ROS reference for HOBBIT

Topics

/head/move

This topic is used to move HOBBIT's head.
Message Type: std_msgs/String

Message	Description
center_center	look straight
up_center	look up
down_center	look down
center_right	look right
center_left	look left
up_right	look to upper right corner
up_left	look to upper left corner
down_right	look to lower right corner
down_left	look to lower left corner
littledown_center	look little down
to_grasp	look to grasp
to_turntable	look to turntable
search_table	look to table

/head/emo

This topic is used to control HOBBIT's eyes
Message type: std_msgs/String

Message	Description
HAPPY	look happy
VHAPPY	look very happy
LTIRED	look little tired
VTIRED	look very tired
CONCERNED	look concerned
SAD	look sad
WONDERING	wonder
NEUTRAL	look neutral
SLEEPING	sleep

/cmd_vel

This topic is used to move HOBBIT a certain distance
Message type: geometry_msgs/Twist

roscpp

roscpp is a C++ implementation of ROS. It provides a client library that enables C++ programmers to quickly interface with ROS Topics, Services, and Parameters.

Writing a Simple Publisher and Subscriber

This example covers how to write a publisher and subscriber node in C++.

```
#include "ros/ros.h"
#include "std_msgs/String.h"

#include <sstream>

/**
 * This tutorial demonstrates simple sending of messages over the ROS
     system.
 */
int main(int argc, char **argv)
{
  /**
   * The ros::init() function needs to see argc and argv so that it can
       perform
   * any ROS arguments and name remapping that were provided at the
       command line.
   * For programmatic remappings you can use a different version of init()
       which takes
   * remappings directly, but for most command-line programs, passing
       argc and argv is
   * the easiest way to do it. The third argument to init() is the name of
       the node.
   *
   * You must call one of the versions of ros::init() before using any other
   * part of the ROS system.
   */
```

```
ros::init(argc, argv, "talker");

/**
 * NodeHandle is the main access point to communications with the ROS
      system.
 * The first NodeHandle constructed will fully initialize this node, and
     the last
 * NodeHandle destructed will close down the node.
 */
ros::NodeHandle n;

/**
 * The advertise() function is how you tell ROS that you want to
 * publish on a given topic name. This invokes a call to the ROS
 * master node, which keeps a registry of who is publishing and who
 * is subscribing. After this advertise() call is made, the master
 * node will notify anyone who is trying to subscribe to this topic name,
 * and they will in turn negotiate a peer-to-peer connection with this
 * node. advertise() returns a Publisher object which allows you to
 * publish messages on that topic through a call to publish(). Once
 * all copies of the returned Publisher object are destroyed, the topic
 * will be automatically unadvertised.
 *
 * The second parameter to advertise() is the size of the message queue
 * used for publishing messages. If messages are published more quickly
 * than we can send them, the number here specifies how many messages
     to
 * buffer up before throwing some away.
 */
ros::Publisher chatter_pub = n.advertise<std_msgs::String>("chatter",
    1000);

ros::Rate loop_rate(10);

/**
 * A count of how many messages we have sent. This is used to create
 * a unique string for each message.
 */
int count = 0;
while (ros::ok())
{
```

```
    /**
     * This is a message object. You stuff it with data, and then publish it.
     */
    std_msgs::String msg;

    std::stringstream ss;
    ss << "hello world " << count;
    msg.data = ss.str();

    ROS_INFO("%s", msg.data.c_str());

    /**
     * The publish() function is how you send messages. The parameter
     * is the message object. The type of this object must agree with the
         type
     * given as a template parameter to the advertise<>() call, as was
         done
     * in the constructor above.
     */
    chatter_pub.publish(msg);

    ros::spinOnce();

    loop_rate.sleep();
    ++count;
  }


  return 0;
}
```