
XBOX 360

Contents

MEET TURTLEBOT

From the Clearpath site:

TurtleBot 2 is the world's most popular low cost, open source robot for education and research. This second generation personal robot is equipped with a powerful Kobuki robot base, a dual-core netbook, ASUS Xtion PRO Sensor (Might be a Kinect) and a gyroscope. All components have been seamlessly integrated to deliver an out-of-the-box development platform. Tap into the thriving open source ROS developer community and get started learning robotics on day one.

http://www.clearpathrobotics.com/turtlebot-2-open-source-robot/

Here are the specifications from the website:

TurtleBot

TM

PERSONAL ROBOTICS PLATFORM

SIDEFRONTTOP354 mm[14 in]420 mm[16.5 in]89mm[3.5 in]155mm[6 in]208mm[8 in]354 mm[14 in]317.5 mm[12.5 in]354 mm[14 in]

TECHNICAL SPECIFICATIONS

SIZE AND WEIGHT

EXTERNAL DIMENSIONS (L x W x H)	354 x 354 x 420 mm (14.0 x 14.0 x 16.5 in)
WEIGHT	6.3 kg (13.9 lb)
WHEELS (Diameter)	76 mm (3 in)
GROUND CLEARANCE	15 mm (0.6 in)

SPEED AND PERFORMANCE

MAX. PAYLOAD	5 kg (11 lb)
MAX. SPEED	0.65 m/s (2.1 ft/s)
MAX. ROTATIONAL SPEED	180°/S

BATTERY AND POWER SYSTEM

STANDARD BATTERY	2200 mAh Li-Ion
EXTENDED BATTERY	4400 mAh Li-Ion
USER POWER	5 V and 19V (1A), 12 V (1.5A), 12V (5A)

SENSORS

3D VISION SENSOR (ASUS Xtion PRO LIVE)*	Color Camera: 640px x 480px, 30 fps.	Depth Camera: 640px 480px, 30 fps
ENCORDERS	25700 cps	11.5 ticks/mm
RATE GYRO	110 deg/s Factory Calibrated	
AUXILIARY SENSORS	3x forward bump, 3x cliff, 2x wheel drop	

COMPUTER **(subject to change)**

MEMORY	4 GB
SCREEN	11.6in (1366x768)
PROCESSOR	Intel Core i3-4010U
GRAPHICS	Intel® HD Graphics
INTERNAL HARD DRIVE	500 GB
WIFI	802.11n
OPTICAL DRIVE	Not Applicable

***Check this – One of our TurtleBots has a Kinect.**

One good application of turtlebot

https://www.youtube.com/watch?v=roZ6DV8lNZc

This is a supporting material for the publication "Viewpoint based Moblie Robotic Exploration aiding Object Search in Indoor Environment" by Karthik Desingh, Akhil Nagariya, K Madhava Krishna at ICVGIP 2012

Introducing Yujin Robot's Kobuki

Yujin Robot Innovation Published on Aug 5, 2014

https://www.youtube.com/watch?v=t-KTHkbUwrU

Kobuki Motors, Gyro for Movement

http://kobuki.yujinrobot.com/documentation/

http://kobuki.yujinrobot.com/wiki/motor-details/

- Brushed DC Motor
- Motor Manufacturer: Standard Motor
- Part Name: RP385-ST-2060
- Rated Voltage: 12 V
- Rated Load: 5 mN·m
- No Load Current: 210 mA
- No Load Speed: 9960 rpm ± 15%
- Rated Load Current: 750 mA
- Rated Load Speed: 8800 rpm ± 15%
- Armature Resistance: 1.5506 Ω at 25°C
- Armature Inductance: 1.51 mH
- Torque Constant(Kt): 10.913 mN·m/A
- Velocity Constant(Kv): 830 rpm/V
- Stall Current: 6.1 A
- Stall Torque: 33 mN·m

Control Method

- Driven by voltage source(H-bridge)
- Controlled by Pulse-width modulation(PWM)

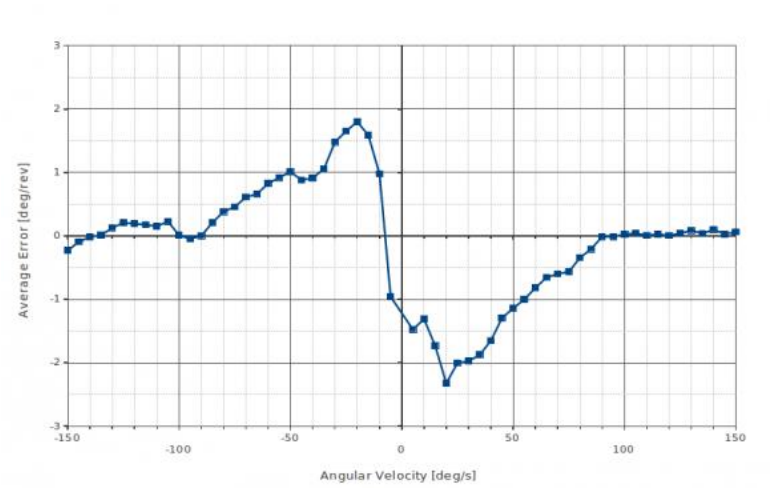
You want details – See my website Ceng 6533

- RobotControl MotorControl_1 MotorControl_2 Bode_Tracking
- StepperMotorDesign RobotPWMcontrol

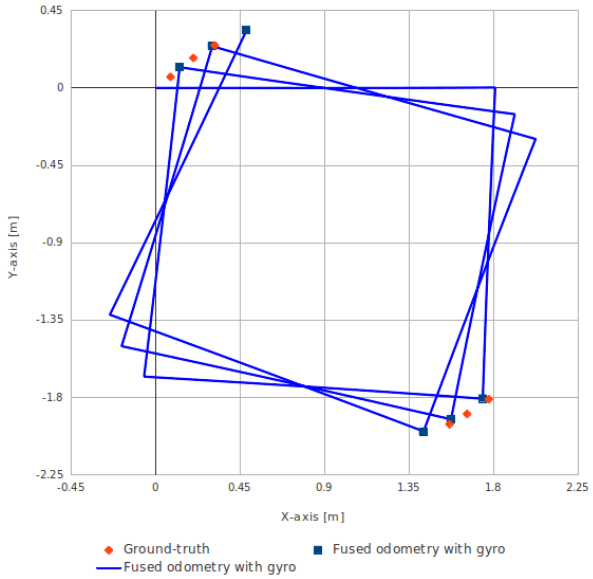
Need – some experience with Laplace Transforms

http://kobuki.yujinrobot.com/wiki/gyro-details/

- 3-Axis Digital Gyroscope
- Manufacturer : STMicroelectronics
- Part Name : L3G4200D
- Measurement Range: ±250 deg/s
- Yaw axis is factory calibrated within the range of ±20 deg/s to ±100 deg/s



This graph shows the average heading error per revolution of gyro, when robot rotates with a given velocity.



This graph shows the position error of fused odometry with gyro, when robot moves along a square path. Robot moved with 0.1 m/s on the line segment and rotated with 30 deg/s on the corner.

Number of turns of square path	Angular Error [deg]
0.5	0.47
1.5	1.99
2.5	3.18

This table shows the calculated angular error, when robot arrived at the diagonally opposite corner from the starting point(0.0, 0.0).

TURN ON TURTLEBOT – Cheat Sheet

1. POWER TO NETBOOK
2. LOG ON NETBOOK PASS: TB
3. POWER ON BASE (Button to right of base)
4. CONNECT NETBOOK TO BASE (lower left of base)
5. **CONNECT TO BUFFALO ROUTER**

ON WORKSTATION FOR KEYBOARD TELEOP

1. CONNECT TO BUFFALO ROUTER (System settings > Network)

2. Terminal 1: **$ . .turtlebot 2** (Set up HP210 Netbook as ROS MASTER)

#This makes TurtleBot the Master through the Buffalo Router 2/8/2016
export ROS_MASTER_URI=http://192.168.11.110:11311 # TurtleBot2 IP as MASTER
export ROS_IP=192.168.11.120 # Wireless IP on Workstation

3. Terminal 1 $ **ssh turtlebot-0877@192.168.11.110**

Enter Password turtlebot@192.168.11.1110's password: xxxxxxxx

4. $ **roslaunch turtlebot_bringup minimal.launch**

$. .turtlebot2

#This makes TurtleBot the Master through the Buffalo Router 2/08/2016 $ . .turtlebot2
export ROS_MASTER_URI=http://192.168.11.110:11311 # TurtleBot 2 IP as MASTER
export ROS_IP=192.168.11.120 # Wireless IP on Workstation uNCommented out 1/25/2016

Keyboard Control of TurtleBot

After Minimal Launch

Terminal 2
1$ . .turtlebot2
2. $ roslaunch turtlebot_teleop keyboard_teleop.launch

Control Your Turtlebot!

Moving around:
u i o
j k l
m , .
q/z : increase/decrease max speeds by 10%
w/x : increase/decrease only linear speed by 10%
e/c : increase/decrease only angular speed by 10%
space key, k : force stop
anything else : stop smoothly

CTRL-C to quit

Python Script to Control TurtleBot

TERMINAL 2
$ . .turtlebot2
$ python python_GoInCircle.py

TurtleBot Dashboard and Joystick Control

Dashboard
$ . .turtlebot2
$ ssh turtlebot-0877@192.168.11.110 password: turtlebot

turtlebot@turtlebot-0428:~$ **roslaunch turtlebot_bringup minimal.launch**

TERMINAL 2
tlharmanphd@D125-43873:~$ **. .turtlebot2**
tlharmanphd@D125-43873:~$ **roslaunch turtlebot_dashboard turtlebot_dashboard.launch**

Joystick xBOX 360
$ . .turtlebot2
$ ssh turtlebot-0877@192.168.11.110 password: turtlebot

turtlebot@turtlebot-0428:~$ **roslaunch turtlebot_bringup minimal.launch**

TERMINAL 2 JOYSTICK
tlharmanphd@D125-43873:~$ **. .turtlebot2**
tlharmanphd@D125-43873:~$ **roslaunch turtlebot_teleop xbox360_teleop.launch**

Hold the "dead-man" button

```
tlharmanphd@D125-43873:~$ . .turtlebot2
tlharmanphd@D125-43873:~$ rospack list turtle    <Tab>  <Tab>
turtle_actionlib                turtlebot_navigation
turtlebot_actions               turtlebot_panorama
turtlebot_bringup               turtlebot_rapps
turtlebot_calibration           turtlebot_rviz_launchers
turtlebot_capabilities          turtlebot_stage
turtlebot_dashboard             turtlebot_stdr
turtlebot_description           turtlebot_teleop
turtlebot_follower              turtlesim
turtlebot_gazebo                turtle_tf
turtlebot_interactive_markers   turtle_tf2
turtlebot_msgs
```

http://wiki.ros.org/Robots/TurtleBot

The packages were downloaded thus:

$ sudo apt-get install ros-indigo-turtlebot ros-indigo-turtlebot-apps ros-indigo-turtlebot-interactions ros-indigo-turtlebot-simulator ros-indigo-kobuki-ftdi ros-indigo-rocon-remocon ros-indigo-rocon-qt-library ros-indigo-ar-track-alvar-msgs

Nodes and Topics of Interest for TurtleBot

tlharmanphd@D125-43873:~$ **rosnode list**
/app_manager
/bumper2pointcloud
/capability_server
/capability_server_nodelet_manager
/cmd_vel_mux
/diagnostic_aggregator
/interactions
/master
/mobile_base
/mobile_base_nodelet_manager
/robot_state_publisher
/rosout
/turtlebot_laptop_battery
/zeroconf/zeroconf

http://wiki.ros.org/nodelet

Nodelets are designed to provide a way to run multiple algorithms on a single machine, in a single process, without incurring copy costs when passing messages intraprocess. roscpp has optimizations to do zero copy pointer passing between publish and subscribe calls within the same node. To do this nodelets allow dynamic loading of classes into the same node, however they provide simple separate namespaces such

that the nodelet acts like a seperate node, despite being in the same process. This has been extended further in that it is dynamically loadable at runtime using pluginlib.

I think this means that the nodelets are more efficient than nodes.

Here is an example from https://cse.sc.edu/~jokane/teaching/574/notes-turtlebot.pdf
CSCE574 – Robotics Spring 2014 – Notes on Turtlebot robots
He uses the multiplexing capability of TurtleBot control as an example.

From O'Kane: 3.2.2 Velocity Command Multiplexer
As you control the robot, you may have several different nodes that want to publish `cmd vel` messages. Which one should have control of the robot? If everyone publishes directly to `cmd vel`, then the robot will always try to execute the command in the most recent message it has received. This is obviously not a good solution if, for example, you'd like to take teleoperative control of the robot to override automatically generated commands sent by your software.

ROS provides a solution to this problem in the form of a `multiplexer node`. Each node that wants to send movement commands to the robot, instead of publishing directly to `cmd vel`, publishes messages on one of three different topics:
- `/cmd vel mux/input/navi`
- `/cmd vel mux/input/teleop`
- `/cmd vel mux/input/safety controller`

(Note that these specific topic names are determined by a configuration file; there's nothing stopping you from changing them, or adding others if you like.) When messages arrive on any of these topics, `cmdvel mux` decides which should take the highest priority, and forwards the corresponding messages to the `turtlebot node` via `cmd vel`.

Here's a launch file entry for this node:
```
<node
pkg="nodelet"
type="nodelet"
name="cmd_vel_mux"
args="standalone yocs_cmd_vel_mux/CmdVelMuxNodelet"
>
<param name="yaml_cfg_file" value="$(find turtlebot_bringup)/param/mux.yaml"/>
```

```
<remap from="cmd_vel_mux/input/teleop" to="turtlebot_teleop/cmd_vel"/>
<remap from="cmd_vel_mux/output" to="cmd_vel"/>
</node>
```

There are three noteworthy things here.
1. First and most noticeably, the `cmd vel mux` functionality is actually provided by a **nodelet** rather than a full-fledged node. The idea is, when nodes are very small and very simple, to reduce overhead by combining the functionality of several nodes into a single process. Usually, we would first start a nodelet manager, and then load one or more nodelets into that manager. In this case, however, there's only one nodelet, so we can launch it as a `standalone` node, without a separate manager.

2. Second, we must provide a configuration file, which defines the input and output topics for the multiplexer, along with a priority level and a timeout for each input topic. You might understand the role of `cmd vel mux` better if you examine this configuration file.

3. Finally, we use several `remap` entries to modify the topic names used by this node, to ensure that the correct connections are made with the other nodes.

3.2.3 Teleoperation Node

Once the `turtlebot node` and the `cmd vel mux` nodes are running, you can teleoperate the robot using a command like this on your workstation:

```
$ rosrun turtlebot_teleop turtlebot_teleop_key
```

See Jason O'Kanes website. He has a free book:

A Gentle Introduction to ROS

TOPICS for TurtleBot

tlharmanphd@D125-43873:~$ **rostopic list**

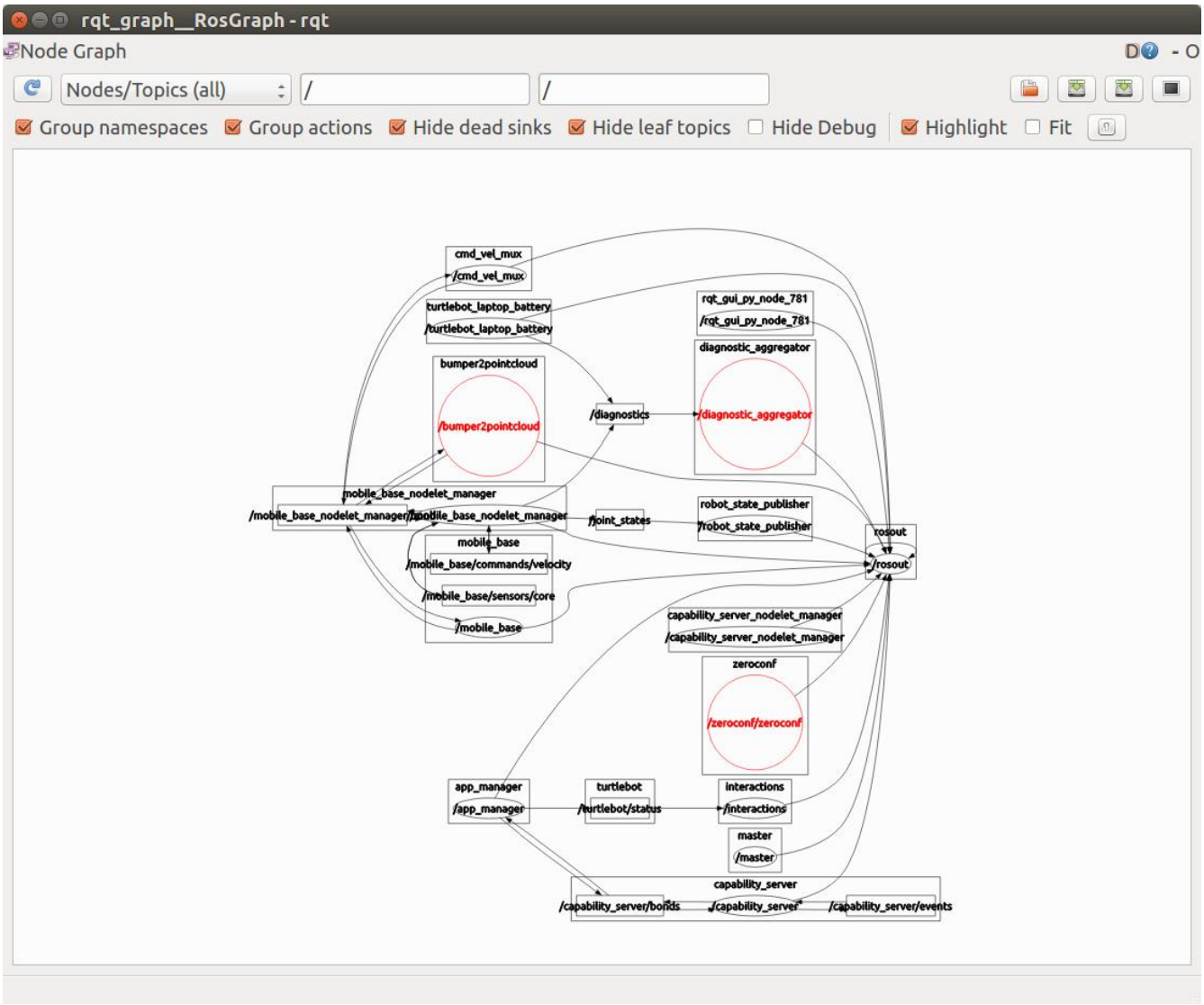
/capability_server/bonds
/capability_server/events
/cmd_vel_mux/active
/cmd_vel_mux/input/navi
/cmd_vel_mux/input/safety_controller
/cmd_vel_mux/input/teleop
/cmd_vel_mux/parameter_descriptions
/cmd_vel_mux/parameter_updates
.

/joint_states
/laptop_charge
/mobile_base/commands/controller_info
/mobile_base/commands/digital_output
/mobile_base/commands/external_power
/mobile_base/commands/led1
/mobile_base/commands/led2
/mobile_base/commands/motor_power
/mobile_base/commands/reset_odometry
/mobile_base/commands/sound
/mobile_base/commands/velocity
/mobile_base/controller_info
/mobile_base/debug/raw_control_command
/mobile_base/debug/raw_data_command
/mobile_base/debug/raw_data_stream
/mobile_base/events/bumper
/mobile_base/events/button
/mobile_base/events/cliff
/mobile_base/events/digital_input
/mobile_base/events/power_system
/mobile_base/events/robot_state
/mobile_base/events/wheel_drop
/mobile_base/sensors/bumper_pointcloud
/mobile_base/sensors/core

/mobile_base/sensors/dock_ir
/mobile_base/sensors/imu_data
/mobile_base/sensors/imu_data_raw
/mobile_base/version_info
/mobile_base_nodelet_manager/bond
/odom
.
/tf
/tf_static
/turtlebot/incompatible_rapp_list
/turtlebot/rapp_list
/turtlebot/status

RQT_GRAPH for TURTLEBOT – Nodes and Topics

tlharmanphd@D125-43873:~$ rqt_graph



Let's look at a useful node - **cmd_vel_mux**

tlharmanphd@D125-43873:~**$ rosnode info cmd_vel_mux**

--

```
Node [/cmd_vel_mux]
Publications:
 * /rosout [rosgraph_msgs/Log]
 * /mobile_base_nodelet_manager/bond [bond/Status]

Subscriptions:
 * /mobile_base_nodelet_manager/bond [bond/Status]

Services:
 * /cmd_vel_mux/set_logger_level
 * /cmd_vel_mux/get_loggers

contacting node http://192.168.11.110:58142/ ...
Pid: 3215
Connections:
 * topic: /rosout
    * to: /rosout
    * direction: outbound
    * transport: TCPROS
 * topic: /mobile_base_nodelet_manager/bond
    * to: /cmd_vel_mux
    * direction: outbound
    * transport: INTRAPROCESS
 * topic: /mobile_base_nodelet_manager/bond
                * to: /mobile_base_nodelet_manager
    * direction: outbound
    * transport: TCPROS
 * topic: /mobile_base_nodelet_manager/bond
    * to: /bumper2pointcloud
    * direction: outbound
    * transport: TCPROS
 * topic: /mobile_base_nodelet_manager/bond
    * to: /mobile_base
    * direction: outbound
    * transport: TCPROS
 * topic: /mobile_base_nodelet_manager/bond
    * to: /cmd_vel_mux (http://192.168.11.110:58142/)
    * direction: inbound
    * transport: INTRAPROCESS
 * topic: /mobile_base_nodelet_manager/bond
    * to: /mobile_base_nodelet_manager (http://192.168.11.110:58917/)
    * direction: inbound
                * transport: TCPROS
 * topic: /mobile_base_nodelet_manager/bond
    * to: /bumper2pointcloud (http://192.168.11.110:48128/)
    * direction: inbound
    * transport: TCPROS
 * topic: /mobile_base_nodelet_manager/bond
    * to: /mobile_base (http://192.168.11.110:41037/)
    * direction: inbound
    * transport: TCPROS
```

Kill A Node

You can close the window with the node /hello defined or kill the node with **rosnode kill <node>** command**.**

tlharmanphd@D125-43873:~$ **rosnode kill -h**

```
Usage: rosnode kill [node]...
Options:
  -h, --help  show this help message and exit
  -a, --all   kill all nodes
```

To check running process use **$ps –ef** to see all the processes running.

KEYBOARD CONTROL OF TURTLEBOT

tlharmanphd@D125-43873:~**$ . .turtlebot2 (For D125 Workstation)**

tlharmanphd@D125-43873:~**$ roslaunch turtlebot_teleop keyboard_teleop.launch**

```
... logging to /home/tlharmanphd/.ros/log/4d34d82a-d5c8-11e5-978f-8019347aeccf/roslaunch-D125-43873-
1655.log
Checking log directory for disk usage. This may take awhile.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://192.168.11.120:38615/

SUMMARY
========

PARAMETERS
 * /rosdistro: indigo
 * /rosversion: 1.11.16
 * /turtlebot_teleop_keyboard/scale_angular: 1.5
 * /turtlebot_teleop_keyboard/scale_linear: 0.5

NODES
  / turtlebot_teleop_keyboard (turtlebot_teleop/turtlebot_teleop_key)

ROS_MASTER_URI=http://192.168.11.110:11311

core service [/rosout] found
process[turtlebot_teleop_keyboard-1]: started with pid [1664]

Control Your Turtlebot!
---------------------------
Moving around:
   u    i    o
   j    k    l
   m    ,    .

q/z : increase/decrease max speeds by 10%
w/x : increase/decrease only linear speed by 10%
e/c : increase/decrease only angular speed by 10%
space key, k : force stop
anything else : stop smoothly

CTRL-C to quit

currently:          speed 0.2          turn
```

ROS NODES, TOPICS, MESSAGES AND SERVICES USING TURTLEBOT KEYBOARD

To clear the screen of the long list:
tlharmanphd@D125-43873:~$ **clear**

/turtlebot_teleop_keyboard NODE

tlharmanphd@D125-43873:~$ **rosnode info turtlebot_teleop_keyboard**

```
--------------------------------------------------------------------------------
Node [/turtlebot_teleop_keyboard]
Publications:
 * /rosout [rosgraph_msgs/Log]
 * /cmd_vel_mux/input/teleop [geometry_msgs/Twist]

Subscriptions: None

Services:
 * /turtlebot_teleop_keyboard/set_logger_level
 * /turtlebot_teleop_keyboard/get_loggers

contacting node http://192.168.11.120:50071/ ...
Pid: 1664
Connections:
 * topic: /cmd_vel_mux/input/teleop
    * to: /mobile_base_nodelet_manager
    * direction: outbound
            * transport: TCPROS
 * topic: /rosout
    * to: /rosout
    * direction: outbound
    * transport: TCPROS
```

The node **/turtlebot_teleop_keyboard** publishes two topics but subscribes to none because the keyboard outputs data. The services for the node are listed also. These refer to logging. The topic **/cmd_vel_mux/input/teleop** with the message type **geometry_msgs/Twist** will be studied in detail.

Messages

tlharmanphd@D125-43873:/$ **rosmsg help**

```
rosmsg is a command-line tool for displaying information about ROS Message types.

Commands:
        rosmsg show       Show message description
        rosmsg list       List all messages
        rosmsg md5        Display message md5sum
        rosmsg package    List messages in a package
        rosmsg packages   List packages that contain messages

Type rosmsg <command> -h for more detailed usage
```

If a topic publishes a message, we can determine the message type and read the message. There is a long list of messages for TurtleBot. Some important packages/MessageTypes are as follows:

Messages that involve the Kobuki base:

tlharmanphd@D125-43873:~$ **rosmsg list | grep kobuki**

```
kobuki_msgs/AutoDockingAction
kobuki_msgs/AutoDockingActionFeedback
kobuki_msgs/AutoDockingActionGoal
kobuki_msgs/AutoDockingActionResult
kobuki_msgs/AutoDockingFeedback
kobuki_msgs/AutoDockingGoal
kobuki_msgs/AutoDockingResult
kobuki_msgs/BumperEvent
kobuki_msgs/ButtonEvent
kobuki_msgs/CliffEvent
kobuki_msgs/ControllerInfo
kobuki_msgs/DigitalInputEvent
kobuki_msgs/DigitalOutput
kobuki_msgs/DockInfraRed
kobuki_msgs/ExternalPower
kobuki_msgs/KeyboardInput
kobuki_msgs/Led
kobuki_msgs/MotorPower
kobuki_msgs/PowerSystemEvent
kobuki_msgs/RobotStateEvent
kobuki_msgs/ScanAngle
kobuki_msgs/SensorState
kobuki_msgs/Sound
kobuki_msgs/VersionInfo
kobuki_msgs/WheelDropEvent
tlharmanphd@D125-43873:~$
```

Messages for position, orientation, etc. Commands and responses

tlharmanphd@D125-43873:~$ **rosmsg list | grep geometry**

```
geometry_msgs/Accel
geometry_msgs/AccelStamped
geometry_msgs/AccelWithCovariance
geometry_msgs/AccelWithCovarianceStamped
geometry_msgs/Inertia
geometry_msgs/InertiaStamped
geometry_msgs/Point
geometry_msgs/Point32
geometry_msgs/PointStamped
geometry_msgs/Polygon
geometry_msgs/PolygonStamped
geometry_msgs/Pose
geometry_msgs/Pose2D
geometry_msgs/PoseArray
geometry_msgs/PoseStamped
geometry_msgs/PoseWithCovariance
geometry_msgs/PoseWithCovarianceStamped
geometry_msgs/Quaternion
```

geometry_msgs/QuaternionStamped
geometry_msgs/Transform
geometry_msgs/TransformStamped
geometry_msgs/Twist
geometry_msgs/TwistStamped
geometry_msgs/TwistWithCovariance
geometry_msgs/TwistWithCovarianceStamped
geometry_msgs/Vector3
geometry_msgs/Vector3Stamped
geometry_msgs/Wrench
geometry_msgs/WrenchStamped

tlharmanphd@D125-43873:~$ **rosmsg list | grep turtle**

turtle_actionlib/ShapeAction
turtle_actionlib/ShapeActionFeedback
turtle_actionlib/ShapeActionGoal
turtle_actionlib/ShapeActionResult
turtle_actionlib/ShapeFeedback
turtle_actionlib/ShapeGoal
turtle_actionlib/ShapeResult
turtle_actionlib/Velocity
turtlebot_actions/FindFiducialAction
turtlebot_actions/FindFiducialActionFeedback
turtlebot_actions/FindFiducialActionGoal
turtlebot_actions/FindFiducialActionResult
turtlebot_actions/FindFiducialFeedback
turtlebot_actions/FindFiducialGoal
turtlebot_actions/FindFiducialResult
turtlebot_actions/TurtlebotMoveAction
turtlebot_actions/TurtlebotMoveActionFeedback
turtlebot_actions/TurtlebotMoveActionGoal
turtlebot_actions/TurtlebotMoveActionResult
turtlebot_actions/TurtlebotMoveFeedback
turtlebot_actions/TurtlebotMoveGoal
turtlebot_actions/TurtlebotMoveResult
turtlebot_calibration/ScanAngle
turtlebot_msgs/PanoramaImg
turtlesim/Color
turtlesim/Pose

We have another new item here – actions:

Summarizing from Quigley **Programming Robots with ROS, A Practical Introduction to the Robot Operating System** – Morgan Quigley, Brian Gerkey, William D. Smart

Chapter 5. Actions: ROS services are useful for synchronous request/response interactions—that is, for those cases where asynchronous ROS topics don't seem like the best fit. However, services aren't always the best fit, either, in particular when the request that's being made is more than a simple instruction of the form "get (or set) the value of X." While services are handy for simple get/set interactions like querying status and managing configuration, they don't work well when you need to

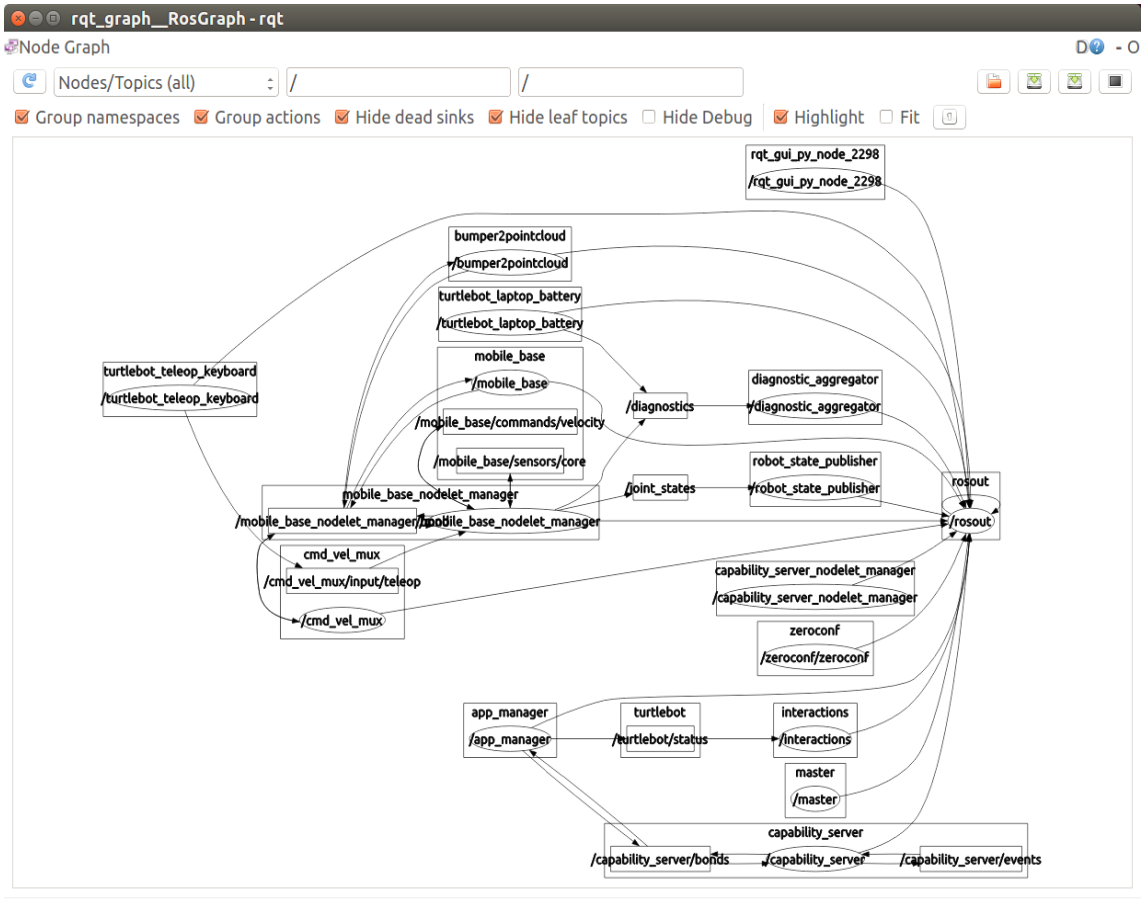
initiate a long-running task. For example, imagine commanding a robot to drive to some distant location; call it goto_position . The robot will require significant time (seconds, minutes, perhaps longer) to do so, with the exact amount of time impossible to know in advance, since obstacles may arise that result in a longer path.

ROS actions are the best way to implement interfaces to time-extended, goal-oriented behaviors like goto_position . While services are synchronous, actions are asynchronous. Similar to the request and response of a service, an action uses a goal to initiate a behavior and sends a result when the behavior is complete. But the action further uses feedback to provide updates on the behavior's progress toward the goal and also allows for goals to be canceled. Actions are themselves implemented using topics. An action is essentially a higher-level protocol that specifies how a set of topics (goal, result, feedback, etc.) should be used in combination.

The first step in creating a new action is to define the goal , result , and feedback message formats in an action definition file , which by convention has the suffix .action . The .action file format is similar to the .srv format used to define services, just with an additional field. And, as with services, each field within an .action file will become its own message.

RQT_GRAPH with /turtlebot_teleop_keyboard Node

tlharmanphd@D125-43873:~$ rqt_graph (Show /turtlebot_teleop_keyboard Node)



tlharmanphd@D125-43873:~$ **rostopic type /cmd_vel_mux/input/teleop**

geometry_msgs/Twist

The word “type” in this context is referring to the concept of a
data type . It’s important to understand message types because they determine the content of the messages. That is, the message type of a topic tells you what information is included in each message on that topic, and how that information is organized.

From the message *type* we can fine the format of the message. Be sure to note that Twist in the message type starts with a capital letter. http://wiki.ros.org/rostopic

tlharmanphd@D125-43873:~$ **rosmsg show geometry_msgs/Twist**

```
geometry_msgs/Vector3 linear
  float64 x
  float64 y
  float64 z
geometry_msgs/Vector3 angular
  float64 x
  float64 y
  float64 z
```

Rosmsg show geometry_msgs/Pose

tlharmanphd@D125-43873:~**$ rosmsg show geometry_msgs/Pose**

```
geometry_msgs/Point position
  float64 x
  float64 y
  float64 z
geometry_msgs/Quaternion orientation
  float64 x
  float64 y
  float64 z
  float64 w
```

http://wiki.ros.org/msg

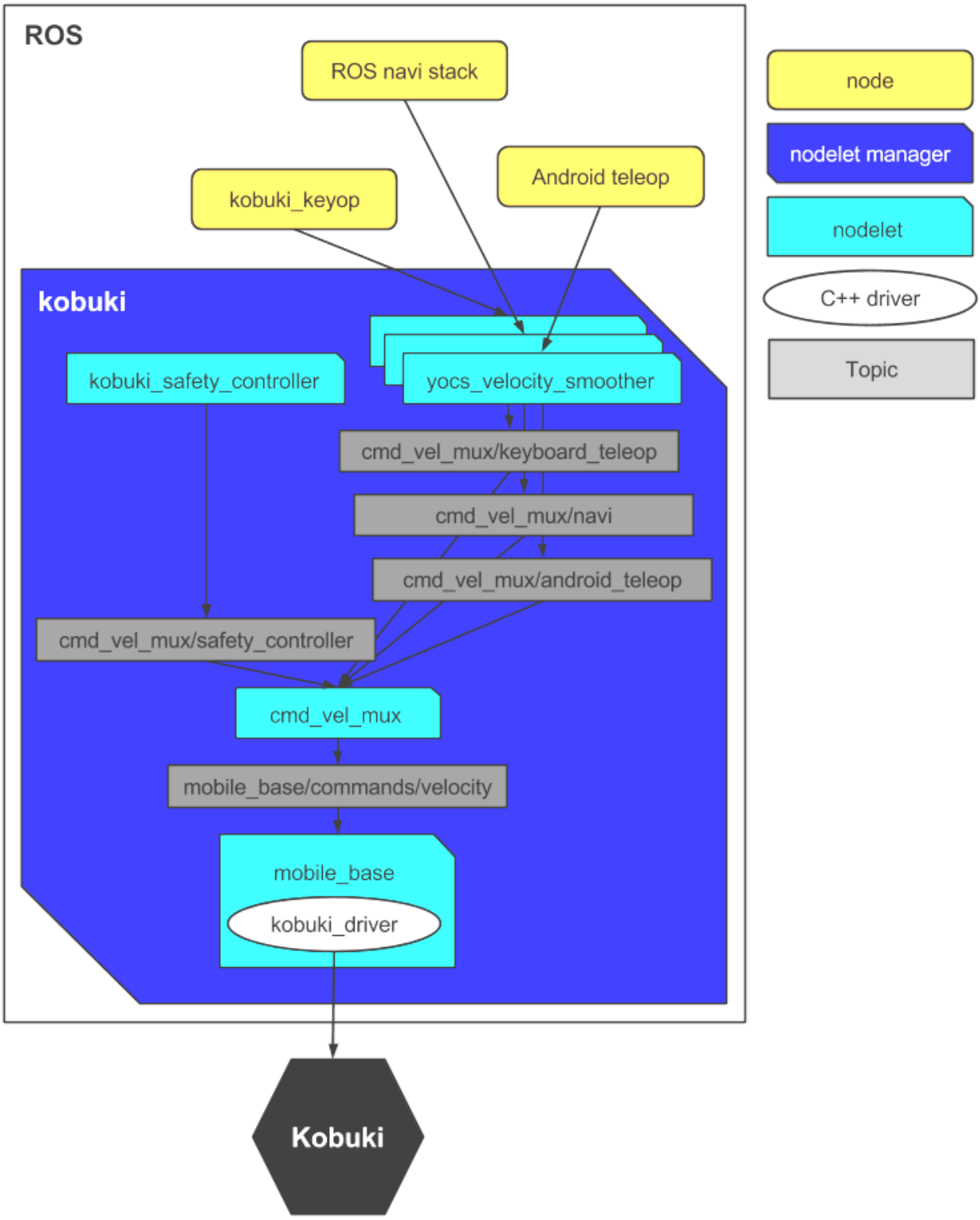
To understand the format of the message it is necessary to find the message type. The types include integers of 8, 16, 32, or 64 bits, floating point numbers, strings and other formats. The structure of the message type is:

<field> <constant>

where the field defines the type of data and the constant is the name.

Kobuki Control

http://wiki.ros.org/kobuki/Tutorials/Kobuki's%20Control%20System



Rosmsg show nav_msgs/Odometry

tlharmanphd@D125-43873:~$ **rosmsg show nav_msgs/Odometry**

```
std_msgs/Header header
  uint32 seq
  time stamp
  string frame_id
string child_frame_id
geometry_msgs/PoseWithCovariance pose
  geometry_msgs/Pose pose
    geometry_msgs/Point position
      float64 x
      float64 y
      float64 z
    geometry_msgs/Quaternion orientation
      float64 x
      float64 y
      float64 z
      float64 w
  float64[36] covariance
geometry_msgs/TwistWithCovariance twist
  geometry_msgs/Twist twist
    geometry_msgs/Vector3 linear
      float64 x
      float64 y
      float64 z
    geometry_msgs/Vector3 angular
      float64 x
      float64 y
      float64 z
  float64[36] covariance
```

http://answers.ros.org/question/12438/posestamped-and-pose-type-difference/

Pose is the x,y,z position and quaternion orientation of the robot, a `rosmsg show Pose` reveals:

```
[geometry_msgs/Pose]:

geometry_msgs/Point position

  float64 x

  float64 y

  float64 z

geometry_msgs/Quaternion orientation

  float64 x

  float64 y

  float64 z

  float64 w
```

While PoseStamped is simply a Pose message with the standard ROS header:

```
[geometry_msgs/PoseStamped]:

Header header

  uint32 seq

  time stamp

  string frame_id

geometry_msgs/Pose pose

  geometry_msgs/Point position

    float64 x

    float64 y

    float64 z

  geometry_msgs/Quaternion orientation

    float64 x

    float64 y

    float64 z

    float64 w
```

I think it depends on which stack you are using for which message is used, and I believe that PoseStamped is largely preferred because it includes the coordinate frame_id of the given Pose, as well as the time stamp that that Pose is valid.

On the other hand, if you don't need time information (say you are storing a time-independant Path), you could use an array of Poses, which would not need the additional header information.

Example:

```
header:
  seq: 26892
  stamp:
    secs: 1453674417
    nsecs: 187541901
  frame_id: odom
child_frame_id: base_footprint
pose:
  pose:
    position:
      x: -0.574884068509
      y: 1.18914280788
      z: 0.0
```

Rostopic echo /odom/pose/pose
From Quigley page 296 Just position and orientation

tlharmanphd@D125-43873:~$ **rostopic echo /odom/pose/pose**

```
position: (Arbitrary)
  x: 0.242611984228
  y: 0.00375067019721
  z: 0.0
orientation:
  x: 0.0
  y: 0.0
  z: 0.002967055375
  w: 0.999995598282
---
position:
  x: 0.242611984228
  y: 0.00375067019721
  z: 0.0
orientation:
  x: 0.0
  y: 0.0
  z: 0.002967055375
  w: 0.999995598282
```

RE-START MINIMAL LAUNCH SET X=0, Y=0. OUTPUT TO TEXT FILE

tlharmanphd@D125-43873:~$ **rostopic echo /odom/pose/pose >> tb_pose_test1.txt**

```
position:
  x: 0.0
  y: 0.0
  z: 0.0
orientation:
  x: 0.0
  y: 0.0
  z: 0.0
  w: 1.0
```

FINAL – MOVE IN STRAIGHT LINE ABOUT 1.2 METERS

```
position:
  x: 1.22930107254
  y: -0.0141608381814
  z: 0.0
orientation:
  x: 0.0
  y: 0.0
  z: -0.00741758129944
  w: 0.999972489365
```

GO BACK

```
pose:
 position:
    x: 0.0111620718896
    y: -0.053471637895
    z: 0.0
```

```
  orientation:
    x: 0.0
    y: 0.0
    z: 0.0312363117969
    w: 0.999512027354
  covariance: [0.1, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.1, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.7976931348623157e+308, 0.0, 0.0,
  0.0, 0.0, 0.0, 0.0, 0.0, 1.7976931348623157e+308, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.7976931348623157e+308, 0.0, 0.0, 0.0,
  0.0, 0.0, 0.0, 0.0, 0.05]
```

Rostopic echo /odom/pose

tlharmanphd@D125-43873:~$ **rostopic echo /odom/pose**

```
pose:
  position:
    x: 0.0111620718896
    y: -0.053471637895
    z: 0.0
  orientation:
    x: 0.0
    y: 0.0
    z: 0.0312363117969
    w: 0.999512027354
  covariance: [0.1, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.1, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.7976931348623157e+308, 0.0, 0.0,
  0.0, 0.0, 0.0, 0.0, 0.0, 1.7976931348623157e+308, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.7976931348623157e+308, 0.0, 0.0, 0.0,
  0.0, 0.0, 0.0, 0.0, 0.05]
  ---
```

tlharmanphd@D125-43873:~$ ^C

```
pose:
  position:
    x: 0.0111620718896
    y: -0.053471637895
    z: 0.0
  orientation:
    x: 0.0
    y: 0.0
    z: 0.0312363117969
    w: 0.999512027354
  covariance: [0.1, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.1, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.7976931348623157e+308, 0.0, 0.0,
  0.0, 0.0, 0.0, 0.0, 0.0, 1.7976931348623157e+308, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.7976931348623157e+308, 0.0, 0.0, 0.0,
  0.0, 0.0, 0.0, 0.0, 0.05]
  ---
```

tlharmanphd@D125-43873:~$ ^C

ROS SERVICES with TURTLEBOT

tlharmanphd@D125-43873:/$ **rosservice help**

Commands:

rosservice args	print service arguments
rosservice call	call the service with the provided arguments
rosservice find	find services by service type
rosservice info	print information about service
rosservice list	list active services
rosservice type	print service type
rosservice uri	print service ROSRPC uri

Type rosservice <command> -h for more detailed usage, e.g. 'rosservice call -h'

Use the $**rosservice list** command to see the services for the active node.

tlharmanphd@D125-43873:~$ **rosservice list**

/app_manager/get_loggers
/app_manager/set_logger_level
/bumper2pointcloud/get_loggers
/bumper2pointcloud/set_logger_level
/capability_server/establish_bond
/capability_server/free_capability
/capability_server/get_capability_spec
/capability_server/get_capability_specs
/capability_server/get_interfaces
/capability_server/get_loggers
/capability_server/get_nodelet_manager_name
/capability_server/get_providers
/capability_server/get_remappings
/capability_server/get_running_capabilities
/capability_server/get_semantic_interfaces
/capability_server/reload_capabilities
/capability_server/set_logger_level
/capability_server/start_capability
/capability_server/stop_capability
/capability_server/use_capability
/capability_server_nodelet_manager/get_loggers
/capability_server_nodelet_manager/list
/capability_server_nodelet_manager/load_nodelet
/capability_server_nodelet_manager/set_logger_level
/capability_server_nodelet_manager/unload_nodelet
/cmd_vel_mux/get_loggers
/cmd_vel_mux/set_logger_level
/cmd_vel_mux/set_parameters
/diagnostic_aggregator/get_loggers
/diagnostic_aggregator/set_logger_level
/interactions/get_interaction
/interactions/get_interactions
/interactions/get_loggers
/interactions/get_roles
/interactions/request_interaction
/interactions/set_interactions
/interactions/set_logger_level
/master/get_loggers

/master/set_logger_level
/mobile_base/get_loggers
/mobile_base/set_logger_level
/mobile_base_nodelet_manager/get_loggers
/mobile_base_nodelet_manager/list
/mobile_base_nodelet_manager/load_nodelet
/mobile_base_nodelet_manager/set_logger_level
/mobile_base_nodelet_manager/unload_nodelet
/robot_state_publisher/get_loggers
/robot_state_publisher/set_logger_level
/rosout/get_loggers
/rosout/set_logger_level
/rqt_gui_py_node_2298/get_loggers
/rqt_gui_py_node_2298/set_logger_level
/turtlebot/invite
/turtlebot/list_rapps
/turtlebot/platform_info
/turtlebot/start_rapp
/turtlebot/stop_rapp
/turtlebot_laptop_battery/get_loggers
/turtlebot_laptop_battery/set_logger_level
/turtlebot_teleop_keyboard/get_loggers
/turtlebot_teleop_keyboard/set_logger_level
/zeroconf/add_listener
/zeroconf/add_service
/zeroconf/list_discovered_services
/zeroconf/list_published_services
/zeroconf/remove_listener
/zeroconf/remove_service
/zeroconf/zeroconf/get_loggers
/zeroconf/zeroconf/set_logger_level
tlharmanphd@D125-43873:~$

---

tlharmanphd@D125-43873:/$ **rosservice help**

Commands:

rosservice args	print service arguments
rosservice call	call the service with the provided arguments
rosservice find	find services by service type
rosservice info	print information about service
rosservice list	list active services
rosservice type	print service type
rosservice uri	print service ROSRPC uri

Type rosservice <command> -h for more detailed usage, e.g. 'rosservice call -h'

Use the $**rosservice list** command to see the services for the active node.

PYTHON SCRIPT TO CONTROL TURTLEBOT

We will present a simple Python script to move the TurtleBot in this section. The basic approach to creating a script begins with a design. The design should detail the activity to be accomplished. For example, a script could command TurtleBot to move straight ahead, make several turns, and then stop. The next step is to determine the commands to TurtleBot to accomplish the tasks. Finally, a script is written and tested to see if TurtleBot responds in the expected way. The remote computer will execute the Python script and TurtleBot will move as directed if the script is correctly written.

In terms of the TurtleBot commands that will be used, we can summarize the process as follows:

- design the program outlining the activities for TurtleBot when the script executes
- determine the nodes, topics, and messages to be sent (published) or received (subscribed) from the TurtleBot during the activity
- study the ROS Python tutorials and examples to determine the way to write Python statements that send or receive messages between the remote computer and the TurtleBot.

There is a great deal of documentation describing ROS Python scripts. The statement structure is fixed for many operations. The site `http://wiki.ros.org/rospy` describes briefly `rospy` which is called the ROS client library for Python. The purpose is to allow statements written in Python language to interface with ROS topics and services.

The site `http://wiki.ros.org/rospy_tutorials` contains a list of tutorials. At the top of the tutorial page will be a choice of distributions of ROS and Indigo is chosen for our discussions. A specific tutorial that describes many of the Python statements that are used in a typical script can be found here:

`http://wiki.ros.org/ROS/Tutorials/WritingPublisherSubscriber(python)`

To find the nodes that are active after the `keyboard_teleop.launch` file was launched type:

```
$ rosnode list
```

/app_manager
/bumper2pointcloud
/capability_server
/capability_server_nodelet_manager
/cmd_vel_mux
/diagnostic_aggregator
/interactions
/master
/mobile_base
/mobile_base_nodelet_manager
/robot_state_publisher
/rosout
/turtlebot_laptop_battery
/turtlebot_teleop_keyboard
/zeroconf/zeroconf

The nodes are described in the Kobuki tutorial at

`http://wiki.ros.org/kobuki/Tutorials/Kobuki's%20Control%20System`

According to the site, the **mobile_base** node listens for commands such as velocity and publishes sensor information. The **cmd_vel_mux** serves to multiplex commands to assure that only one velocity command at a time is relayed through to the mobile base.

In a previous example we used the command `rostopic pub` to publish the linear and angular `geometry_msgs/Twist` data to move TurtleBot. The Python script that follows will accomplish essentially the same thing. The script will send Twist message on the topic `cmd_vel_mux/input/navi`.

A Python script will be created to move TurtleBot forward in a simple example. If you are not very familiar with Python, it may be best to study and execute the example script and then refer to the ROS tutorials. The procedure to create an executable script on the remote computer is as follows:

1. Write the script with the required format for a ROS Python script using an ordinary text editor.
2. Give the script a name in the format <name>.py and save the script.

We have called our script ControlTurtleBot.py and saved it in our home directory.

To make the scrip executable, execute the Ubuntu command:

```
$ chmod +x ControlTurtleBot.py
```

Make sure the TurtleBot is ready by running the minimal launch. Then in a new terminal window, type the command:

PYTHON AFTER MINIMAL LAUNCH

Terminal 1: $ . .turtlebot 2
$ ssh turtlebot-0877@192.168.11.110 turtlebot
$ roslaunch turtlebot_bringup minimal.launch

```
tlharmanphd@D125-43873:~$ pwd       /home/tlharmanphd
TERMINAL 2
     $ . .turtlebot2
     $ python python_GoInCircle.py
```

In our example, *Ctrl+C* is used to stop the TurtleBot. The comments in the script explain the statements. The tutorials listed previously give further details of Python scripts written using the ROS conventions.

```
#!/usr/bin/env python          python_GoInCircle

# A very basic TurtleBot script that moves TurtleBot InCircle indefinitely.
Press CTRL + C to stop.  To run:
# On TurtleBot:
# $ roslaunch turtlebot_bringup minimal.launch
# On work station:
# $ python  python_GoInCircle

import rospy
from geometry_msgs.msg import Twist

class GoInCircle():
    def __init__(self):
        # initiliaze
        rospy.init_node('GoInCircle', anonymous=False)

     # tell user how to stop TurtleBot
     rospy.loginfo("To stop TurtleBot CTRL + C")

        # What function to call when you ctrl + c
        rospy.on_shutdown(self.shutdown)

     # Create a publisher which can "talk" to TurtleBot and tell it to move
        # Tip: You may need to change cmd_vel_mux/input/navi to /cmd_vel if
you're not using TurtleBot2
        self.cmd_vel = rospy.Publisher('cmd_vel_mux/input/navi', Twist,
queue_size=10)

     #TurtleBot will stop if we don't keep telling it to move.  How often
should we tell it to move? 10 HZ
        r = rospy.Rate(10);

        # Twist is a datatype for velocity
        move_cmd = Twist()
     # let's go forward at 0.2 m/s
        move_cmd.linear.x = 0.2
     # let's turn at 1.0 radians/s  About 6 seconds to complete circle
     move_cmd.angular.z = 1.0

     # as long as you haven't ctrl + c keeping doing...
        while not rospy.is_shutdown():
          # publish the velocity
             self.cmd_vel.publish(move_cmd)
          # wait for 0.1 seconds (10 HZ) and publish again
             r.sleep()


    def shutdown(self):
        # stop turtlebot
        rospy.loginfo("Stop TurtleBot")
     # a default Twist has linear.x of 0 and angular.z of 0.  So it'll stop
TurtleBot
```

```
        self.cmd_vel.publish(Twist())
      # sleep just makes sure TurtleBot receives the stop command prior to
shutting down the script
        rospy.sleep(1)

if __name__ == '__main__':
    try:
        GoInCircle()
    except:
        rospy.loginfo("GoInCircle node terminated.")
```

See the new node **/GoInCircle**

tlharmanphd@D125-43873:~$ . .turtlebot2
tlharmanphd@D125-43873:~$ **rosnode list**

/GoInCircle
/app_manager
/bumper2pointcloud
/capability_server
/capability_server_nodelet_manager
/cmd_vel_mux
/diagnostic_aggregator
/interactions
/master
/mobile_base
/mobile_base_nodelet_manager
/robot_state_publisher
/rosout
/turtlebot_laptop_battery
/zeroconf/zeroconf
tlharmanphd@D125-43873:~$ rostopic list
/capability_server/bonds
/capability_server/events
/cmd_vel_mux/active
/cmd_vel_mux/input/navi
/cmd_vel_mux/input/safety_controller
/cmd_vel_mux/input/teleop
/cmd_vel_mux/parameter_descriptions
/cmd_vel_mux/parameter_updates
/diagnostics
/diagnostics_agg
/diagnostics_toplevel_state
/gateway/force_update
/gateway/gateway_info
/info
/interactions/interactive_clients
/interactions/pairing
/joint_states
/laptop_charge
/mobile_base/commands/controller_info
/mobile_base/commands/digital_output
/mobile_base/commands/external_power
/mobile_base/commands/led1
/mobile_base/commands/led2
/mobile_base/commands/motor_power
/mobile_base/commands/reset_odometry
/mobile_base/commands/sound
/mobile_base/commands/velocity
/mobile_base/controller_info
/mobile_base/debug/raw_control_command
/mobile_base/debug/raw_data_command
/mobile_base/debug/raw_data_stream
/mobile_base/events/bumper
/mobile_base/events/button
/mobile_base/events/cliff
/mobile_base/events/digital_input
/mobile_base/events/power_system
/mobile_base/events/robot_state
/mobile_base/events/wheel_drop
/mobile_base/sensors/bumper_pointcloud

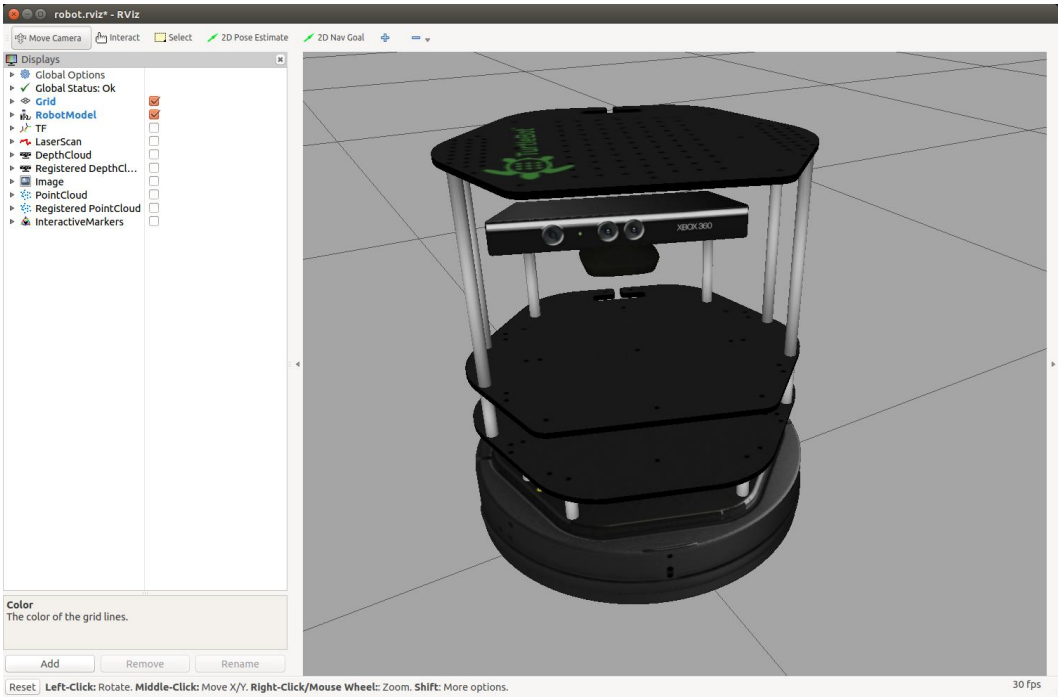
/mobile_base/sensors/core
/mobile_base/sensors/dock_ir
/mobile_base/sensors/imu_data
/mobile_base/sensors/imu_data_raw
/mobile_base/version_info
/mobile_base_nodelet_manager/bond
/odom
/rosout
/rosout_agg
/tf
/tf_static
/turtlebot/incompatible_rapp_list
/turtlebot/rapp_list
/turtlebot/status
/zeroconf/lost_connections
/zeroconf/new_connections
tlharmanphd@D125-43873:~$

Real TurtleBot's odometry display in rviz

The commands used in simulation can be used with the physical TurtleBot. After bringing up the real TurtleBot with minimal launch, start rviz on the remote computer:

```
$ roslaunch turtlebot_rviz_launchers view_robot.launch
```

TurtleBot will appear in rviz as this screenshot shows



Then, set up rviz with **odom** for **Fixed Frame** and **Add** > **By topic** > **Odometry.**

Run the command to move TurtleBot in a circle

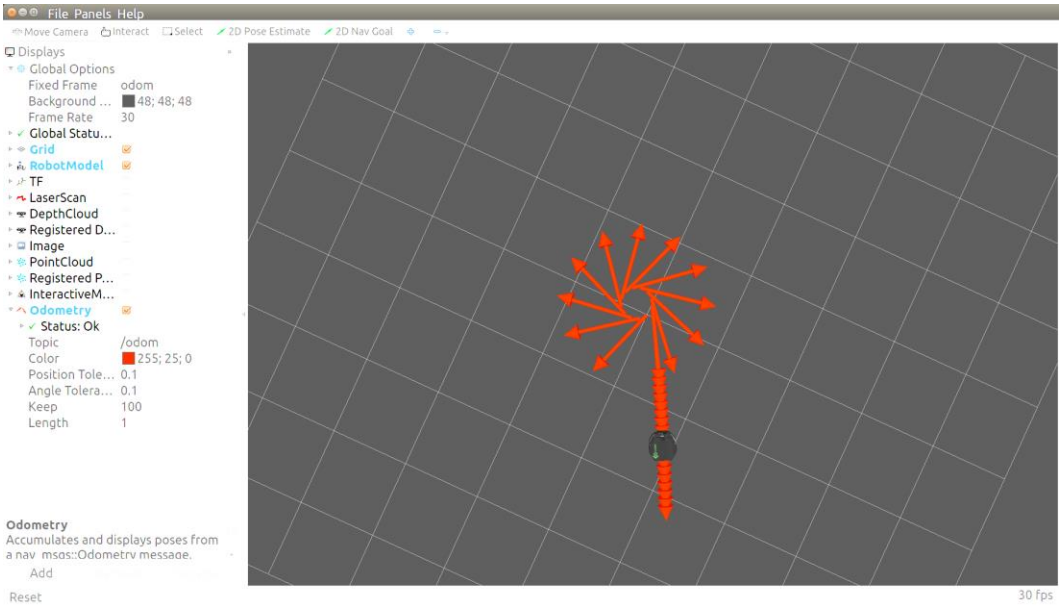
```
$ rostopic pub -r 10 /mobile_base/commands/velocity \geometry_msgs/Twist '{linear: {x: 0.1, y: 0, z: 0}, angular: {x: 0, y: 0, z: -0.5}}'
```

Stop TurtleBot by pressing *Ctrl+c* with focus on the window in which you executed the command to move the robot.

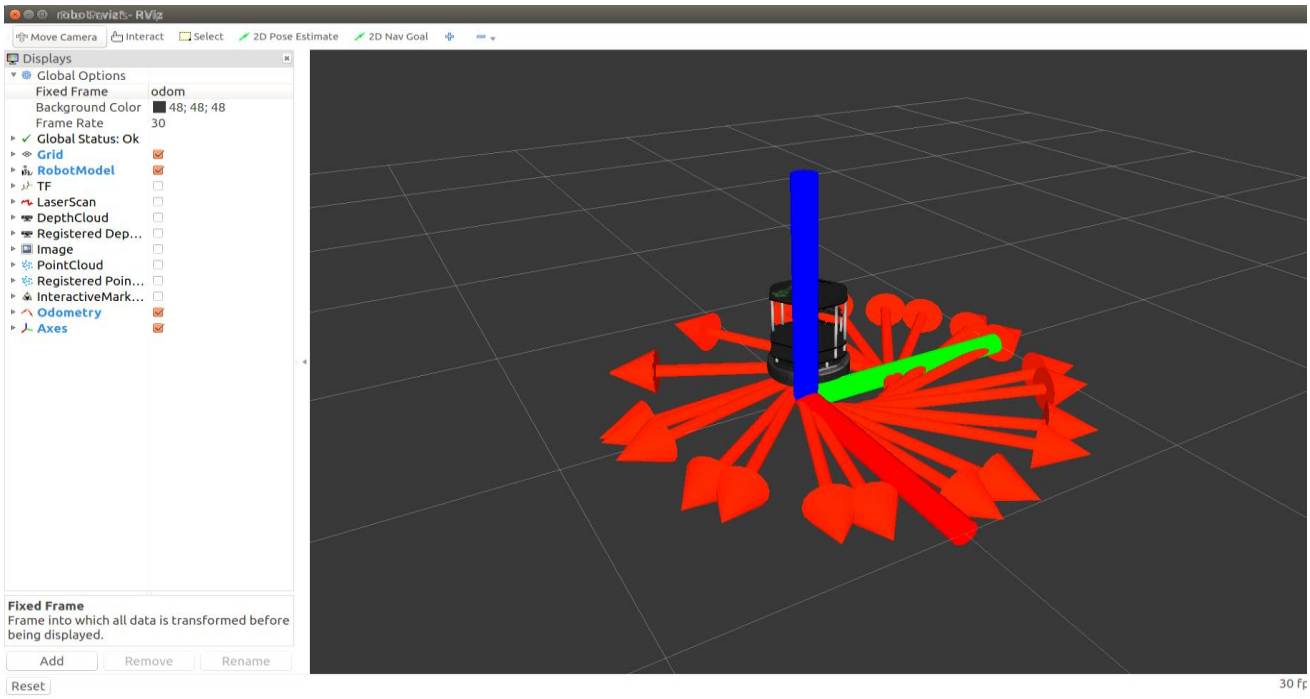
For the next screenshot, TurtleBot's turning was stopped with *Ctrl+C* and the Python script was executed that drives TurtleBot in a circle until *Ctrl+C* is pressed again.

The command is

```
$python python_GoInCircle.py
```



TurtleBot's path after Twist message and run of Python script



.

DASHBOARD OF TURTLEBOT

tlharmanphd@D125-43873:~$ **. .turtlebot2**

tlharmanphd@D125-43873:~$ **roslaunch turtlebot_dashboard turtlebot_dashboard.launch**

... logging to /home/tlharmanphd/.ros/log/b27145fe-d698-11e5-9b35-8019347aeccf/roslaunch-D125-43873-14885.log

Checking log directory for disk usage. This may take awhile.

Press Ctrl-C to interrupt

Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://192.168.11.120:49247/

SUMMARY

========

PARAMETERS

* /rosdistro: indigo

* /rosversion: 1.11.16

NODES

rqt_gui (rqt_gui/rqt_gui)

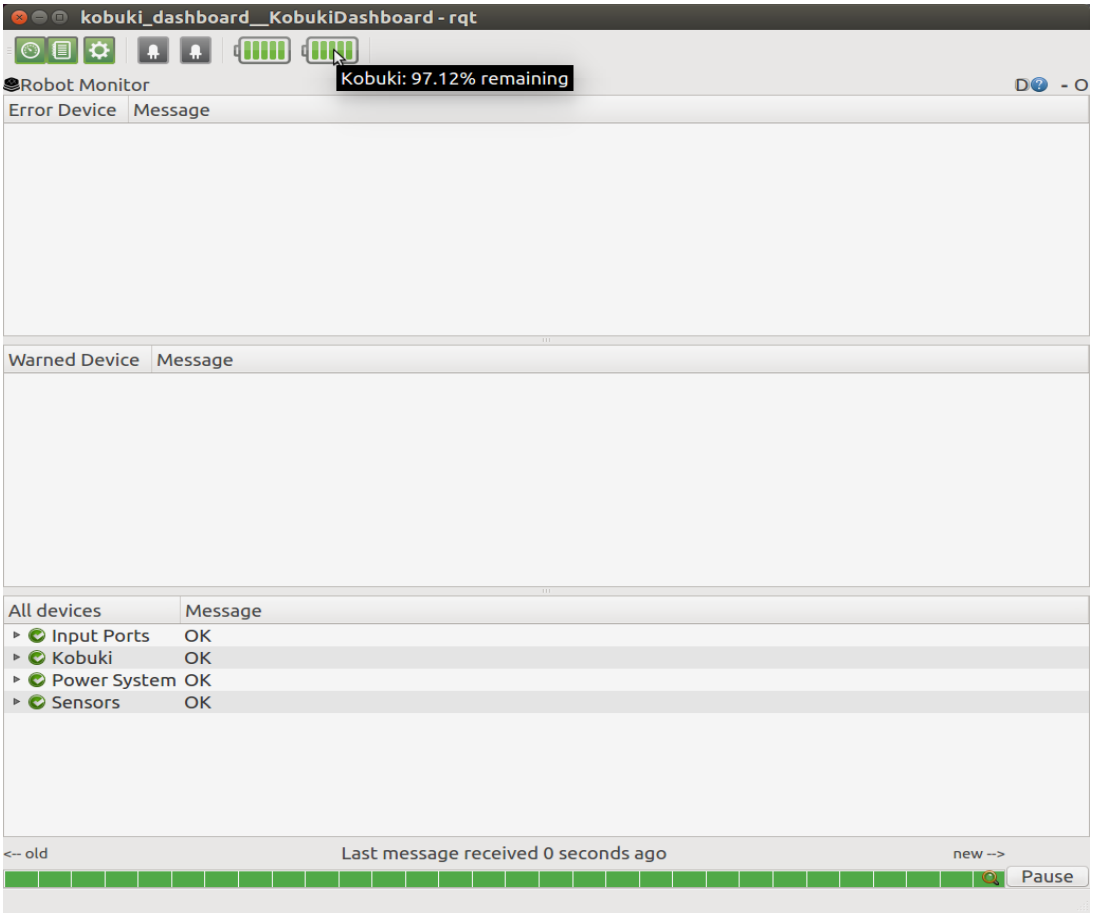
ROS_MASTER_URI=http://192.168.11.110:11311

c0re service [/rosout] found

process[rqt_gui-1]: started with pid [14894]

WARNING: Package "ompl" does not follow the version conventions. It should not contain leading zeros (unless the number is 0).

WARNING: Package "ompl" does not follow the version conventions. It should not contain leading zeros (unless the number is 0).



JOYSTICK xBOX 360

http://wiki.ros.org/turtlebot_teleop

The turtlebot_teleop package provides launch files for teleoperation with different input devices.

- For a keyboard teleoperation use:
  - roslaunch turtlebot_teleop keyboard_teleop.launch

For a ps3 joystick use:

  - roslaunch turtlebot_teleop ps3_teleop.launch

For a xbox360 joystick use:

  - roslaunch turtlebot_teleop xbox360_teleop.launch

http://wiki.ros.org/joy

$ . .turtlebot2
$ ssh turtlebot-0877@192.168.11.110 password: turtlebot

turtlebot@turtlebot-0428:~$ **roslaunch turtlebot_bringup minimal.launch**

TERMINAL 2 JOYSTICK
tlharmanphd@D125-43873:~$ **. .turtlebot2**
tlharmanphd@D125-43873:~$ **roslaunch turtlebot_teleop xbox360_teleop.launch**

tlharmanphd@D125-43873:~$ **. .turtlebot2**

tlharmanphd@D125-43873:~$ **roslaunch turtlebot_teleop xbox360_teleop.launch**

... logging to /home/tlharmanphd/.ros/log/b27145fe-d698-11e5-9b35-8019347aeccf/roslaunch-D125-43873-15282.log

Checking log directory for disk usage. This may take awhile.

Press Ctrl-C to interrupt

Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://192.168.11.120:35530/

SUMMARY

========

PARAMETERS

*** /rosdistro: indigo**

*** /rosversion: 1.11.16**

* /teleop_velocity_smoother/accel_lim_v: 1.0

* /teleop_velocity_smoother/accel_lim_w: 2.0

* /teleop_velocity_smoother/decel_factor: 1.5

* /teleop_velocity_smoother/frequency: 20.0

* /teleop_velocity_smoother/robot_feedback: 2

* /teleop_velocity_smoother/speed_lim_v: 0.8

* /teleop_velocity_smoother/speed_lim_w: 5.4

* /turtlebot_teleop_joystick/axis_angular: 0

* /turtlebot_teleop_joystick/axis_deadman: 4

* /turtlebot_teleop_joystick/axis_linear: 1

* /turtlebot_teleop_joystick/scale_angular: 1.5

* /turtlebot_teleop_joystick/scale_linear: 0.5

NODES

joystick (joy/joy_node)

teleop_velocity_smoother (nodelet/nodelet)

turtlebot_teleop_joystick (turtlebot_teleop/turtlebot_teleop_joy)

ROS_MASTER_URI=http://192.168.11.110:11311

core service [/rosout] found

process[teleop_velocity_smoother-1]: started with pid [15291]

process[turtlebot_teleop_joystick-2]: started with pid [15292]

process[joystick-3]: started with pid [15293]

PARAMETER SERVER

rosparam help

tlharmanphd@D125-43873:/$ **rosparam help**
rosparam is a command-line tool for getting, setting, and deleting parameters from the ROS Parameter Server.

Commands:

rosparam set	set parameter
rosparam get	get parameter
rosparam load	load parameters from file
rosparam dump	dump parameters to file
rosparam delete	delete parameter
rosparam list	list parameter names

tlharmanphd@D125-43873:~**$ rosparam get /rosversion**
'1.11.16

tlharmanphd@D125-43873:~**$ rosparam get /rosdistro**
'indigo

tlharmanphd@D125-43873:/$ **rosmsg help**
rosmsg is a command-line tool for displaying information about ROS Message types.

Commands:

rosmsg show	Show message description
rosmsg list	List all messages
rosmsg md5	Display message md5sum
rosmsg package	List messages in a package
rosmsg packages	List packages that contain messages

Type rosmsg <command> -h for more detailed usage

Hold left button and use left stick to move.

tlharmanphd@D125-43873:~**$ rosparam set /teleop_velocity_smoother/speed_lim_v 0.8**

tlharmanphd@D125-43873:~**$ rosparam get /teleop_velocity_smoother/speed_lim_v**

Ros Parameters after joy node

tlharmanphd@D125-43873:~$ rosparam list

/app_manager/auto_rapp_installation
/app_manager/auto_start_rapp
/app_manager/capability_server_name
/app_manager/local_remote_controllers_only
/app_manager/preferred
/app_manager/rapp_package_blacklist
/app_manager/rapp_package_whitelist
/app_manager/robot_icon
/app_manager/robot_name
/app_manager/robot_type
/app_manager/screen
/app_manager/simulation
/app_manager/use_gateway_uuids
/bumper2pointcloud/pointcloud_radius
/capability_server/blacklist
/capability_server/defaults/kobuki_capabilities/KobukiBringup
/capability_server/defaults/kobuki_capabilities/KobukiBumper
/capability_server/defaults/kobuki_capabilities/KobukiCliffDetection
/capability_server/defaults/kobuki_capabilities/KobukiLED
/capability_server/defaults/kobuki_capabilities/KobukiLED1
/capability_server/defaults/kobuki_capabilities/KobukiLED2
/capability_server/defaults/kobuki_capabilities/KobukiWheelDropDetection
/capability_server/defaults/std_capabilities/Diagnostics
/capability_server/defaults/std_capabilities/DifferentialMobileBase
/capability_server/defaults/std_capabilities/LaserSensor
/capability_server/defaults/std_capabilities/RGBDSensor
/capability_server/defaults/std_capabilities/RobotStatePublisher
/capability_server/defaults/turtlebot_capabilities/TurtleBotBringup
/capability_server/nodelet_manager_name
/capability_server/package_whitelist
/cmd_vel_mux/yaml_cfg_file
/description
/diagnostic_aggregator/analyzers/input_ports/contains
/diagnostic_aggregator/analyzers/input_ports/path
/diagnostic_aggregator/analyzers/input_ports/remove_prefix
/diagnostic_aggregator/analyzers/input_ports/timeout
/diagnostic_aggregator/analyzers/input_ports/type
/diagnostic_aggregator/analyzers/kobuki/contains
/diagnostic_aggregator/analyzers/kobuki/path
/diagnostic_aggregator/analyzers/kobuki/remove_prefix
/diagnostic_aggregator/analyzers/kobuki/timeout
/diagnostic_aggregator/analyzers/kobuki/type
/diagnostic_aggregator/analyzers/power/contains
/diagnostic_aggregator/analyzers/power/path
/diagnostic_aggregator/analyzers/power/remove_prefix
/diagnostic_aggregator/analyzers/power/timeout
/diagnostic_aggregator/analyzers/power/type
/diagnostic_aggregator/analyzers/sensors/contains
/diagnostic_aggregator/analyzers/sensors/path
/diagnostic_aggregator/analyzers/sensors/remove_prefix
/diagnostic_aggregator/analyzers/sensors/timeout

/diagnostic_aggregator/analyzers/sensors/type
/diagnostic_aggregator/base_path
/diagnostic_aggregator/pub_rate
/icon
/interactions/interactions
/interactions/pairing
/interactions/rosbridge_address
/interactions/rosbridge_port
/interactions/webserver_address
/mobile_base/base_frame
/mobile_base/battery_capacity
/mobile_base/battery_dangerous
/mobile_base/battery_low
/mobile_base/cmd_vel_timeout
/mobile_base/device_port
/mobile_base/odom_frame
/mobile_base/publish_tf
/mobile_base/use_imu_heading
/mobile_base/wheel_left_joint_name
/mobile_base/wheel_right_joint_name
/name
/robot/name
/robot/type
/robot_description
/robot_state_publisher/publish_frequency
/rocon/version
/rosdistro
/roslaunch/uris/host_192_168_11_110__50801
/roslaunch/uris/host_192_168_11_110__56061
/roslaunch/uris/host_192_168_11_120__35530
/roslaunch/uris/host_192_168_11_120__38309
/roslaunch/uris/host_192_168_11_120__49247
/rosversion
/run_id
/teleop_velocity_smoother/accel_lim_v
/teleop_velocity_smoother/accel_lim_w
/teleop_velocity_smoother/decel_factor
/teleop_velocity_smoother/frequency
/teleop_velocity_smoother/robot_feedback
/teleop_velocity_smoother/speed_lim_v
/teleop_velocity_smoother/speed_lim_w
/turtlebot_laptop_battery/acpi_path
/turtlebot_teleop_joystick/axis_angular
/turtlebot_teleop_joystick/axis_deadman
/turtlebot_teleop_joystick/axis_linear
/turtlebot_teleop_joystick/scale_angular
/turtlebot_teleop_joystick/scale_linear
/turtlebot_teleop_keyboard/scale_angular
/turtlebot_teleop_keyboard/scale_linear
/use_sim_time
/version
/zeroconf/zeroconf/services

tlharmanphd@D125-43873:~$ **rosparam get /robot/name**

turtlebot

tlharmanphd@D125-43873:~$ **rosparam get /turtlebot_teleop_joystick/axis_linear**
1
tlharmanphd@D125-43873:~$
tlharmanphd@D125-43873:~$ **rosparam get /teleop_velocity_smoother/frequency**
20.0
tlharmanphd@D125-43873:~$ **rosparam set /teleop_velocity_smoother/frequency 40.0**
tlharmanphd@D125-43873:~$ **rosparam get /teleop_velocity_smoother/frequency**
40.0

APPENDIX I REFERENCES NEEDS UPDATING - REFERENCES

The textbook *Learning ROS for Robotics Programming* by Aaron Martinez is useful. The examples are in C++.

A Gentle Introduction to ROS by Jason M. O'Kane is very readable and can be downloaded from the site: http://www.cse.sc.edu/~jokane/agitr/agitr-letter.pdf
The author's website is http://www.cse.sc.edu/~jokane/agitr/

These other ROS books might be helpful as referenced by O'Kane:

- ROS by Example by R. Patrick Goebel
- Learning ROS for Robotics Programming

  by Aaron Martinez and Enrique Fernandez. The examples are in C++.

Always be sure to check of any changes in the Ubuntu or ROS distribution. This *Turtlesim Guide* is written using Ubuntu 14.04 and ROS Indigo.

 If you are new to ROS - don't be impatient. There is a great deal to learn but the Turtlesim example shown here should make things easier.

The ROS official tutorials are at these WEB sites: **http://wiki.ros.org/turtlesim/Tutorials**

ROS Tutorials Helpful for the Examples to Follow:

- ROS/Tutorials/UnderstandingNodes
- ROS/Tutorials/UnderstandingTopics
- ROS/Tutorials/UnderstandingServicesParams

Programming Robots with ROS, A Practical Introduction to the Robot Operating System – Morgan Quigley, Brian Gerkey, William D. Smart 2015, O'Reilly

Other useful references are Listed in Appendix

GETTING STARTED WITH TURTLESIM
http://wiki.ros.org/turtlesim
***GENTLE INTRODUCTION* O'KANE CHAPTER 2**
http://www.cse.sc.edu/~jokane/agitr/agitr-letter-start.pdf
TUTORIALS USING TURTLESIM – A LIST
http://wiki.ros.org/turtlesim/Tutorials

ROS CONCEPTS

ROS has three levels of concepts: the Filesystem level, the Computation Graph level, and the Community level. These levels and concepts are summarized below and later sections go into each of these in greater detail.

The filesystem level concepts mainly cover ROS resources that you encounter on disk, such as packages, metapackages, manifests, repositories, messages, and services

The *Computation Graph* is the peer-to-peer network of ROS processes that are processing data together. The basic Computation Graph concepts of ROS are *nodes*, *Master*, *Parameter Server*, *messages*, *services*, *topics*, and *bags*, all of which provide data to the Graph in different ways.

The ROS Community Level concepts are ROS resources that enable separate communities to exchange software and knowledge. These resources include distributions, repositories, ROS wiki, ROS answers, and a Blog.

In addition to the three levels of concepts, ROS also defines two types of names -- Package Resource Names and Graph Resource Names -- which are discussed below.

http://wiki.ros.org/ROS/Concepts

ROSCORE

From the ROS tutorial http://wiki.ros.org/roscore

roscore is a collection of nodes and programs that are pre-requisites of a ROS-based system. You **must** have a roscore running in order for ROS nodes to communicate. It is launched using the `roscore` command.

ROS MASTER

The ROS Master provides naming and registration services to the rest of the nodes in the ROS system. It tracks publishers and subscribers to topics as well as services. The role of the Master is to enable individual ROS nodes to locate one another. Once these nodes have located each other they communicate with each other peer-to-peer.
http://wiki.ros.org/Master

Clearpath diagram of Master
http://www.clearpathrobotics.com/blog/how-to-guide-ros-101/

ROS NODES AND TURTLESIM

http://wiki.ros.org/ROS/Tutorials/UnderstandingNodes

ROS TOPICS AND TURTLESIM

http://wiki.ros.org/ROS/Tutorials/UnderstandingTopics

ROSSERVICE
rosservice contains the rosservice command-line tool for listing and querying ROS Services

http://wiki.ros.org/rosservice

ROSSERVICE AND ROS SERVICE PARAMETERS
This tutorial introduces ROS services, and parameters as well as using the rosservice and rosparam commandline tools.
http://wiki.ros.org/ROS/Tutorials/UnderstandingServicesParams

http://wiki.ros.org/Parameter%20Server

http://wiki.ros.org/rosparam

http://www.cse.sc.edu/~jokane/agitr/agitr-small-param.pdf (Chapter 7 of O'Kane)

ROSSERVICE AND ROS TELEPORT PARAMETER

Let's bring the turtle to a known starting point using absolute teleportation. Its inputs are [x y theta]. The origin [0 0 0] is offscreen so we will start with [1 1 0]. The turtle should be facing to the right (0*).

```
rosservice call /turtle1/teleport_absolute 1 1 0
```
https://sites.google.com/site/ubrobotics/ros-documentation

USING RQT_PLOT, RQT_CONSOLE AND ROSLAUNCH WITH TURTLESIM
http://wiki.ros.org/rqt_plot

This tutorial introduces ROS using rqt_console and rqt_logger_level for debugging and roslaunch for starting many nodes at once.
http://wiki.ros.org/ROS/Tutorials/UsingRqtconsoleRoslaunch

ROSBAG TURTLESIM EXAMPLE
This tutorial will teach you how to record data from a running ROS system into a .bag file, and then to play back the data to produce similar behavior in a running system.

Keywords: data, rosbag, record, play, info, bag

TURTLESIM EXAMPLE
http://wiki.ros.org/rosbag/Tutorials/Recording%20and%20playing%20back%20data/

DATA LOGGING USING ROSBAG
http://www.fer.unizg.hr/_download/repository/p08-rosbag.pdf

INTRODUCTION TO TF AND TURTLESIM
This tutorial will give you a good idea of what tf can do for you. It shows off some of the tf power in a multi-robot example using turtlesim. This also introduces using tf_echo, view_frames, rqt_tf_tree, and rviz.
http://wiki.ros.org/tf/Tutorials/Introduction%20to%20tf/

YAML Command LINE
Several ROS tools (rostopic, rosservice) use the YAML markup language on the command line. YAML was chosen as, in most cases, it offers a very simple, nearly markup-less solution to typing in typed parameters.

For a quick overview of YAML, please see YAML Overview.

http://wiki.ros.org/ROS/YAMLCommandLine

Appendix

A group of simple demos and examples to run on your TurtleBot to help you get started with ROS and TurtleBot.

https://github.com/turtlebot/turtlebot_apps

If you really want the details:

Graveyard/follower	Remove capabilities dependancy for follower, graveyard capabilities v…	2 years ago
software/pano	2.3.3	9 months ago
turtlebot_actions	2.3.3	9 months ago
turtlebot_apps	2.3.3	9 months ago
turtlebot_calibration	2.3.3	9 months ago
turtlebot_follower	2.3.3	9 months ago
turtlebot_navigation	Update CMakeLists.txt	9 months ago
turtlebot_panorama	2.3.3	9 months ago
turtlebot_rapps	Merge branch 'indigo' of https://github.com/turtlebot/turtlebot_apps …	8 months ago
.gitignore	adding gitignore	3 years ago
.hgignore	added android_map_nav app	5 years ago
README.md	Create README.md	3 years ago