ViperX 300 Robot Arm 6DOF

- website: https://www.trossenrobotics.com/viperx-300-robot-arm-6dof.aspx
- Hardware Setup
  - Don't rotate the arm more than 180°s either way
  - Flat surface
  - 先插電再插 usb
  - Do not connect or disconnect DYNAMIXEL when power is being supplied.

Specifications

http://support.interbotix.com/html/specifications/vx3006dof.html

- joint_order: [waist, shoulder, elbow, wrist_angle, wrist_rotate, gripper]
  - shadows: [shoulder_shadow, elbow_shadow]
- info_index_map: {'waist': 0, 'shoulder': 1, 'elbow': 2, 'forearm_roll': 3, 'wrist_angle': 4, 'wrist_rotate': 5}
- 6DOF: add forearm_roll
- X-Series Power Hub: http://support.interbotix.com/html/softwarefirmware/firmware/tutorials/changeu2d2.html?highlight=u2d2

Viper 300 6 DOF	
Degrees of Freedom	6
Reach	750mm
Total Span	1500mm
Accuracy	1mm
Working Payload	750g
Total Servos	9
Wrist Rotate	Yes

Joint	Min	Max	Servo ID(s)
Waist	-180	180	1
Shoulder	-101	101	2+3
Elbow	-101	92	4+5
Wrist Angle	-107	130	6
Forearm Roll	-180	180	7
Wrist Rotate	-180	180	8
Gripper	42mm	116mm	9

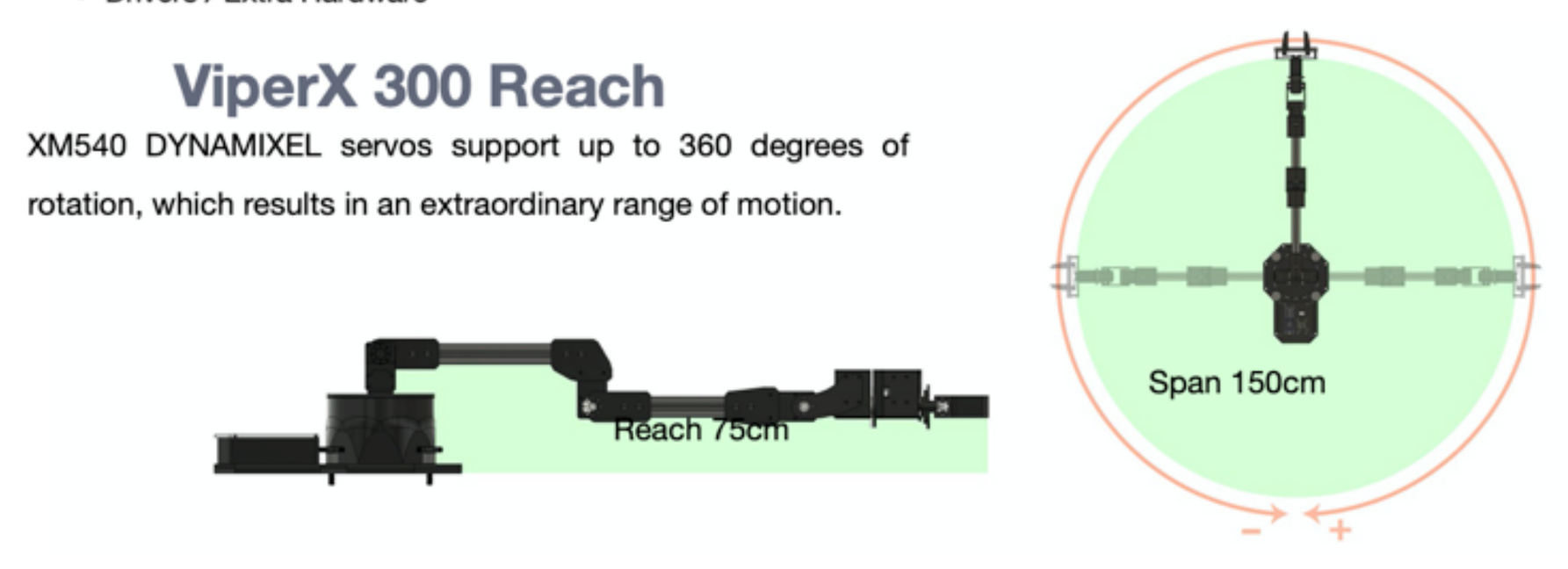
Product Features:

- XM540-W270 & XM430-W350 Servos
- Industrial Bearing Base
- Aluminum Construction
- U2D2 for Onboard Processing
- Gripper with Interchangeable Fingers
- 6 DOF

Includes:

- 7x XM540-W270 Actuators
- 2x XM430-W350 Actuators
- 1x DYNAMIXEL U2D2
- 1x CM9.04c
- 1x 12v10a Power Supply
- 1x USB2 Micro Cable
- ViperX 300 Hardware & Frames
- Extra Gripper Foam / Rubber
- Drivers / Extra Hardware

ViperX 300 Arm 6DOF	
Degrees of Freedom	6
Reach	750mm
Total Span	1500mm
Accuracy	1mm
Working Payload	750g
Total Servos	9
Wrist Rotate	Yes



ROS

- github:
  - interbotix_ros_manipulators: https://github.com/Interbotix/interbotix_ros_manipulators
  - interbotix_ros_toolboxes/…/interbotix_xs_modules/: https://github.com/Interbotix/interbotix_ros_toolboxes/tree/main/interbotix_xs_toolbox/interbotix_xs_modules/src/interbotix_xs_moduleshttps://github.com/Interbotix/interbotix_ros_arms/blob/master/interbotix_descriptions/urdf/vx300s.urdf.xacro
  - https://github.com/Interbotix/interbotix_ros_arms/blob/master/interbotix_descriptions/urdf/vx300s.urdf.xacro
- Robot name: `vx300s`
- `vx300s.yaml` specifies the names and initial register values for all the motors that make up a specfic robot arm: https://github.com/Interbotix/interbotix_ros_manipulators/blob/main/interbotix_ros_xsarms/interbotix_xsarm_control/config/vx300s.yaml
  - For a full explanation of each of these parameters, check out the Motor Config file template: https://github.com/Interbotix/interbotix_ros_core/blob/main/interbotix_ros_xseries/interbotix_xs_sdk/config/motor_configs_template.yaml.
  - The other file located in that directory is the Mode Config one (a.k.a mode.yaml). The parameters in there define the desired operating modes for either a group of joints or single joints, and whether or not they should be torqued on/off at node startup: https://github.com/Interbotix/interbotix_ros_core/blob/main/interbotix_ros_xseries/interbotix_xs_sdk/config/mode_configs_template.yaml.
- Get familiar with the virtual robot model by launching it in Rviz and playing with the joint_state_publisher.

```
$ roslaunch interbotix_xsarm_descriptions xsarm_description.launch robot_mod
```

- This package contains the necessary config files to get any of the many Interbotix X-Series arms working with MoveIt: https://github.com/Interbotix/interbotix_ros_manipulators/tree/main/interbotix_ros_xsarms/interbotix_xsarm_moveit
- To run this package on the physical robot, type the line below in a terminal:

```
$ roslaunch interbotix_xsarm_moveit xsarm_moveit.launch robot_model:=vx300s
```

Interbotix X-Series Arm Python API Demos

https://github.com/Interbotix/interbotix_ros_manipulators/tree/main/interbotix_ros_xsarms/examples/python_demos

ROS Command

- To get started, open up a terminal and type (assuming a WidowX 250 is being launched)…

```
$ roslaunch interbotix_xsarm_control xsarm_control.launch \
    robot_model:=vx300s
```

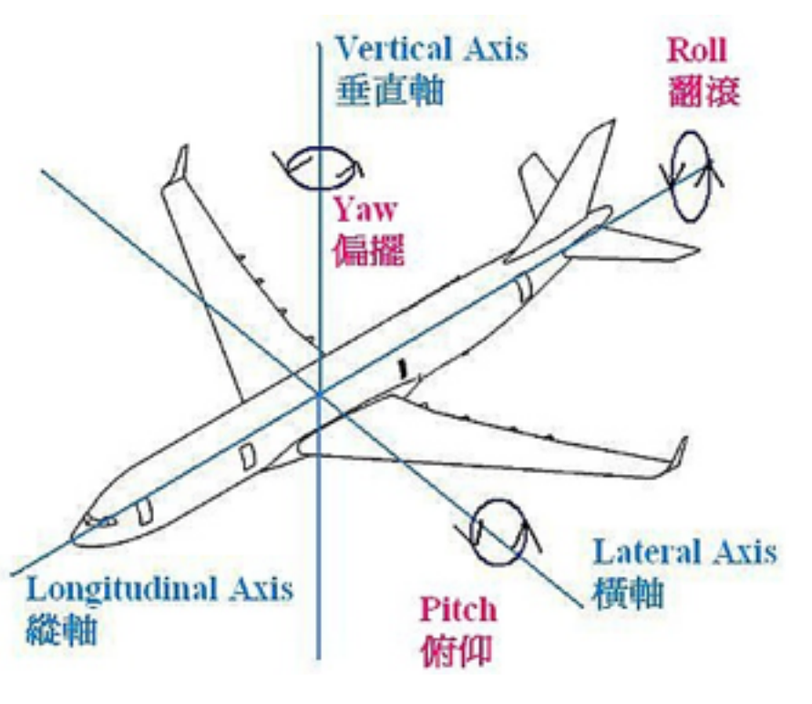
- Note, that if you want to test out your code first on a simulated arm, you can type…

```
$ roslaunch interbotix_xsarm_control xsarm_control.launch \
    robot_model:=vx300s use_sim:=true
```

- Then, in another terminal, navigate to this directory and type…

```
$ python bartender.py        # python3 bartender.py if using ROS Noetic
```

Pose Definition



- End-effector poses are specified from /<robot_name>/ee_gripper_link (a.k.a the 'Body' frame) to /<robot_name>/base_link (a.k.a the 'Space' frame).
- The 'red' axis is the 'X-frame', the 'green' axis is the 'Y-frame', and the 'blue' axis is the 'Z-frame'.
- The end-effector **should not be pitched past +/- 89 degrees**.
- In the code documentation, this transform is knows as `T_sb` (i.e. the transform that specifies the 'Body' frame 'b' in terms of the 'Space' frame 's').
- The way this information is stored is via a transformation matrix as shown below. (3-dimensional rotation matrix + translational position (i.e. xyz))

$$T = \begin{bmatrix} R & p \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & p_1 \\ r_{21} & r_{22} & r_{23} & p_2 \\ r_{31} & r_{32} & r_{33} & p_3 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- `robot.arm.set_ee_pose_matrix(T_sd)` : `T_sd` is what you should be specifying.
  - `T_sb` defines the current end-effector pose whereas `T_sd` defines the desired ('d') end-effector pose with respect to the 'Space' frame.
- `robot.arm.set_ee_pose_components(x, y, z, roll, pitch, yaw)` : specifically the **x, y, z, roll, pitch, and yaw** of the 'Body' frame with respect to the 'Space' frame (where x, y, and z are in **meters**, and roll, pitch and yaw are in **radians**).
- `robot.arm.set_ee_cartesian_trajectory` : accepts ***relative values*** **(x, y, z, roll, pitch, and yaw)** only.
  - The end-effector should follow as it travels from its current pose to the desired pose such that it moves in a **straight line**.
  - function arguments:
    - `moving_time` : the duration of the trajectory. The number of waypoints generated depends on it.
    - `wp_period` : the period of time between waypoints.
      - For example, if the whole trajectory should take 2 seconds and the waypoint period is 0.05 seconds, there will be a total of 2/0.05 = 40 waypoints.
    - `wp_moving_time` : the duration of time it should take for the arm joints to go from one waypoint to the next.
    - `wp_accel_time` : the time it should spend accelerating while doing so.
      - Together, they help to perform smoothing on the trajectory. If the values are too small, the joints will do a good job following the waypoints but the motion might be very jerky. If the values are too large, the motion will be very smooth, but the joints will not do a good job following the waypoints.
  - The end-effector poses are defined with respect to a virtual frame called `T_sy` . This frame has the exact same x, y, z, roll, and pitch as the 'Space' frame. However, it contains the **'yaw' of the 'Body' frame**.
- `robot.arm.get_ee_pose_command()` : get the latest commanded end-effector pose w.r.t the Space frame.
  - @return <4x4 matrix> - Transformation matrix
- `robot.arm.get_ee_pose()` : get the actual end-effector pose w.r.t the Space frame
  - @return <4x4 matrix> - Transformation matrix
- `robot.arm.set_single_joint_position(joint_name, position)` : Command a single joint to a desired position.
  - @param joint_name - name of the joint to control
  - @param position - desired position [rad]
- `robot.arm.capture_joint_positions()` : Resets self.joint_commands to be the actual positions seen by the encoders.
  - should be used whenever joints are torqued off, right after torquing them on again
  - get current `robot.arm.T_sb` & `robot.arm.joint_commands`
- `robot.arm.publish_positions(joints_positions)` : Helper function to publish joint positions and block if necessary.
  - Set actual positions of servos.
- `robot.arm.check_joint_limits(positions)` : Helper function to check to make sure the desired arm group's joint positions are all within their respective joint limits.
  - @param positions - the positions [rad] to check.
  - @return < bool > - True if all positions are within limits; False otherwise.
- `robot.arm.check_single_joint_limit(joint_name, position)` : Helper function to check to make sure a desired position for a given joint is within its limits.
  - @return < bool > - True if within limits; False otherwise.
- `robot.arm.go_to_home_pose()`
  - theta_list: [0, 0, 0, 0, 0, 0, 0]
    - joint_order: [waist, shoulder, elbow, wrist_angle, wrist_rotate, gripper]
  - Goes up and become right angle.
  - Caution!!! It might fall.
- `robot.arm.go_to_sleep_pose()`
  - theta_list: [0, -1.85, 1.55, 0, 0.8, 0, 0]
    - joint_order: [waist, shoulder, elbow, wrist_angle, wrist_rotate, gripper]
  - It looks like the thinker.

Gripper

- `class InterbotixGripperXSInterface(object)`
  - @param `gripper_pressure` - fraction from 0 - 1 where '0' means the gripper operates at 'gripper_pressure_lower_limit' and '1' means the gripper operates at 'gripper_pressure_upper_limit'
  - @param `gripper_pressure_lower_limit` - lowest 'effort' that should be applied to the gripper if gripper_pressure is set to 0; it should be high enough to open/close the gripper (~150 PWM or ~400 mA current)
  - @param `gripper_pressure_upper_limit` - largest 'effort' that should be applied to the gripper if gripper_pressure is set to 1; it should be low enough that the motor doesn't 'overload' when gripping an object for a few seconds (~350 PWM or ~900 mA)
- Gripper Open/Close Control
  - `robot.gripper.close(delay=2.0)` : close the gripper and then delay by seconds
  - `robot.gripper.open(delay=2.0)` : open the gripper and then delay by seconds
  - `robot.gripper.set_pressure(1.0)` : set pressure value in [0, 1], it then gives gripper PWM value `gripper_pressure_lower_limit + pressure * (gripper_pressure_upper_limit - gripper_pressure_lower_limit)`
  - PWM 值越大，gripper 開/閉 速度越快
- https://github.com/Interbotix/interbotix_ros_arms/tree/master/interbotix_descriptions/meshes/meshes_vx300s

---

DYNAMIXEL SDK

- ROBOTIS Customer Support: https://www.youtube.com/c/ROBOTISCS/featured
  - Zero to DYNAMIXEL: https://www.youtube.com/watch?v=3B6fmo9OFUc
  - FAQ: https://emanual.robotis.com/docs/en/faq/faq_dynamixel/
- DYNAMIXEL SDK:
  - https://emanual.robotis.com/docs/en/software/dynamixel/dynamixel_sdk/overview/#concept
  - http://wiki.ros.org/dynamixel_sdk
- XM Series: https://emanual.robotis.com/docs/en/dxl/x_m/
  - 6x XM540-W270: https://emanual.robotis.com/docs/en/dxl/x/xm540-w270/
  - 2x XM430-W350: https://emanual.robotis.com/docs/en/dxl/x/xm430-w350/
- Controller: DYNAMIXEL U2D2
  - https://www.trossenrobotics.com/dynamixel-u2d2.aspx
  - e-manual: http://support.robotis.com/en/product/auxdevice/interface/u2d2.htm
  - U2D2 is a small size USB communication converter that enables to control and to operate the DYNAMIXEL with the PC. It uses the USB cable to connect to the PC and prevents damage of the USB terminals. **It has both 3Pin connectors for TTL communication and 4Pin connectors for RS-485 communication embedded for easier control and access for the Dynamixel X series. (Supports UART as well.)** Requires the convertible cable for Molex connector using Dynamixels.
  - U2D2 does not supply power to the DYNAMIXEL, therefore, an external power supply should be used to provide power to the DYNAMIXEL.
  - signal interface
    - TTL: Models with 3P connectors (AX-Series, MX-Series, X-Series)
    - RS485: Models with 4P connectors (DX-Series, RX-Series, EX-Series, MX-Series, X-Series, PRO-Series)

Motor Control

- github (sample codes): https://github.com/ROBOTIS-GIT/DynamixelSDK
- control table (packet):
  - https://emanual.robotis.com/docs/en/dxl/x/xm430-w350/
  - https://www.youtube.com/watch?v=cQIDjrfDb24&list=PLEf1s0tzVSnS5rs4-Dh3qoZDw6mYLISc0&index=2

Position

- 0~4095
- +/-180: 0, -90: 1023, 0: 2048, 90: 3073
- [2048, 1023, 450]