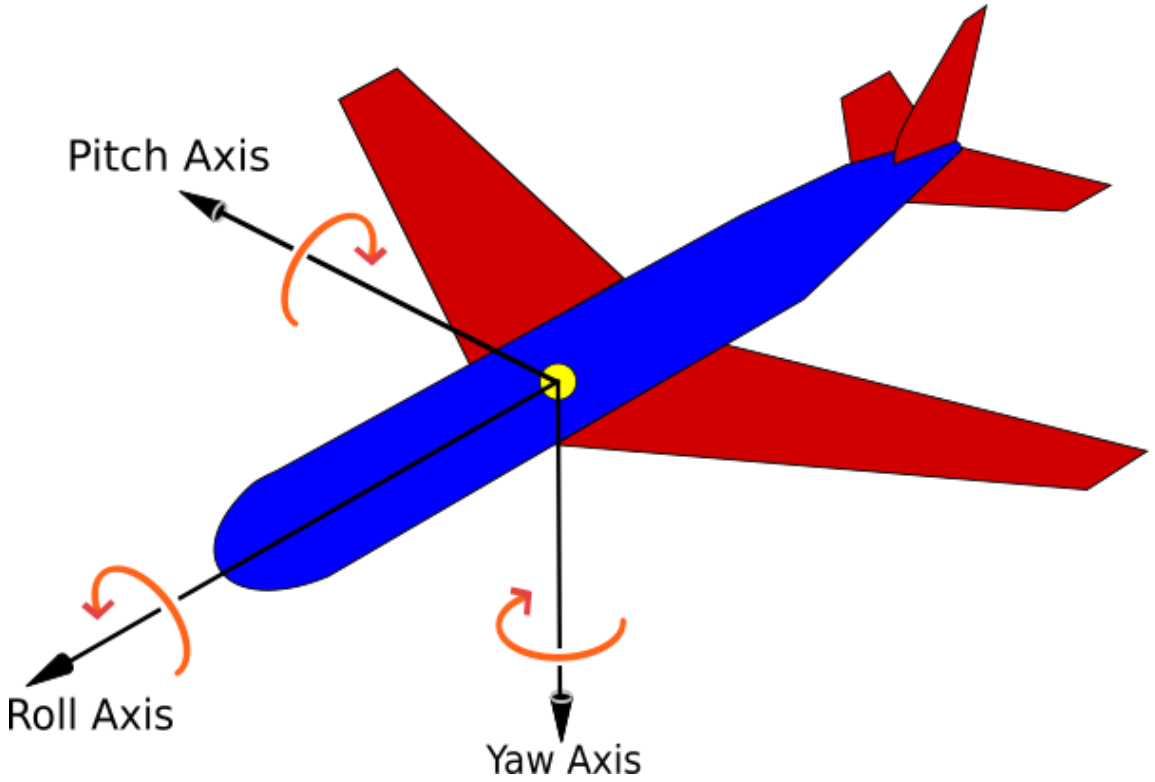
Orientation defined as a sequence of rotation around three axes

- **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

goo.gl/DBwhhC

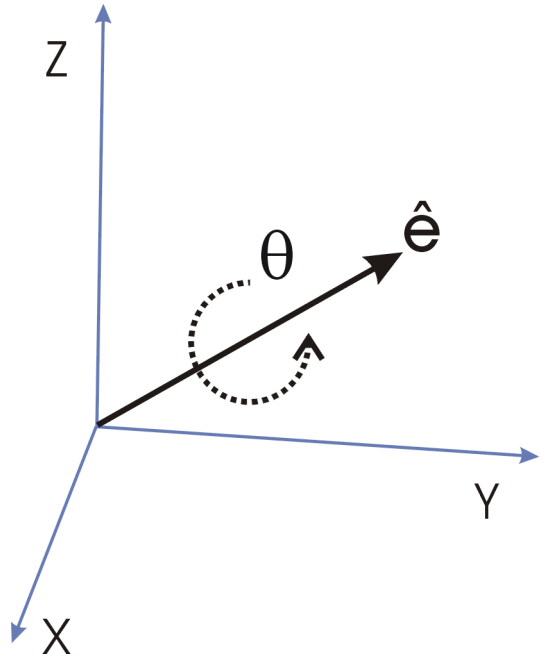
Four values used to define a vector in the 3D (x, y, z) space and a rotation around that axis (w).

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

goo.gl/DBwhhC

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 LAUNCH FILE

goo.gl/DBwhhC

```
<launch>
   <node pkg="diffdrive" type="joy_cmd" name="joy_cmd" />
   <node pkg="diffdrive" type="mux" name="mux" />
   <node pkg="diffdrive" type="control" name="control" />
   <node pkg="diffdrive" type="pose" name="pose">
      <param name="x" value="0.0"/>
      <param name="y" value="0.0"/>
      <param name="yaw" value="0.0"/>
   </node>
</launch>
```

EXTRA JUICE

goo.gl/DBwhhC

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

goo.gl/DBwhhC

1. Start the ROS framework

   ```
   roscore
   ```

2. Run gazebo as a ROS node

   ```
   roslaunch turtlebot_gazebo turtle_world.launch
   ```

3. Start the ROS nodes with the launch file

   ```
   roslaunch diffdrive diffdrive.launch
   ```