General conventions:

- NEVER use global namespaces to define the topics. For example for the joint state publisher the topic name has to be "joint_states" and not "/joint_states"
- Describe the kinematic using macros (urdf.xacro files)
- create configuration files for the parameters that could change depending on the hardware, for example the direction of the motor or the port of the device. You can load the parameters to the parameter server and use rosparam to get the value on your node, and if the parameter is not found give to it a default value (nh_.param<int>("MotorDirection",MotorDirection_, 1);)
- Divide the control and the model of the robot, the packages structure should looks:

```
ComponentName_control
        launch/   (if needed)
        config/   (if needed)
        src/
        include/   (if needed)
        CMakeLists.txt
        package.xml


ComponentName_description
        urdf/   (only .urdf.xacro files)
        meshes/   (if it is possible, only stl files)
        CMakeLists.txt
        package.xml
```

Control Driver:

For the control driver, at least, your should provide the following topics:

Publisher:
- joint_states : sensor_msgs/JointState

Subcribers:

Command(depending on your component you will need only one or all of them):
- position_controller/command : std_msgs/Float64Multiarray
- velocity_controller/command : std_msgs/Float64Multiarray
- trajectory_controller/command : control_msgs/FollowJointTrajectoryAction.h

Example:https://github.com/ipa320/cob_driver/blob/indigo_dev/cob_head_axis/ros/src/cob_head_axis.cpp

My **recommendation**, especially for the arm: use ros_control, it is standard, good documented and there are a lot of tools that we can reuse (for example **Moveit**!) : http://wiki.ros.org/ros_control

You have 2 levels, driver(hardware_interface) and controller.

For the driver you have to provide the following topics:

Publisher:

- joint_states : hardware_interface/joint_state_interface

Subcribers:

Command:

- hardware_interface/joint_command_interface

You have also to use the controller manager, to upload your controllers. As example you can take a look on cf_control code : https://github.com/squirrel-project/squirrel_robotino_arm/blob/indigo_dev/cf_control/src/cf_control_node.cpp

or, as extended example: ros_canopen https://github.com/ros-industrial/ros_canopen/tree/indigo-devel/canopen_motor_node (files https://github.com/ros-industrial/ros_canopen/blob/indigo-devel/canopen_motor_node/include/canopen_motor_node/robot_layer.h and https://github.com/ros-industrial/ros_canopen/blob/indigo-devel/canopen_motor_node/src/robot_layer.cpp)

For the controller level you need a configuration file and a launch file:

Example of config file:

```
 ------------------------------------------------------------

 ## joint_names
joint_names: [arm_1_joint, arm_2_joint...]

## control_mode_adapter
max_command_silence: 0.5

## joint_state_controller
joint_state_controller:
   type: joint_state_controller/JointStateController
   publish_rate: 50

## joint trajectory controller
joint_trajectory_controller:
  type: position_controllers/JointTrajectoryController
  joints:
     - arm_1_joint
     - arm_2_joint
...
  constraints:
     goal_time: 0.6
     stopped_velocity_tolerance: 0.05
     arm_1_joint: {trajectory: 0.1, goal: 0.1}
```

```
    arm_2_joint: {trajectory: 0.1, goal: 0.1}
...
  stop_trajectory_duration: 0.5
  state_publish_rate:  25
  action_monitor_rate: 10
  required_drive_mode: 7

## position controller
joint_group_position_controller:
  type: position_controllers/JointGroupPositionController
  joints:
     - arm_1_joint
     - arm_2_joint
...
  required_drive_mode: 1
arm_1_joint_position_controller:
  type: position_controllers/JointPositionController
  joint: arm_1_joint
  required_drive_mode: 1
arm_2_joint_position_controller:
  type: position_controllers/JointPositionController
  joint: arm_2_joint
  required_drive_mode: 1
...
```

Example of launch file:

```
<?xml version="1.0"?>
<launch>

	<group ns="$(arg component_name)">
		<!-- load controller configs -->
		<rosparam command="load" file=".....$(arg component_name)_controller.yaml"
/>
		<!-- start_controllers -->
		<node pkg="controller_manager" type="controller_manager" name="$(arg
component_name)_controller_spawner" args="spawn joint_state_controller" respawn="false"
output="screen"/>
	</group>
</launch>
```

The joint trajectory controller is required to use Moveit! and also to use some tools like **rqt** (graphical and intuitive tool to move the arm) , the advance of ros_control is that everything is already there you only have to publish the right topic.