MATLAB Example

Jump to bottom

Khoa Nguyen edited this page 9 days ago · 7 revisions

Controlling the Dobot from MATLAB

Please read before controlling the robot:

Sending commands through MATLAB when connected to the physical Dobot should result in movement. Ensure adequate signage is used, the environment is clear of obstacles, and personnel within the surrounding area are notified of your intentions. Additional measures outlined in your SWMS and RA should be compliant with the environment set up.

The robot should also be supervised at all times. When not in use, make sure that it is powered off.

Before sending any motion command to the robot, check the trajectory for potential collisions.

Ensure that you are connected to the ROS Master (if you are using a Window machine with a Raspberry Pi or other mini computers) by typing the following command to MATLAB command window:

```
rosnode list
```

Check if `/dobot_magician/dobot_magician_node` is presented.

MATLAB command

An example `DobotMagician` class is provided alongside with a MATLAB test script as an example of how to control the Dobot through MATLAB. The driver uses ROS topics to control different aspects of the robot, ranging from joint and end effector motion, to IO, conveyor belt, and linear rail commands.

You are free to modify or change the provided example MATLAB commands to suit your application.

Intialise connection

To initialise a connection to the ROS computer from MATLAB, call `rosinit` with the correct IP address specified. If you are running the driver directly from you PC, simply call `rosinit`

```
%% If using the UTS Pi, please check the IP address
rosinit('192.168.0.253');
%% Use this if the driver is on your Linux laptop
rosinit;
```

You can call `rosnode list` to check whether the `/dobot_magician/dobot_magician_node` exists. If not, ensure that:

- If you are using the driver from a seperate computer (Pi), make sure that the ROS computer is fully booted. The Pi usually takes 2 minutes or more to be fully booted.
- Your MATLAB computer can actually "see" the ROS Computer by pinging the IP address of the ROS computer.
- You don't have any firewalls that block the connection.

SAFETY STATUS

Safety status is a feature desgined to allow you to monitor the operation of the Dobot. There are 7 types of status and each of them is coded with a number for representation purpose:

- INVALID - 0
- DISCONNECTED - 1
- INITIALISING - 2
- ESTOPPED - 3
- **OPERATING - 4**
- PAUSED - 5
- STOPPED - 6

Get the current safety status of the robot

To get the current safety status of the robot

```
safetyStatusSubscriber = rossubscriber('/dobot_magician/safety_status');
pause(2); %Allow some time for MATLAB to start the subscriber
currentSafetyStatus = safetyStatusSubscriber.LatestMessage.Data;
```

Note: Before sending motion commands to the robot, you may want to check if the current safety status is 4 or not. If not then the robot may not react appropriately to the control signal. If that's the case then please re-intialise the robot.

Set Initialise Dobot

The Dobot would perform homing to reset the position of all of the encoders of all joints, and if it is set to use with a linear rail, the rail should also home with the robot as well.

To initialise the robot (or reintialise), publish a safety state message with the value of 2 (refer to the Safety State section for more information)

```
[safetyStatePublisher,safetyStateMsg] =
rospublisher('/dobot_magician/target_safety_status');
safetyStateMsg.Data = 2;
send(safetyStatePublisher,safetyStateMsg);
```

Note: When the robot is intialising please make sure that you are not sending other commands to the robot, including commands that read internal values or status of the robot.

Set "Software" Emergency Stop

Please note that this is just a "software" implementation of an emergency stop for the Dobot. After sending this command, if the robot remains in motions, please power down the robot.

When the e-stop command is called, the robot should stop all current motion, including the linear rail or the conveyor belt. It should also turn off all IO ports and reset all IO values to 0. To reset the Dobot, simply reinitialise it.

To e-stop the robot, publish a safety state message with value of 3 (refer to the Safety State section for more information)

```
[safetyStatePublisher,safetyStateMsg] =
rospublisher('/dobot_magician/target_safety_status');
safetyStateMsg.Data = 3;
send(safetyStatePublisher,safetyStateMsg);
```

ROBOT STATE

Get the current joint state

To get the current joint state from the real robot

```
jointStateSubscriber = rossubscriber('/dobot_magician/joint_states'); % Create a
ROS Subscriber to the topic joint_states
pause(2); % Allow some time for a message to appear
currentJointState = jointStateSubscriber.LatestMessage.Position % Get the latest
message
```

Check your connection if the latest message is empty

Get current end effector pose

Please note the end effector pose is the original end effector pose of the Dobot without any tool attached. So if you are using a specific tool for your Dobot, you may want to calculate the offset from the original end effector to the TCP of your tool and apply an additional transformation to get the exact end effector pose according to your tool.

To get the current end effector pose

```
endEffectorPoseSubscriber = rossubscriber('/dobot_magician/end_effector_poses');
% Create a ROS Subscriber to the topic end_effector_poses
pause(2); %Allow some time for MATLAB to start the subscriber
currentEndEffectorPoseMsg = endEffectorPoseSubscriber.LatestMessage;
% Extract the position of the end effector from the received message
currentEndEffectorPosition = [currentEndEffectorPoseMsg.Pose.Position.X,
                              currentEndEffectorPoseMsg.Pose.Position.Y,
                              currentEndEffectorPoseMsg.Pose.Position.Z];
% Extract the orientation of the end effector
currentEndEffectorQuat = [currentEndEffectorPoseMsg.Pose.Orientation.W,
                          currentEndEffectorPoseMsg.Pose.Orientation.X,
                          currentEndEffectorPoseMsg.Pose.Orientation.Y,
                          currentEndEffectorPoseMsg.Pose.Orientation.Z];
% Convert from quaternion to euler
[roll,pitch,yaw] = quat2eul(currentEndEffectorQuat);
```

Set target joint state

To set the target joint state for the dobot, you can publish a JointTrajectory message with a single joint position, as the driver currently does not support a joint trajectory (Future feature)

To send the target joint state

```
jointTarget = [0,0.4,0.3,0]; % Remember that the Dobot has 4 joints by default.

[targetJointTrajPub,targetJointTrajMsg] =
rospublisher('/dobot_magician/target_joint_states');
trajectoryPoint = rosmessage("trajectory_msgs/JointTrajectoryPoint");
trajectoryPoint.Positions = jointTarget;
targetJointTrajMsg.Points = trajectoryPoint;
```

```
send(targetJointTrajPub,targetJointTrajMsg);
```

Set target end effector state

To set target end effector pose (please note that this is also the original end effector pose of the Dobot without any tool attached), you can publish a Pose message.

To send the target end effector pose

```
endEffectorPosition = [0.2,0,0.1];
endEffectorRotation = [0,0,0];

[targetEndEffectorPub,targetEndEffectorMsg] =
rospublisher('/dobot_magician/target_end_effector_pose');

targetEndEffectorMsg.Position.X = endEffectorPosition(1);
targetEndEffectorMsg.Position.Y = endEffectorPosition(2);
targetEndEffectorMsg.Position.Z = endEffectorPosition(3);

qua = eul2quat(endEffectorRotation);
targetEndEffectorMsg.Orientation.W = qua(1);
targetEndEffectorMsg.Orientation.X = qua(2);
targetEndEffectorMsg.Orientation.Y = qua(3);
targetEndEffectorMsg.Orientation.Z = qua(4);

send(targetEndEffectorPub,targetEndEffectorMsg);
```

DEFAULT TOOL STATE

If you are using the provided suction cup or pincer gripper of the Dobot, the following commands allow you to read or set the current state of the tool (ON or OFF)

- ON - 1
- OFF - 0

Get current tool state of the robot

```
toolStateSubscriber = rossubscriber('/dobot_magician/tool_state');
pause(2); %Allow some time for MATLAB to start the subscriber
currentToolState = toolStateSubscriber.LatestMessage.Data;
```

Set tool state

```
% Turn on the tool
[toolStatePub, toolStateMsg] =
rospublisher('/dobot_magician/target_tool_state');
toolStateMsg.Data = true;
send(toolStatePub,toolStateMsg);
```

Note: this command would only work if you have the default end effector of the roobt attached properly with the vacuum connected.

IO PORTS

The IO data are defined as the IO pin type and value of all of the IO pins on the Dobot. The IO pin types (or multiplexing) are listed as follow, and similar to the safety status, they are also coded in numbers:

- INVALID - 0
- DIGITAL OUTPUT - 1
- PWM OUTPUT - 2
- DIGITAL INPUT - 3
- ADC INPUT - 4
- PULL-UP DIGITAL INPUT - 5
- PULL-DOWN DIGITAL INPUT - 6

There are 20 IO ports on the Dobot, to know which one is which and supports what types of IO, please refer to page 29 - 32 of the Dobot's Use Manual.

The default format of the received data is an array with size of 40. The first 20 values are the corresponding IO Multiplexing of 20 ports and the last 20 values represent the value of those respective ports.

Get current IO data of the robot

To get the current IO data of the Dobot

```
ioDataSubscriber = rossubscriber('/dobot_magician/io_data');
pause(2); %Allow some time for MATLAB to start the subscriber
ioMultiplex = ioDataSubscriber.LatestMessage.Data(2:21);
ioData = ioDataSubscriber.LatestMessage.Data(23:42);
```

Set IO port data

For example, to set port 1 to Digial Output with output value of HIGH

```
address = 1;
ioMux = 1;
data = 1;
[ioDataPub,ioDataMsg] = rospublisher('/dobot_magician/target_io_state');
ioData = [address,ioMux,data];
ioDataMsg.Data = ioData;
send(ioDataPub,ioDataMsg);
```

LINEAR RAIL

Get current position of the linear rail

To use the Dobot with the Linear Rail, first it needs to be configured to recognise the input from the rail. To do so, send the following command:

```
[railStatusPublisher, railStatusMsg] =
rospublisher('/dobot_magician/target_rail_status');
railStatusMsg.Data = true;
send(railStatusPublisher,railStatusMsg);
```

You can also turn off the use of linear rail by sending a "false" status using the above command.

NOTE: SENDING A FALSE STATUS TO TURN OFF THE USE OF THE LINEAR RAIL WOULD ONLY REMOVE THE ABILITY TO CONTROL THE POSITION OF THE RAIL. IF YOU REINITIALISE THE ROBOT, IT WOULD PERFORM HOMING WITH THE RAIL AS EVEN THOUGH THE RAIL STATUS IS SET TO FALSE.

RIGHT NOW IT IS UNCLEAR WHAT CAUSE THIS ISSUE, AS WE NEED SOME TIME TO INVESTIGATE. SO PLEASE MAKE SURE TO KEEP THIS IN MIND WHENEVER YOU ARE USING THE ROBOT WITH THE LINEAR RAIL.

It is recommended that you reinitialise the Dobot, since the linear rail doesn't store its' position internally. Therefore, similar to the Dobot, it is better to reintialise the Dobot. Once the Dobot has acknowledged that the linear rail is in use, it should perform the homing sequence with the rail as well. Refer to the Initialise Section to see how to intialise the Dobot.

After reintialising, you can get the current position of the linear rail using the following command:

```
railPositionSubscriber = rossubscriber('/dobot_magician/rail_position');
pause(2); %Allow some time for MATLAB to start the subscriber
currentRailPosition = railPositionSubscriber.LatestMessage.Data;
```

Set linear rail position

Please refer to section **Get current position of the linear rail** for how to setup the robot with the linear rail. The robot needs to be configured with the linear rail before the following commands can be used:

```
position = 0.5 % Move to the position of 0.5
[railPosPub,railPosMsg] = rospublisher('/dobot_magician/target_rail_position');
railPosMsg.Data = position;
send(railPosPub,railPosMsg);
```

CONVEYOR BELT

Set Conveyor belt velocity

To set the velocity for the conveyor belt, please send the follow command:

```
enableConveyor = true % Enable the conveyor belt
velocity = 15000 % this velocity is the number of ticks per second of the
stepper motor that moves the conveyor belt.
[beltPub,beltMsg] = rospublisher('/dobot_magician/target_e_motor_state');
beltMsg.Data = [enableConveyor,velocity];
send(beltPub,beltMsg);
```

Note: The velocity of the belt is not in meter per second. It is the number of pulse of the stepper motor that controls the belt produces. You have to work out a way to convert pulse per second to meter per second if you really need to use that value. Furthermore, it is recommened that the velocity should be less that 15000.

Pages 4

Find a Page…

Home

Instructions For Native Linux

Instructions For Raspberry Pi 3 And DobotRobot

MATLAB Example

Clone this wiki locally