TDDE05: Lab 0: Introduction to ROS

Cyrille Berger

January 25, 2021

Contents

| Intro | oduction | 1 | | | | | |
|---------------------|--|---|--|--|--|--|--|
| 2 Running | | | | | | | |
| 2.1 | On Campus | 2 | | | | | |
| 2.2 | thinlinc | 2 | | | | | |
| ROS | | 2 | | | | | |
| 3.1 | middleware | 2 | | | | | |
| 3.2 | catkin | 2 | | | | | |
| Sim | ılator | 2 | | | | | |
| Programming a robot | | | | | | | |
| 5.1 | Get the base code | 3 | | | | | |
| 5.2 | Start ROS and the simulator | 4 | | | | | |
| 5.3 | RViz | 4 | | | | | |
| 5.4 | Get started with the command line | 5 | | | | | |
| | 5.4.1 rostopic | 5 | | | | | |
| 5.5 | Create a package | 7 | | | | | |
| 5.6 | Saving rviz configuration | 7 | | | | | |
| 5.7 | Basic Husky Controller (in Python) | 7 | | | | | |
| 5.8 | Get started with GNU Screen | 9 | | | | | |
| 5.9 | Creating user interface with ROS | 11 | | | | | |
| 5.10 | Demonstration | 11 | | | | | |
| | Intro Rum 2.1 2.2 ROS 3.1 3.2 Simu 5.1 5.2 5.3 5.4 5.5 5.6 5.7 5.8 5.9 5.10 | Introduction Running 2.1 On Campus 2.2 thinlinc 2.2 thinlinc ROS 3.1 middleware 3.2 catkin 3.2 catkin Simulator Programming a robot 5.1 Get the base code 5.2 Start ROS and the simulator 5.3 RViz 5.4 Get started with the command line 5.4.1 rostopic 5.5 Create a package 5.6 Saving rviz configuration 5.7 Basic Husky Controller (in Python) 5.8 Get started with GNU Screen 5.9 Creating user interface with ROS 5.10 Demonstration | | | | | |

1 Introduction

"The Robot Operating System (ROS) is a flexible framework for writing robot software. It is a collection of tools, libraries, and conventions that aim to simplify the task of creating complex and robust robot behavior across a wide variety of robotic platforms" [2]. It is commonly use by many robots in research and industry. ROS can transparently interract with real hardware or with a simulator.

The objective of this lab is to facilate first contact to the ROS framework.

2 Running

2.1 On Campus

ROS is already installed on IDA computers, you can find it in the directories */opt/ros/melodic*, */usr* and */courses/TDDE05/software*.

2.2 thinlinc

Running on personnal laptops is currently not supported, however it is possible to use thinlinc to access remotely IDA's computers. To use thinlinc, follow the installation instruction at http://www.cendio.com/thinlinc/download and connect to the IDA server: *thinlinc.edu.liu.se*.

3 ROS

ROS is a framework composed of a communication middleware and a set of modules providing different functionnalities for robots.

3.1 middleware

With ROS the various algorithms, interfaces to hardware and simulators all run in different processes. ROS include communication libraries for various programming languages: C++ (roscpp http://wiki.ros.org/roscpp) or Python (rospy http://wiki.ros.org/roscpp). There are libraries for more programming languages, but they are not as well supported and it is recommended that you stick to C++ and Python for this course.

ROS programs mainly communicate through the use of topic. ROS topics follow a multi-publishers/multi-subscribers pattern: ROS programs subscribe to individual topics and they will then receive the messages that are published by other programs on the topic. Topics are associated to a specific type of a ROS message (examples of standard messages can be found at http://docs.ros.org/kinetic/api/std_msgs/html/ index-msg.html, it is possible to create custom messages to suit your needs).

In addition to topics, ROS programs can offer *service calls*, which is the ROS implementation of Remote Procedure Calls (RPC).

3.2 catkin

ROS also aim to provide a standard method for developing and distributing packages using a tool called *catkin* which relies on the *cmake* build system.

4 Simulator

This year we use *simple simulator*, a very basic 2D simulator, which can run well in thinlinc.

5 Programming a robot

5.1 Get the base code

- 1. First you will need to add the following to your .bashrc file:
- alias start-tdde05=". /courses/TDDE05/software/bin/start-tdde05.sh"

In every terminal, before issuing a ROS command, you will need to run:

```
1 start-tdde05
```

2. Then create a ROS workspace for your project:

```
1 mkdir -p ~/TDDE05/catkin_ws/src
2 cd ~/TDDE05/catkin_ws/
```

3 catkin init

You need to use ~/TDDE05/catkin_ws/ as a directory for the start-tdde05 script to work properly!

3. Then you need to create a ~/TDDE05/.ros_port file. This file will be used to set the port number used for communication by ROS. When running on thinlinc, the port number need to be different for each user. In the following command, replace XX with your group number:

```
echo "112XX" > ~/TDDE05/.ros_port
start-tdde5
```

4. Then get the code of your project:

```
1 cd ~/TDDE05/catkin_ws/src
2 mkdir air_labs
3 cd air_labs
4 git init
```

Once you have gotten access to your gitlab repository (replace XX with your group number and YY with the correct year number), **the repositories might not be created yet once you start the labs**:

```
git remote add origin git@gitlab.liu.se:tdde05-20YY/air-labs-XX.git
git push -u origin master
```

Next time you want to access the code from a new account:

```
1 cd
```

- git clone git@gitlab.liu.se:tdde05-20YY/air-labs-XX.git air_labs
- 5. To build the project (do not forget start-tdde05 if you are using a new terminal):
- 1 cd ~/TDDE05/catkin_ws
- 2 catkin build

You can run the catkin build command from any directory under \~/TDDE05/catkin_ws.

6. You are now ready to start programming your robot!

5.2 Start ROS and the simulator

In two different terminals:

- Start the ROS middleware:
- 1 start-tdde05
- 2 roscore
 - Start the simple simulator:

```
1 start-tdde05
```

2 rosrun air_simple_sim simple_sim.py __ns:=/husky0

5.3 RViz

To be able to see what is happening in the simulated world, we can use a tool called rviz:

```
1 start-tdde05
2 rviz
```

When it opens it shows a window like in figure 1. rviz can be used to display the location of a robot and all the sensor data. For now, we will use it to display the location of the robot:

- Press on the button Add, at the bottom of the Displays panel
- It should show a dialog with a tab called By display type (see figure 2)
- Select the item TF (circled in red in figure 2)
- Click on Ok



Figure 1: RViz empty window

| te visualization ly display type | | | |
|---|---|--|--|
| iy display type | | | |
| | By topic | | |
| Marker MarkerArr Odometry Path PointClour PoseArray PoseArray RelativeH RelativeH RelativeH TF Temperat VenchSt | ay j j22 ped covariance umidity el el urre amped | | |

Figure 2: RViz Add dialog

• In the Displays panel, make sure that Fixed Frame is set to odom

The rviz window should look like on figure 3. There are currently two frames, odom corresponding to the origin of the world and husky0/base_footprint corresponding to the location of the robot.

5.4 Get started with the command line

5.4.1 rostopic

rostopic is a command line tool for interracting with topics. It can be used to get information about a topic, such as the type, publishers and subscribers:

```
1 rostopic info /husky0/odometry
```

rostopic can be used to read what is a published on a topic. The following command can be used to display what the observation of the wheel velocity:



Figure 3: RViz with TF

```
1 rostopic echo /husky0/odometry
```

rostopic can be used to publish data on a topic:

```
rostopic pub [topicname] [topictype] [value in yaml]
```

For example to command some velocity to your robot, first you can check the topic type with. You can use tab completion to help with finding topicname, tyopictype and formatting the yaml value.

```
rostopic info /husky0/wheel_velocity_cmd
```

The sensor_msgs/JointState (http://docs.ros.org/api/sensor_msgs/html/ msg/JointState.html) message has the following structure:

```
1 Header header
2
3 string[] name
4 float64[] position
5 float64[] velocity
6 float64[] effort
```

The YAML format use a key/value structure, and you do not need to specify all the field for your message, the following command will make your robot move in straight line with the maximum velocity of 1.0m/s.

```
rostopic pub /husky0/wheel_cmd sensor_msgs/JointState "name: ['left', 'right']
effort: [1, 1]"
```

It essentially tells the robot to apply full power on the left and right wheels. You can check the velocity of your robot with:

```
1 rostopic echo /husky0/odometry
```

You can check the velocity of the wheels of your robot with:

```
rostopic echo /husky0/wheel_velocities
```

5.5 Create a package

We will use the command *catkin_create_pkg* to create a new package for the tool that we are going to write in this lab.

```
1 start-tdde05
```

```
2 cd ~/TDDE05/catkin_ws/src/air_labs
```

```
3 catkin_create_pkg air_lab0 rospy
```

This will create a package called *air_lab0* which depends on *rospy*. In practice, it creates a directory called *air_lab0* containing two files:

Then after creating a package, even if it is only python, you need to *build it* at least once and reload the environment:

1 catkin build

- 2 start-tdde05
 - 1. *package.xml*: meta information about your package: authors, license and more importantly ROS packages dependencies
 - 2. *CMakeLists.txt*: this is the file used by the build system to compile your ROS programs. This file contains a lot of documentation on how to use it.

5.6 Saving rviz configuration

You can now save the rviz configuration in your air_lab0 package. Go in the menu File and save your configuration in

~/TDDE05/catkin_ws/src/air_labs/air_lab0/rviz/labs.rviz. You can use to start rviz with your configuration file:

rviz -d `rospack find air_lab0`/rviz/labs.rviz

5.7 Basic Husky Controller (in Python)

In this part, you will write a basic controller that takes a string as input and send some simple effort to the simulator.

We will listen on a topic called /husky0/text_command using a std_msgs/String message. You can use the following command to see the structure of the message:

1 rosmsg info std_msgs/String

```
1 cd ~/TDDE05/catkin_ws/src/air_labs/air_lab0/src
```

2 touch simple_text_controller.py

3

chmod u+x simple_text_controller.py

Then in your favorite text editor:

```
#!/usr/bin/env python
 1
 2
    import math
 3
 4
    # rosoy is the main API for ROS
 5
 6
    import rospy
 7
    # import library with ros messages
 8
    import sensor_msgs.msg
 9
    import std_msgs.msg
10
11
    #
12
    # Class that contains the logic for our simple controller:
13
    #
        - listen to the "text_cmd" topic, accept
14
          "forward", "backward", "left", "right" and "stop"
    #
15
          as valid commands.
16
    #
        - publish a wheel command on the "wheel_cmd" topic
    #
17
18
    #
    class simple_text_controller:
19
        def __init__(self):
20
            # Subscribe to the "text_cmd" topic which has
21
             # "std_msgs/String" as message type.
22
            # It will call "self.command_callback" every
23
            # time a new message arrive
24
             self.command_sub = rospy.Subscriber("text_cmd",
25
                       std_msgs.msg.String, self.command_callback)
26
27
            # Publisher to the "wheel_cmd" topic which has
28
            # "sensor_msgs/JointState" as message type.
29
             self.effort_pub = rospy.Publisher("wheel_cmd",
30
                       sensor_msgs.msg.JointState, queue_size = 1)
31
32
        # Callback called everytime a new command message
33
        # is received
34
        def command_callback(self, command_msg):
35
             # Compute the effort depending on the received
36
            # command.
37
             if command_msg.data == "forward":
38
                 left effort = 0.5
39
```

```
right_effort = 0.5
40
             elif command_msg.data == "left":
41
                 left_effort = -0.5
42
                 right_effort = 0.5
43
             # TODO backward and right
44
             else:
45
                 # anything else
46
                 left_effort = 0.0
47
                 right_effort = 0.0
48
49
             # Fill the effort_msg with the command values
50
             effort_msg = sensor_msgs.msg.JointState()
51
             effort_msg.name = ["left", "right"]
52
             effort_msg.effort = [left_effort, right_effort]
53
             self.effort_pub.publish(effort_msg)
54
55
    if __name__ == '__main__':
56
        # Initialise the ROS sub system
57
        rospy.init_node('simple_text_controller', anonymous=False)
58
        # Create an instance of our controller
59
        ec = simple_text_controller()
60
        # Start listening to messages and loop forever
61
        rospy.spin()
62
```

Expand with command for turning right and going backward. You can run it with, in a new terminal:

```
start-tdde05
rosrun air_lab0 simple_text_controller.py __ns:=/husky0
```

To send command to it, in a new terminal:

start-tdde05
rostopic pub /husky0/text_cmd std_msgs/String "data: 'forward'"

5.8 Get started with GNU Screen

It can be annoying to have so many windows and terminals open, a solution is to use GNU Screen, which allows to start all the processes in a single terminal window.

The easiest way to use it is to create a screen configuration file:

1. In the air_lab0 project:

1 roscd air_lab0

```
2 mkdir screen
```

2. Then in that directory you can create a file called *labs*, with the following content (replace the ... accordingly):

```
# Configuration
 1
    deflogin on
 2
    autodetach on
 3
 4
    caption always
 5
 6
    bindkey ^w screen
 7
8
    bindkey ^p prev
    bindkey ^n next
9
10
    bindkey ^x quit
    bind q quit
11
    bindkey ^l windowlist
12
    bindkey ^e copy
13
14
    # Pre-defined tabs
15
16
   screen O
17
18
   title "roscore"
    stuff "start-tdde05\015"
19
    stuff "roscore\015"
20
21
22
    screen 1
   title "simple sim - no obstacles"
23
    stuff "start-tdde05; rosrun air_simple_sim simple_sim.py __ns:=/husky0\015"
24
25
26
    screen 2
   title "rviz"
27
    stuff "start-tdde05; rviz -d `rospack find air_lab0`/rviz/labs.rviz\015"
28
20
   screen 3
30
   title "simple_text_controller"
31
    stuff "start-tdde05; rosrun air_lab0 simple_...oller.py __ns:=/husky0\015"
32
```

- 3. Then to start it:
- 1 rosscreen air_lab0 labs

The following shortkeys can be used:

- ctrl+w to create a new command tab
- ctrl+p to navigate to the previous tab
- ctrl+n to navigate to the next tab

- ctrl+x to quit
- ctrl+l to show the list of tabs
- ctrl+e to scroll in the log, using up/down arrow (page up/down works as well), press ESC to stop scrolling

In the screen configuration file:

- *screen* ## indicates a new tab with number ##
- *title* indicates the name of the tab (as shown in the list of tabs)
- *stuff* indicates the command that is run in the tab, the \015 at the end indicates whether the command is ran when starting the screen or if you have to start it manually

5.9 Creating user interface with ROS

There is the possibility in ROS to build user interface with a program called *rqt*, which can be used for visualisation and for giving commands.

For this lab we are going to use it to send commands to the text controller and monitor some topics. In a terminal, you can launch:

1 rqt

This should show an empty application with a menu:

- in the menu, select Plugins, Robot topics, Message publisher and Topic Monitor
- in the Message Publisher panel, add the /husky0/text_cmd topic five times and then set the message in the expression column to forward, backward, left, right and stop respectively.
- Use the Topic Monitor panel to monitor the odometry and wheel command topics.

This should show an interface similar to figure 4. In the editor line you can type the name of the topic where you want to publish a velocity.

Add the rqt command to your screen file.

5.10 Demonstration

- Show your screen file
- Show in rviz how your robot respond to the forward, backward, left, right and stop command using rqt.

| 8 | | | | Default - rqt | | · • • |
|-------------------------|--|--|---|------------------------------------|-------------|--|
| File J | Plugins <u>R</u> unning Pe | srspectives Help | | | | |
| Mess | age Publisher | | | | | D01 - |
| е т | opic /husky0/text cm | nd | * Type std msgs/String | | | * Freq. 1 * Hz + - |
| nnic | w ha | na rat | expression | | | |
| - D | /husky0/text cmd str | d msgs/String 1.0 | 0 | | | |
| | data str | ring | 'forward' | | | |
| | /husky0/text_cmd sto data sti | d_msgs/String 1.0 ring | 0 'backward' | | | |
| | /husky0/text_cmd str | d_msgs/String 1.0 | 0 | | | |
| | data str | ring | 'left' | | | |
| | data sti | /_msgs/string_1.u ring | 'right' | | | |
| • | /husky0/text_cmd str | d_msgs/String 1.0 | 0 | | | |
| | data str | ring | 'stop' | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| opic | Monitor | | | | | |
| opic | Monitor | * Type | | Bandwidth | Hz | Dog · |
| opic | Monitor | * Type geometry m | sgs/PointStamped | Bandwidth | Hz | Value not monitored |
| opic | Monitor /clicked_point /husky0/odometry | * Type geometry_m nav_msgs/00 | sgs/PointStamped | Bandwidth 13.24KB/s | Hz 18.20 | Value D not monitored |
| opic opic | Monitor /clicked_point /husky0/odometry child_frame_id | Type geometry_m nav_msgs/00 string | sgs/PointStamped Jometry | Bandwidth 13.24KB/s | Hz 18.20 | Value not monitored huskyQhase_footprint* |
| opic opic | Monitor /clicked_point /husky0/odometry child_frame_id header | Type geometry_m nav_msgs/0c string std_msgs/He | sgs/PointStamped Sometry ader | Bandwidth 13.24KB/s | Hz 18.20 | Value D n 🕽 not monitored *busky@base_feetprint* |
| opic | Menitor /clicked_point /husky0/odometry child_frame_id header pose | Type geometry_m nav_msgs/Oc string std_msgs/He geometry_m | sgs/PointStamped sometry ader sgs/PoseWithCovariance | Bandwidth 13.24KB/s | Hz 18.20 | Value not monitored fuusky0base_footprint* |
| opic | Monitor /clicked_point /husky0/odometry child_frame_id header pose twist | Type geometry_m nav_msgs/Oc string std_msgs/He geometry_m geometry_m | sgs/PointStamped formetry sgs/PoseWithCovariance sgs/FwistWithCovariance | Bandwidth 13.24KB/s | Hz 18.20 | Value not monitored fundy obase_footprint |
| opic opic | Monitor /clicked_point /husky0/odometry child frame_id header pose twist covariance | Type geometry_m nav_msgs/Dc stid_msgs/He geometry_m float64(36) | sgs/PointStamped Jornetry ader gg/ToseWithCovariance gg/TosetwitWithCovariance | Bandwidth 13.24KB/s | Hz 18.20 | Value Image: Control of the control of th |
| opic opic V | Monitor /clicked_point /husky0/odometry child frame_id beader pose twist covariance * twist * angular | Type geometry_m nav_msg/Qo string std_msgs/He geometry_m float64[36] geometry_m geometry_m | sgs/PointStamped Jonnetry ader gg/PoseWithCovariance gg/TwistWithCovariance gg/TwistWithCovariance | Bandwidth 13.24KB/s | Hz 18.20 | Value not monitored Training/base_footprimt* (0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, |
| opic opic V | Monitor /clicked.point /fusky0jodometry child_frame_ld header pose twist covariance twist covariance twist * ungular * | Type geometry_m nav_msgs/Do string std_msgs/He geometry_m geometry_m float64(36) geometry_m float64 | sgs/PointStamped formetry sgs/PossWthCovariance gg/TwistWthCovariance gg/Twist sgs/Vector3 | Bandwidth 13.24KB/s | Hz 18.20 | Yahar Image: Control of the control of th |
| pic v | Monitor /ctlicked_point /husky0/odometry child frame_id header pose twist covariance * twist * angular × v | Type geometry_m nav_msgs/Dc string std_msgs/He geometry_m float64[36] geometry_m float64[36] | sgyPointStamped formetry ader ggyPoissWithCovariance ggyTwistt ggyTwistt ggyTwistt | Bandwidth 13.24KB/s | Hz 18.20 | Value rock monitored Yausy Qlasse, footprint* (00, 00, 00, 00, 00, 00, 00, 00, 00, 00, |
| pic V V | Menitor /clicked_point /husky0lodometry child frame_id header pose twist covariance * twist * angular x y z | Type geometry_m nav_msgs/0t string std_msgs/Ne geometry_m geometry_m float64 float64 float64 float64 | sgs/PointStamped formetry ader ggs/TwistWth/Covariance ggs/TwistWth/Covariance ggs/Twist ggs/Vector3 | Bandwidth 13.24KB/s | Hz 18.20 | Value Image: Control of the control of th |
| opic opic | Menitor /clicked_point /husky0/odometry /husky0/odometry bose twist covariance * twist * angular × y z * linear | Type geometry m nav_msgs/Oc string std_msgs/He geometry_m float64(36) geometry_m float64 float64 geometry_m float64 geometry_m | rgyPointStamped formetry ader gg/PossWithCovariance gg/TwistWithCovariance gg/Vector3 gg/Vector3 | Bandwidth 13.24KB/s | Hz 18.20 | Wates Image: Control of the control of th |
| opic opic | Monitor (clicked_point /nusky0idometry child frame_id beader covariance twist covariance twist vangular y z v linear x | Type geometry m nav_msgs/0c string std_msgs/lec std_msgs/lec geometry m geometry m fbat64/36 geometry m fbat64 | sguPointStamped Iometry ader ggrPosWHDCovariance ggrTwittWDCovariance ggrWettWDCovariance ggrWettor3 | Bandwidth 13.24KB/s | Hz 18.20 | Value Image: Control of the control of th |
| opic opic | Klonitor /clicked_point /huskyBodometry child_frame_id header covariance * twist * angular * angular y * linear x y | Type geometry m nav_msgs/0c string std_msgs/He geometry m float64(36) geometry m float64(36) float64 float64 float64 float64 | sgyPointStamped formtry ader ggrPosWthCovariance ggrWeittVtCovariance ggrWeittVtCovariance ggrWeittVtCovariance ggrWeittr3 | Bandwidth 13.24KB/s | Hz 18.20 | Value Image: Control of Co |
| ppic | Monitor /clicked point huidydiodometry huidydiodometry huidydiodometry pore pore twist vangular x y y v linear x y y y z v linear x y y z | Type geometry_m nav_msgs/0c string std_msgs/lbc string geometry_m float64 float64 float64 float64 float64 float64 | ngstPointStamped formstry gopPoseWHXCovariance ggpPoseWHXCovariance ggpTinsttWHXCovariance ggpTinstt ggrWetter3 | Bandwidth 13.24KB/s | Hz 18.20 | Water Image: Control of Co |
| apic apic | Monitor Atlicked point Anuskyloidometry child frame id header pose twist covariance v twist v angular v twist v angular v z linear x y z Muskyloidometry fullowetry fu | Type geometry_m nav_msg40c string std_msg4lbe geometry_m fbatts41361 geometry_m fbatts4 f | zgurðeintstamped formetry aðar ggyrðesttið Kovariance ggyrðestið Kovariance ggyrðestor ggyrðestor ggyrðestor ggyrðestor ggregenne | Bandwidth 13.24KB/s | Hz 18.20 | Value Image: Control of the control of th |
| opic opic × | Monitor /clicked_point /musky0idometry child_frame_id header pose to covariance * twist * angular x y linear x y x y musky0idheat_cmd /musky0idhead_cmd | Type geamatry, m wmsgr0x std, msgr0x std, msgr0x geometry, m float64(36) geometry, m float64 float64 float64 float64 float64 float64 float64 sensor, mags | egyPuintStamped formetry adar ggr/PastWHCcovariance ggr/PastWHCcovariance ggr/PastWHCcovariance ggr/Vector3 ggr/Vector3 ing joerState | Bandwidth 13.24KB/s 66.188/s | Hz 18.20 | Value Image: Control of the control of th |
| opic v | Monitor /clicked_point /nusky0iodometry child_frame_id header pose twist covariance * twist varyular y z linear x y musky0iotact_cmd profilear // header // he | Type geometry_m nav_msg40c string std, msg4Ne geometry_m fbatis4(36) geometry_m fbatis4(36) geometry_m fbatis4 fbatis4 fbatis4 fbatis4 fbatis4 fbatis4 fbatis4 fbatis4 std, msg5Ne | rgyffeintStamped fornetry awbr ggyffeatWthCovariance ggyffattWthCovariance ggyffeatWthCovariance ggyffeatState ggyffeatState awbr | Bandwidth 13.24KB/s 66.188/s | Hz 18.20 | Value Image: Control of Co |
| opic opic v | Monitor /clicked_point /musky0idometry child_frame_id header poso to covariance * twist * angular x y / / / / / / / / / / / / / | Type geometry, m regs/bit std_mgs/bit geometry_m float64(36) geometry_m float64 f | agyPointStamped formstry doer ggyPostWHCovariance ggyPostWHCovariance ggyPostWHCovariance ggyPostWHCovariance agyPostState ader | Bandwidth 13.24KB/s 66.188/s | Hz 18.20 | Value Image: Control of the start of the st |
| opic opic v | Monitor (clicked_point huskyt)todometry child_frame_id pose covariance v twist v angular v angular v angular v angular v twist v angular v an | Type geometry m nav msgshle geometry m geometry m floats4(36) geometry m floats4(36) geometry m floats4 floats4 | aggPointStamped Konstry GyrPastMtCovariance GyrPastMtCovariance gyrPastMtCovariance gyrPastMtCovariance gyrVector3 mg_Wector3 mg_werState ader | Bandwidth 13.24KB/s 66.188/s | Hz 18.20 | Water Image: Control of Co |
| opic opic V | Monitor /clicked_point /hashybiodometry /hashybiodometry /hashybiodometry /bashybiodometry / / / / / / / / / / / / / / / / / / / | Type geometry_m nav_msgs/0c string std_msgs/he geometry_m fbast64(36) geometry_m fbast64 | aguPointStamped Jonetry adar Martin M | Bandwidth 13.24KB/s 66.188/s | Hz 18.20 | Value Image: Control of the control of th |
| | Monitor (clicked point husky0iodometry child frame_id poor covariance covariance viation viati | Type geometry m raw msg2004 atting geometry m foat64(38) geometry m geometry m geometry m foat64 foat64 foat64 foat64 atd msg5St setror msg geometry m foat64 foaf64 foaf666 foaf666 foaf666 foaf666 foaf666 foaf666 foaf666 foaf666 foaf666 foaf666 foaf6666 foaf666 foaf666 foaf6666 foaf6666666 foaf6666666666 | aguPointStamped Iometry dor ggPoseWthCovariance ggPoseWthCovariance ggPovetta ggPovett | Bandwidth 13.24KB/s 66.188/s | Hz 18.20 | Value Image: Control of the control of th |
| Topic opic V V | Monitor Chicked_point Anasybiodometry hassybiodometry baader * otat * otat * ot | Type geometry m, nayayöv ati magikie ati magikie ati magikie geometry m, faatfal faatfal faatf | agg/PointStamped Smotty Smotty gg/PointStAmped Gg/PointStAte gg/PointStAte gg/PointStAte gg/PointStAte dig gg/PointStAte dig gg/PointStAte gg/ | Bandwidth 13.24KB/s 66.188/s | Hz 18.20 | Value Image: Control of Co |
| | Henitor Anistyskolonisty Anistyskolonisty Anistyskolonisty Anistyskolonisty to to to to to to to to to to | Type geometry, m reav, msg30C, starting geometry, m geometry, m geometry, m geometry, m geometry, m foat64 geometry, m foat64 geometry, m foat64 foaf64 <li< td=""><td>aggPointStamped Jonnetry def def ggrTest ggrTest ggrTest ag</td><td>Bandwidth 13.24KB/s 66.18B/s</td><td>Hz 18.20</td><td>Value Image: Control of the control of th</td></li<> | aggPointStamped Jonnetry def def ggrTest ggrTest ggrTest ag | Bandwidth 13.24KB/s 66.18B/s | Hz 18.20 | Value Image: Control of the control of th |

Figure 4: rqt with a Message Publisher and Topic Monitor panel.

References

- [2] Robot operating system (ros). http://www.ros.org/.