

# Introduction to Cognitive Robotics

David Vernon

[www.vernon.eu/cognitive\\_robotics](http://www.vernon.eu/cognitive_robotics)

# Lecture 5

[www.vernon.eu/cognitive\\_robotics/CR\\_05.pdf](http://www.vernon.eu/cognitive_robotics/CR_05.pdf)

## Mobile Robots: ROS

- Introduction to ROS (Robot Operating System)

(Rhymes with “gloss”)

- Introduction to writing ROS software
  - Using the ROS Turtlebot simulator `Turtlesim`
  - Here in C++
  - Later in the course in Lisp with CRAM



Based mainly on J. M. O’Kane, *A Gentle Introduction to ROS*, 2014.  
<https://cse.sc.edu/~jokane/agitr/>

# Writing ROS Software

- Creating a ROS workspace and a ROS package
- Writing ROS programs

1. Example program: Hello World!

2. Example program to publish messages

Send velocity messages on `/turtle1/cmd_vel`

3. Example program to subscribe to messages

Receive pose messages on `/turtle1/pose`

# Writing ROS Software

- Writing ROS programs
  4. Example program to use services

```
/reset  
/clear  
/turtle1/set_pen  
/turtle1/teleport_absolute
```

# Creating a ROS workspace


- We will then create our packages in that workspace
- We will refer to it as the **workspace directory**
- We will create it in the home directory
- We can name the directory anything we wish but, to be consistent with the CRAM examples later in the course, we will name it **~/workspace/ros**

# Creating a ROS workspace

## Create a ROS workspace

- We also need to create a **src** sub-directory in the workspace directory
- We can do all this in one step:

```
~$ mkdir -p ~/workspace/ros/src
```



The -p causes the creation of three directories:  
/workspace  
/workspace/ros  
/workspace/ros/src

# Creating a ROS package

Create a package for the example programs

```
~$ cd ~/workspace/ros/src
```

```
~/workspace/ros/src$ catkin_create_pkg agitr roscpp
```

The second argument identifies a dependency on roscpp  
Other dependencies can be specified by additional arguments

We name the package **agitr** because these examples are based on [A Gentle Introduction to ROS](#) by J. M. O'Kane

This creates

- a directory **agitr** to hold the package, containing the following files
- **package.xml** a configuration file containing the manifest we discussed earlier
- **CMakeLists.txt** a script for CMake, an industrial-strength build system used by ROS
- and the following sub-directories **src** and **include**

# Creating a ROS package

Make sure you are in the **agitr** sub-directory

```
~/workspace/ros/src$ cd agitr
```

```
~/workspace/ros/src/agitr$
```



# Creating a ROS package

## Edit `package.xml`

- Use your preferred editor, e.g. vi or emacs
  - Later in the course, we will be using emacs extensively with CRRAM
  - Perhaps this is a good time to become familiar with it
  - There is an introduction to Emacs in Lecture 21
- Non-essential edits:
  - Update the description
  - Update the name and email of the maintainer

# Example Program: Hello World!

Edit **CMakeLists.txt**

Add the following lines at the end of the file

```
add_executable(hello src/hello.cpp)
target_link_libraries(hello ${catkin_LIBRARIES})
```

# Example Program: Hello World!

- Move to the **agitr/src** sub-directory

```
~/workspace/ros/src$ cd agitr
~/workspace/ros/src/agitr$ cd src
~/workspace/ros/src/agitr/src$
```


- Edit **hello.cpp** and insert the following code

```
/* This is a ROS version of the standard "Hello , World" program */

#include <ros/ros.h> // This header defines the standard ROS classes

int main(int argc, char **argv) {
    ros::init(argc, argv, "hello_world"); // Initialize the ROS system
    ros::NodeHandle nh; // Register this program as a ROS node
    ROS_INFO_STREAM("Hello World!"); // Send some output as a log message
}
```

This is the name of your node



# Example Program: Hello World!

Build the workspace to compile the program

- Because catkin builds all of the packages in the workspace directory, first make sure you are in the workspace directory

```
~/workspace/ros/src/agitr/src$ cd ..
```

```
~/workspace/ros/src/agitr$ cd ..
```

```
~/workspace/ros/src$ cd ..
```

```
~/workspace/ros$
```

or, simply

```
~/workspace/ros/src/agitr$ cd ~/workspace/ros
```

```
~/workspace/ros$
```

# Example Program: Hello World!

Build the workspace to compile the program

- Run `catkin_make`

```
~/workspace/ros$ catkin_make
```

- Add the workspace to your ROS environment by sourcing the generated setup file

```
~/workspace/ros$ source devel/setup.bash
```

- Add the setup to your `.bashrc` file so that you don't have to do this every time you open a new terminal

```
~/workspace/ros$ echo "source $HOME/workspace/ros/devel/setup.bash" >> ~/.bashrc
```

# Example Program: Hello World!

Run the ROS master

If you have not already done so, open a terminal and enter

```
~$ roscore
```

Now, use **roslaunch** to execute the program

Open a second terminal and enter

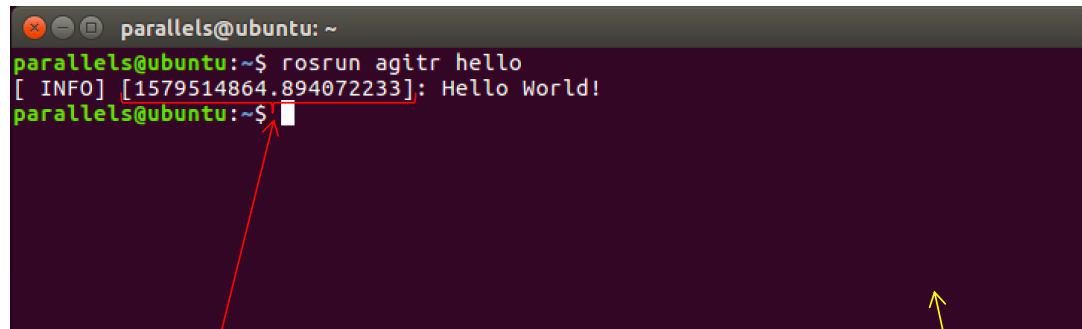
```
~/workspace/ros$ roslaunch agitr hello
```

This is the name of your package

This is the name of your program

# Example Program: Hello World!

If everything works correctly, you should see a message printed to the terminal

A terminal window with a dark purple background. The title bar shows 'parallels@ubuntu: ~'. The prompt is 'parallels@ubuntu:~\$'. The command 'roslaunch agitr hello' has been entered. The output is '[ INFO] [1579514864.894072233]: Hello World!'. The prompt is now 'parallels@ubuntu:~\$' with a white cursor. A red arrow points from the text below to the timestamp '1579514864.894072233'. A yellow arrow points from the text below to the end of the terminal output.

```
parallels@ubuntu: ~
parallels@ubuntu:~$ roslaunch agitr hello
[ INFO] [1579514864.894072233]: Hello World!
parallels@ubuntu:~$
```

This is the time, measured in seconds since January 1, 1970

# Example Program to Publish Messages

Send velocity messages on `/turtle1/cmd_vel`

Make sure you are in the `agitr` sub-directory

```
~/workspace/ros/src$ cd ~/workspace/ros/src/agitr
```

Or, better

```
~/workspace/ros/src$ roscd agitr
```



# Example Program to Publish Messages

Send velocity messages on /turtle1/cmd\_vel

Edit **CMakeLists.txt**

Add the following lines at the end of the file

```
add_executable(pubvel src/pubvel.cpp)
target_link_libraries(pubvel ${catkin_LIBRARIES})
```

# Example Program to Publish Messages

Send velocity messages on `/turtle1/cmd_vel`

Move to the `agitr/src` sub-directory

```
~/workspace/ros/src/agitr$ cd src
```

```
~/workspace/ros/src/agitr/src$
```

# Example Program to Publish Messages

Send velocity messages on /turtle1/cmd\_vel

Edit **pubvel.cpp** and insert the following code

```
/* This program publishes randomly-generated velocity messages for turtlesim */

#include <ros/ros.h>
#include <geometry_msgs/Twist.h> // For geometry_msgs::Twist
#include <stdlib.h>             // For rand() and RAND_MAX

int main(int argc , char **argv) {
    ros::init(argc, argv, "publish_velocity"); // Initialize the ROS system
    ros::NodeHandle nh;                       // Become a node
    ros::Publisher pub = nh.advertise<geometry_msgs::Twist>("turtle1/cmd_vel", 1000);
    srand(time(0)); // Seed the random number generator
    ros::Rate rate(2); // Loop at 2Hz until the node is shut down

    while(ros::ok()) {

        geometry_msgs::Twist msg; // Create the message
        msg.linear.x = double(rand())/double(RAND_MAX); // fill in the fields
        msg.angular.z = 2*3.14159*double(rand())/double(RAND_MAX)-1; // other fields default to 0
        pub.publish(msg); // Publish the message

        /* Send a message to roscout with the details */
        ROS_INFO_STREAM("Sending random velocity command:" << " linear =" << msg.linear.x << " angular =" << msg.angular.z);
        rate.sleep(); // Wait until it's time for another iteration
    }
}
```

# Example Program to Publish Messages

Send velocity messages on /turtle1/cmd\_vel

Edit **pubvel.cpp** and insert the following code

```
/* This program publishes randomly-generated velocity messages for turtlesim */

#include <ros/ros.h>
#include <geometry_msgs/Twist.h> // For geometry_msgs::Twist
#include <stdlib.h>             // For rand() and RAND_MAX

int main(int argc , char **argv) {
    ros::init(argc, argv, "publish_velocity"); // Initialize the ROS system
    ros::NodeHandle nh;                       // Become a node
    ros::Publisher pub = nh.advertise<geometry_msgs::Twist>("turtle1/cmd_vel", 1000);
    srand(time(0)); // Seed the random number generator
    ros::Rate rate(2); // Loop at 2Hz until the node is shut down

    while(ros::ok()) {

        geometry_msgs::Twist msg; // Create the message
        msg.linear.x = double(rand())/double(RAND_MAX); // fill in the fields
        msg.angular.z = 2*3.14159*double(rand())/double(RAND_MAX)-1; // other fields default to 0
        pub.publish(msg); // Publish the message

        /* Send a message to roscout with the details */
        ROS_INFO_STREAM("Sending random velocity command:" << " linear =" << msg.linear.x << " angular =" << msg.angular.z);
        rate.sleep(); // Wait until it's time for another iteration
    }
}
```

Twist type in the `geometry_msgs` package is used to instantiate a message

# Example Program to Publish Messages

Send velocity messages on /turtle1/cmd\_vel

Edit `pubvel.cpp` and insert the following code

```
/* This program publishes randomly-generated velocity messages for turtlesim */

#include <ros/ros.h>
#include <geometry_msgs/Twist.h> // For geometry_msgs::Twist
#include <stdlib.h>              // For rand() and RAND_MAX

int main(int argc , char **argv) {
    ros::init(argc, argv, "publish_velocity"); // Initialize the ROS system
    ros::NodeHandle nh;                       // Become a node
    ros::Publisher pub = nh.advertise<geometry_msgs::Twist>("turtle1/cmd_vel", 1000);
    srand(time(0)); // Seed the random number generator
    ros::Rate rate(2); // Loop at 2Hz until the node is shut down

    while(ros::ok()) {

        geometry_msgs::Twist msg; // Create the message
        msg.linear.x = double(rand())/double(RAND_MAX); // fill in the fields
        msg.angular.z = 2*3.14159*double(rand())/double(RAND_MAX)-1; // other fields default to 0
        pub.publish(msg); // Publish the message

        /* Send a message to rosout with the details */
        ROS_INFO_STREAM("Sending random velocity command:" << " linear =" << msg.linear.x << " angular =" << msg.angular.z);
        rate.sleep(); // Wait until it's time for another iteration
    }
}
```

Instantiate a publisher object

Initialize it by calling the advertise method of the node handle object

Publish message of type `geometry_msgs::Twist`

Publish on the `turtle1/cmd_vel` topic using a queue that can handle 1000 messages

# Example Program to Publish Messages

Send velocity messages on /turtle1/cmd\_vel

Edit **pubvel.cpp** and insert the following code

```
/* This program publishes randomly-generated velocity messages for turtlesim */

#include <ros/ros.h>
#include <geometry_msgs/Twist.h> // For geometry_msgs::Twist
#include <stdlib.h>             // For rand() and RAND_MAX

int main(int argc , char **argv) {
    ros::init(argc, argv, "publish_velocity"); // Initialize the ROS system
    ros::NodeHandle nh;                       // Become a node
    ros::Publisher pub = nh.advertise<geometry_msgs::Twist>("turtle1/cmd_vel", 1000);
    srand(time(0)); // Seed the random number generator
    ros::Rate rate(2); // Loop at 2Hz until the node is shut down

    while(ros::ok()) {
        geometry_msgs::Twist msg; // Create the message
        msg.linear.x = double(rand())/double(RAND_MAX); // fill in the fields
        msg.angular.z = 2*double(rand())/double(RAND_MAX)-1; // other fields default to 0
        pub.publish(msg); // Publish the message

        /* Send a message to roscout with the details */
        ROS_INFO_STREAM("Sending random velocity command:" << " linear =" << msg.linear.x << " angular =" << msg.angular.z);
        rate.sleep(); // Wait until it's time for another iteration
    }
}
```

Instantiate a rate object of the `ros::Rate` class to control how often the messages are published

# Example Program to Publish Messages

Send velocity messages on /turtle1/cmd\_vel

Edit **pubvel.cpp** and insert the following code

```
/* This program publishes randomly-generated velocity messages for turtlesim */

#include <ros/ros.h>
#include <geometry_msgs/Twist.h> // For geometry_msgs::Twist
#include <stdlib.h>              // For rand() and RAND_MAX

int main(int argc , char **argv) {
    ros::init(argc, argv, "publish_velocity"); // Initialize the ROS system
    ros::NodeHandle nh;                       // Become a node
    ros::Publisher pub = nh.advertise<geometry_msgs::Twist>("turtle1/cmd_vel", 1000);
    srand(time(0)); // Seed the random number generator
    ros::Rate rate(2); // Loop at 2Hz until the node is shut down

    while(ros::ok()) {
        geometry_msgs::Twist msg; // Create the message
        msg.linear.x = double(rand())/double(RAND_MAX); // fill in the fields
        msg.angular.z = 2*double(rand())/double(RAND_MAX)-1; // other fields default to 0
        pub.publish(msg); // Publish the message

        /* Send a message to roscout with the details */
        ROS_INFO_STREAM("Sending random velocity command:" << " linear =" << msg.linear.x << " angular =" << msg.angular.z);
        rate.sleep(); // Wait until it's time for another iteration
    }
}
```

Loop, sending messages, while `ros::ok()` returns true (it will return false if you use `roscout kill` on the node, or send an interrupt signal `ctrl-C`, or if the program calls `ros::shutdown()` to terminate itself)

# Example Program to Publish Messages

Send velocity messages on /turtle1/cmd\_vel

Edit **pubvel.cpp** and insert the following code

```
/* This program publishes randomly-generated velocity messages for turtlesim */

#include <ros/ros.h>
#include <geometry_msgs/Twist.h> // For geometry_msgs::Twist
#include <stdlib.h>              // For rand() and RAND_MAX

int main(int argc , char **argv) {
    ros::init(argc, argv, "publish_velocity"); // Initialize the ROS system
    ros::NodeHandle nh;                       // Become a node
    ros::Publisher pub = nh.advertise<geometry_msgs::Twist>("turtle1/cmd_vel", 1000);
    srand(time(0)); // Seed the random number generator
    ros::Rate rate(2); // Loop at 2Hz until the node is shut down

    while(ros::ok()) {

        geometry_msgs::Twist msg;
        msg.linear.x = double(rand())/double(RAND_MAX); // Create the message
        msg.angular.z = 2*double(rand())/double(RAND_MAX)-1; // fill in the fields
        pub.publish(msg); // other fields default to 0
        // Publish the message

        /* Send a message to rosout with the details */
        ROS_INFO_STREAM("Sending random velocity command:" << " linear =" << msg.linear.x << " angular =" << msg.angular.z);
        rate.sleep(); // Wait until it's time for another iteration
    }
}
```

msg.linear.y  
msg.linear.z  
msg.angular.x  
msg.angular.y  
are given their  
default values of zero

Set the linear velocity between 0 and 2

Set the angular velocity to a number between -1 and 1



# Example Program to Publish Messages

Send velocity messages on /turtle1/cmd\_vel

Edit **pubvel.cpp** and insert the following code

```
/* This program publishes randomly-generated velocity messages for turtlesim */

#include <ros/ros.h>
#include <geometry_msgs/Twist.h> // For geometry_msgs::Twist
#include <stdlib.h>              // For rand() and RAND_MAX

int main(int argc , char **argv) {
    ros::init(argc, argv, "publish_velocity"); // Initialize the ROS system
    ros::NodeHandle nh;                       // Become a node
    ros::Publisher pub = nh.advertise<geometry_msgs::Twist>("turtle1/cmd_vel", 1000);
    srand(time(0)); // Seed the random number generator
    ros::Rate rate(2); // Loop at 2Hz until the node is shut down

    while(ros::ok()) {

        geometry_msgs::Twist msg; // Create the message
        msg.linear.x = double(rand())/double(RAND_MAX); // fill in the fields
        msg.angular.z = 2*double(rand())/double(RAND_MAX)-1; // other fields default to 0
        pub.publish(msg); // Publish the message

        /* Send a message to rosout with the details */
        ROS_INFO_STREAM("Sending random velocity command:" << " linear =" << msg.linear.x << " angular =" << msg.angular.z);
        rate.sleep(); // Wait until it's time for another iteration
    }
}
```

Each call to this method causes a delay in the program. The duration of the delay is calculated to ensure the loop iterates at the required rate (in this case 2 Hz)

# Example Program to Publish Messages

Send velocity messages on `/turtle1/cmd_vel`

Build the workspace to compile the program

- Make sure you are in the workspace directory

```
~/workspace/ros/src/agitr/src$ cd ~/workspace/ros  
~/workspace/ros$
```

- Run `catkin_make`

```
~/workspace/ros$ catkin_make
```

# Example Program to Publish Messages

Send velocity messages on `/turtle1/cmd_vel`

If you have not already done it, open a terminal and enter

```
~$ roscore
```

Open a second terminal and enter

```
~$ rosruntime turtlesim turtlesim_node
```

Open a third terminal and enter

```
~$ rosruntime agitr pubvel
```

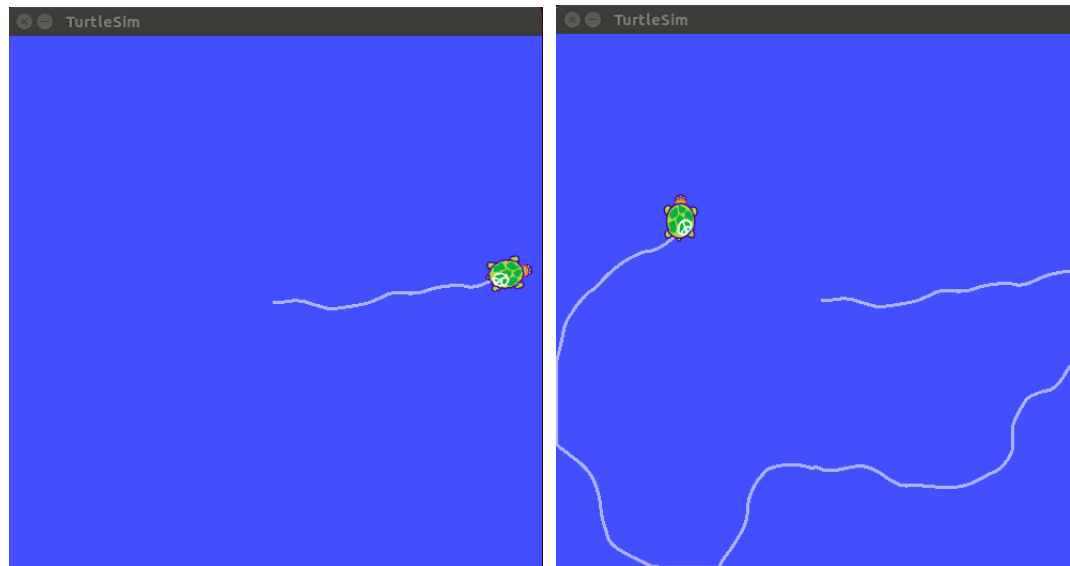
The separate terminals are intended to allow all three commands to execute simultaneously

# Example Program to Publish Messages

Send velocity messages on `/turtle1/cmd_vel`

If everything works correctly, you should see a graphical window similar to those below with the turtle moving randomly around

Again, the appearance of your turtle may differ. The simulator selects from a collection of “mascot” turtles for each of the historical distributions of ROS



# Example Program to Subscribe to Messages

Receive pose messages on /turtle1/pose

Make sure you are in the agitr sub-directory

```
~/workspace/ros/src$ cd ~/workspace/ros/src/agitr
```

# Example Program to Subscribe to Messages

Receive pose messages on /turtle1/pose

Edit **CMakeLists.txt**

Add the following lines at the end of the file

```
add_executable(subpose src/subpose.cpp)
target_link_libraries(subpose ${catkin_LIBRARIES})
```

# Example Program to Subscribe to Messages

Receive pose messages on `/turtle1/pose`

Move to the `agitr/src` sub-directory

```
~/workspace/ros/src/agitr$ cd src
```

```
~/workspace/ros/src/agitr/src$
```

# Example Program to Subscribe to Messages

Receive pose messages on /turtle1/pose

Edit **subpose.cpp** and insert the following code

```
/* This program subscribes to turtle1/pose and shows its messages on the screen */

#include <ros/ros.h>
#include <turtlesim/Pose.h>
#include <iomanip> // for std::setprecision and std::fixed

/* A callback function. Executed each time a new pose message arrives */
void poseMessageReceived(const turtlesim::Pose& msg) {
    ROS_INFO_STREAM(std::setprecision(2) << std::fixed <<
        "position=(" << msg.x << ", " << msg.y << ")" <<
        " direction=" << msg.theta);
}

int main(int argc, char **argv) {

    /* Initialize the ROS system and become a node */
    ros::init(argc, argv, "subscribe_to_pose");
    ros::NodeHandle nh;

    /* Create a subscriber object */
    ros::Subscriber sub = nh.subscribe("turtle1/pose", 1000, &poseMessageReceived);

    /* Let ROS take over */
    ros::spin();
}
```



# Example Program to Subscribe to Messages

Receive pose messages on /turtle1/pose

Edit **subpose.cpp** and insert the following code

```
/* This program subscribes to turtle1/pose and shows its messages on the screen */

#include <ros/ros.h>
#include <turtlesim/Pose.h>
#include <iomanip> // for std::setprecision and std::fixed

/* A callback function. Executed each time a new pose message arrives */
void poseMessageReceived(const turtlesim::Pose& msg) {
    ROS_INFO_STREAM(std::setprecision(2) << std::fixed <<
        "position=(" << msg.x << ", " << msg.y << ")" <<
        " direction=" << msg.theta);
}

int main(int argc, char **argv) {

    /* Initialize the ROS system and become a node */
    ros::init(argc, argv, "subscribe_to_pose");
    ros::NodeHandle nh;

    /* Create a subscriber object */
    ros::Subscriber sub = nh.subscribe("turtle1/pose", 1000, &poseMessageReceived);

    /* Let ROS take over */
    ros::spin();
}
```

The **callback** function is called every time a message is received  
We have to put the code to handle the message in this function

The parameter is the message that arrives.  
The type of the message is defined in the header file <turtlesim/Pose.h>

Here, we simply print the values to the terminal

# Example Program to Subscribe to Messages

Receive pose messages on /turtle1/pose

Edit **subpose.cpp** and insert the following code

```
/* This program subscribes to turtle1/pose and shows its messages on the screen */

#include <ros/ros.h>
#include <turtlesim/Pose.h>
#include <iomanip> // for std::setprecision and std::fixed

/* A callback function. Executed each time a new pose message arrives */
void poseMessageReceived(const turtlesim::Pose& msg) {
    ROS_INFO_STREAM(std::setprecision(2) << std::fixed <<
        "position=(" << msg.x << ", " << msg.y << ")" <<
        " direction=" << msg.theta);
}

int main(int argc, char **argv) {

    /* Initialize the ROS system and become a node */
    ros::init(argc, argv, "subscribe_to_pose");
    ros::NodeHandle nh;

    /* Create a subscriber object */
    ros::Subscriber sub = nh.subscribe("turtle1/pose", 1000, &poseMessageReceived);

    /* Let ROS take over */
    ros::spin();
}
```

Instantiate a subscriber object

Initialize it by calling the subscribe method of the node handle object

Subscribe on the  
turtle1/pose topic

using a queue that can  
handle 1000 messages

Pass a pointer to the callback function that is  
to be called when messages arrive

# Example Program to Subscribe to Messages

Receive pose messages on /turtle1/pose

Edit **subpose.cpp** and insert the following code

```
/* This program subscribes to turtle1/pose and shows its messages on the screen */

#include <ros/ros.h>
#include <turtlesim/Pose.h>
#include <iomanip> // for std::setprecision and std::fixed

/* A callback function. Executed each time a new pose message arrives */
void poseMessageReceived(const turtlesim::Pose& msg) {
    ROS_INFO_STREAM(std::setprecision(2) << std::fixed <<
        "position=(" << msg.x << ", " << msg.y << ")" <<
        " direction=" << msg.theta);
}

int main(int argc, char **argv) {

    /* Initialize the ROS system and become a node */
    ros::init(argc, argv, "subscribe_to_pose");
    ros::NodeHandle nh;

    /* Create a subscriber object */
    ros::Subscriber sub = nh.subscribe("turtle1/pose", 1000, &poseMessageReceived);

    /* Let ROS take over */
    ros::spin();
}
```

**Allow ROS to service the service the callback** by calling `ros::spin()`

Note: won't return control to this main() function; it will just carry on servicing the call the callback function

If you want to do more work here, call `ros::spinOnce()`

This will allow ROS to execute all pending callback calls and then return control to here

You'll probably embed this call in a loop so that you iteratively service the callback and then do some work

# Example Program to Subscribe to Messages

Receive pose messages on /turtle1/pose

Build the workspace to compile the program

- Make sure you are in the workspace directory

```
~/workspace/ros/src/agitr/src$ cd ~/workspace/ros  
~/workspace/rosr$
```

- Run catkin\_make

```
~/workspace/ros$ catkin_make
```

# Example Program to Subscribe to Messages

Receive pose messages on /turtle1/pose

If you have not already done it, open a terminal and enter

```
~$ roscore
```

If you have not already done it, open a second terminal and enter

```
~$ rosrun turtlesim turtlesim_node
```

If you have not already done it, open a third terminal and enter

```
~$ rosrun agitr pubvel
```

Open a fourth terminal and enter

```
~$ rosrun agitr subpose
```

# Example Program to Subscribe to Messages

Receive pose messages on /turtle1/pose

If everything works correctly, you should see messages in the fourth terminal detailing the pose of the turtle

```
parallels@ubuntu: ~  
[ INFO] [1579360943.664355158]: position=(3.92,5.76) direction=0.87  
[ INFO] [1579360943.680300126]: position=(3.94,5.77) direction=0.88  
[ INFO] [1579360943.695999972]: position=(3.95,5.79) direction=0.90  
[ INFO] [1579360943.712432184]: position=(3.97,5.81) direction=0.91  
[ INFO] [1579360943.727963946]: position=(3.98,5.83) direction=0.93  
[ INFO] [1579360943.744337128]: position=(4.00,5.85) direction=0.94  
[ INFO] [1579360943.759466689]: position=(4.01,5.87) direction=0.95  
[ INFO] [1579360943.776494199]: position=(4.02,5.89) direction=0.97  
[ INFO] [1579360943.791662779]: position=(4.04,5.91) direction=0.98  
[ INFO] [1579360943.808297093]: position=(4.05,5.93) direction=1.00  
[ INFO] [1579360943.823777829]: position=(4.06,5.95) direction=1.01  
[ INFO] [1579360943.839688780]: position=(4.08,5.98) direction=1.02  
[ INFO] [1579360943.855985925]: position=(4.09,6.00) direction=1.04  
[ INFO] [1579360943.872017937]: position=(4.10,6.02) direction=1.05  
[ INFO] [1579360943.888096973]: position=(4.11,6.04) direction=1.06  
[ INFO] [1579360943.903398620]: position=(4.12,6.06) direction=1.08  
[ INFO] [1579360943.920691261]: position=(4.13,6.08) direction=1.09
```

# Example Client Program to Use Services

`/clear, /turtle1/set_pen, /turtle1/teleport_absolute`

## 2.1.3 Services

`clear (std_srvs/Empty)`

Clears the turtlesim background and sets the color to the value of the background parameters.

`reset (std_srvs/Empty)`

Resets the turtlesim to the start configuration and sets the background color to the value of the background.

`kill (turtlesim/Kill)`

Kills a turtle by name.

`spawn (turtlesim/Spawn)`

Spawns a turtle at (x, y, theta) and returns the name of the turtle. Also will take name for argument but will fail if a duplicate name.

`turtleX/set_pen (turtlesim/SetPen)`

Sets the pen's color (r g b), width (width), and turns the pen on and off (off).

`turtleX/teleport_absolute (turtlesim/TeleportAbsolute)`

Teleports the turtleX to (x, y, theta).

`turtleX/teleport_relative (turtlesim/TeleportRelative)`

Teleports the turtleX a linear and angular distance from the turtles current position.

<http://wiki.ros.org/turtlesim>

# Example Client Program to Use Services

`/clear, /turtle1/set_pen, /turtle1/teleport_absolute`

Make sure you are in the agitr sub-directory

```
~/workspace/ros/src$ cd ~/workspace/ros/src/agitr
```



# Example Client Program to Use Services

`/clear, /turtle1/set_pen, /turtle1/teleport_absolute`

Edit **CMakeLists.txt**

Add the following lines at the end of the file

```
add_executable(useservices src/useservices.cpp)
target_link_libraries(useservices ${catkin_LIBRARIES})
```

# Example Client Program to Use Services

`/clear, /turtle1/set_pen, /turtle1/teleport_absolute`

Move to the **agitr/src** sub-directory

```
~/workspace/ros/src/agitr$ cd src
```

```
~/workspace/ros/src/agitr/src$
```

# Example Client Program to Use Services

/clear, /turtle1/set\_pen, /turtle1/teleport\_absolute

Edit **useservices.cpp** and insert the following code

```
/* This client program uses a sample of turtlesim services */
/* /clear, /turtle1/set_pen, and /turtle1/teleport_absolute */

#include <ros/ros.h>
#include <turtlesim/TeleportAbsolute.h> // for turtle1/teleport_absolute service
#include <turtlesim/SetPen.h>          // for turtle1/set_pen service
#include <std_srvs/Empty.h>           // for reset and clear services

int main(int argc, char **argv) {

    bool success = true;

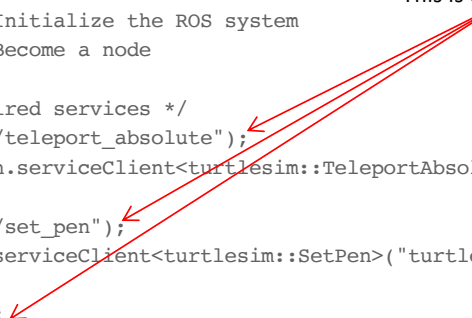
    /* Initialize the ROS system and become a node */
    ros::init(argc, argv, "dvernon"); // Initialize the ROS system
    ros::NodeHandle nh;                // Become a node

    /* Create client objects for the required services */
    ros::service::waitForService("turtle1/teleport_absolute");
    ros::ServiceClient teleportClient = nh.serviceClient<turtlesim::TeleportAbsolute>("turtle1/teleport_absolute");

    ros::service::waitForService("turtle1/set_pen");
    ros::ServiceClient setpenClient = nh.serviceClient<turtlesim::SetPen>("turtle1/set_pen");

    ros::service::waitForService("clear");
    ros::ServiceClient clearClient = nh.serviceClient<std_srvs::Empty>("clear");
```

`waitForService()` waits for a service to be advertised and available. Block until it is.  
This is defensive programming; normally it works without it.



# Example Client Program to Use Services

/clear, /turtle1/set\_pen, /turtle1/teleport\_absolute

Edit **useservices.cpp** and insert the following code

```
/* This client program uses a sample of turtlesim services */
/* /clear, /turtle1/set_pen, and /turtle1/teleport_absolute */

#include <ros/ros.h>
#include <turtlesim/TeleportAbsolute.h> // for turtle1/teleport_absolute service
#include <turtlesim/SetPen.h>          // for turtle1/set_pen service
#include <std_srvs/Empty.h>           // for reset and clear services

int main(int argc, char **argv) {

    bool success = true;

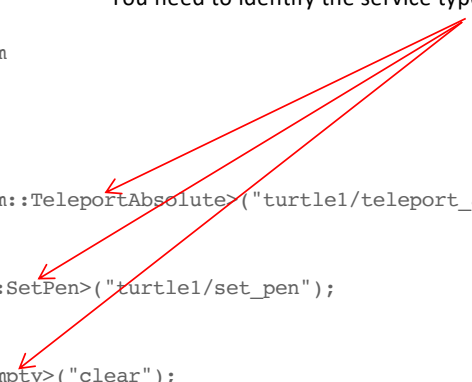
    /* Initialize the ROS system and become a node */
    ros::init(argc, argv, "dvernon"); // Initialize the ROS system
    ros::NodeHandle nh;                // Become a node

    /* Create client objects for the required services */
    ros::service::waitForService("turtle1/teleport_absolute");
    ros::ServiceClient teleportClient = nh.serviceClient<turtlesim::TeleportAbsolute>("turtle1/teleport_absolute");

    ros::service::waitForService("turtle1/set_pen");
    ros::ServiceClient setpenClient = nh.serviceClient<turtlesim::SetPen>("turtle1/set_pen");

    ros::service::waitForService("clear");
    ros::ServiceClient clearClient = nh.serviceClient<std_srvs::Empty>("clear");
```

You need to identify the service type when creating the client object



# Example Client Program to Use Services

/clear, /turtle1/set\_pen, /turtle1/teleport\_absolute

Edit **useservices.cpp** and insert the following code

```
/* This client program uses a sample of turtlesim services */
/* /clear, /turtle1/set_pen, and /turtle1/teleport_absolute */

#include <ros/ros.h>
#include <turtlesim/TeleportAbsolute.h> // for turtle1/teleport_absolute service
#include <turtlesim/SetPen.h> // for turtle1/set_pen service
#include <std_srvs/Empty.h> // for reset and clear services

int main(int argc, char **argv) {

    bool success = true;

    /* Initialize the ROS system and become a node */
    ros::init(argc, argv, "dvernon"); // Initialize the ROS system
    ros::NodeHandle nh; // Become a node

    /* Create client objects for the required services */
    ros::service::waitForService("turtle1/teleport_absolute");
    ros::ServiceClient teleportClient = nh.serviceClient<turtlesim::TeleportAbsolute>("turtle1/teleport_absolute");

    ros::service::waitForService("turtle1/set_pen");
    ros::ServiceClient setpenClient = nh.serviceClient<turtlesim::SetPen>("turtle1/set_pen");

    ros::service::waitForService("clear");
    ros::ServiceClient clearClient = nh.serviceClient<std_srvs::Empty>("clear");
```



The service types are defined in these include files

# Example Client Program to Use Services

`/clear, /turtle1/set_pen, /turtle1/teleport_absolute`

Edit `useservices.cpp` and insert the following code

```
/* This client program uses a sample of turtlesim services */
/* /clear, /turtle1/set_pen, and /turtle1/teleport_absolute */

#include <ros/ros.h>
#include <turtlesim/TeleportAbsolute.h> // for turtle1/teleport_absolute service
#include <turtlesim/SetPen.h>          // for turtle1/set_pen service
#include <std_srvs/Empty.h>           // for reset and clear services

int main(int argc, char **argv) {

    bool success = true;

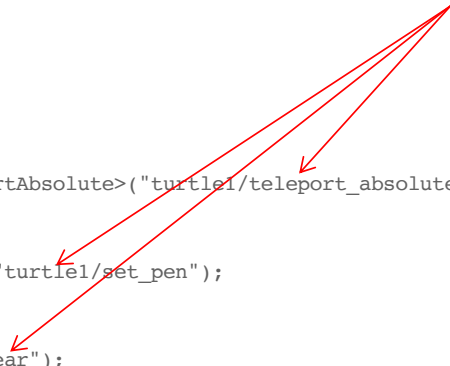
    /* Initialize the ROS system and become a node */
    ros::init(argc, argv, "dvernon"); // Initialize the ROS system
    ros::NodeHandle nh;               // Become a node

    /* Create client objects for the required services */
    ros::service::waitForService("turtle1/teleport_absolute");
    ros::ServiceClient teleportClient = nh.serviceClient<turtlesim::TeleportAbsolute>("turtle1/teleport_absolute");

    ros::service::waitForService("turtle1/set_pen");
    ros::ServiceClient setpenClient = nh.serviceClient<turtlesim::SetPen>("turtle1/set_pen");

    ros::service::waitForService("clear");
    ros::ServiceClient clearClient = nh.serviceClient<std_srvs::Empty>("clear");
```

You also need to define a string naming the service you want to call



# Example Client Program to Use Services

/clear, /turtle1/set\_pen, /turtle1/teleport\_absolute

```
/* Create the service objects services */
turtlesim::TeleportAbsolute teleport_arguments; // reposition the turtle without locomotion
turtlesim::SetPen          pen_arguments;      // turn the pen on/off and change colour
std_srvs::Empty            clear_arguments;    // clear the background

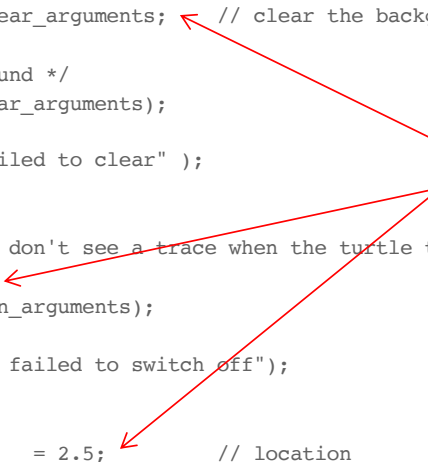
/* clear the simulator background */
success = clearClient.call(clear_arguments);
if (!success) {
    ROS_ERROR_STREAM("Turtle failed to clear" );
}

/* turn the pen off so that we don't see a trace when the turtle teleports */
pen_arguments.request.off = 1;
success = setpenClient.call(pen_arguments);
if (!success) {
    ROS_ERROR_STREAM("TurtlePen failed to switch off");
}

teleport_arguments.request.x    = 2.5; // location
teleport_arguments.request.y    = 3.5; // coordinates
teleport_arguments.request.theta = 3.14159 / 2; // facing up, i.e. 90 degrees

success = teleportClient.call(teleport_arguments);
if (!success) {
    ROS_ERROR_STREAM("Turtle failed to teleport" );
}
```

We instantiate the service classes as service objects  
so that we can assign values to the request members



# Example Client Program to Use Services

/clear, /turtle1/set\_pen, /turtle1/teleport\_absolute

```
/* Create the service objects services */
turtlesim::TeleportAbsolute teleport_arguments; // reposition the turtle without locomotion
turtlesim::SetPen          pen_arguments;      // turn the pen on/off and change colour
std_srvs::Empty            clear_arguments;    // clear the background

/* clear the simulator background */
success = clearClient.call(clear_arguments);
if (!success) {
    ROS_ERROR_STREAM("Turtle failed to clear" );
}

/* turn the pen off so that we don't see a trace when the turtle teleports */
pen_arguments.request.off = 1;
success = setpenClient.call(pen_arguments);
if (!success) {
    ROS_ERROR_STREAM("TurtlePen failed to switch off");
}

teleport_arguments.request.x      = 2.5;      // location
teleport_arguments.request.y      = 3.5;      // coordinates
teleport_arguments.request.theta  = 3.14159 / 2; // facing up, i.e. 90 degrees

success = teleportClient.call(teleport_arguments);
if (!success) {
    ROS_ERROR_STREAM("Turtle failed to teleport" );
}
```

We call the service with the associated arguments



# Example Client Program to Use Services

/clear, /turtle1/set\_pen, /turtle1/teleport\_absolute

```
/* Create the service objects services */
turtlesim::TeleportAbsolute teleport_arguments; // reposition the turtle without locomotion
turtlesim::SetPen          pen_arguments;      // turn the pen on/off and change colour
std_srvs::Empty            clear_arguments;    // clear the background

/* clear the simulator background */
success = clearClient.call(clear_arguments);
if (!success) {
    ROS_ERROR_STREAM("Turtle failed to clear" );
}

/* turn the pen off so that we don't see a trace when the turtle teleports */
pen_arguments.request.off = 1;
success = setpenClient.call(pen_arguments);
if (!success) {
    ROS_ERROR_STREAM("TurtlePen failed to switch off");
}

teleport_arguments.request.x      = 2.5;      // location
teleport_arguments.request.y      = 3.5;      // coordinates
teleport_arguments.request.theta = 3.14159 / 2; // facing up, i.e. 90 degrees

success = teleportClient.call(teleport_arguments);
if (!success) {
    ROS_ERROR_STREAM("Turtle failed to teleport");
}
```

Check to ensure the service call was successful and report an error if not

# Example Client Program to Use Services

`/clear, /turtle1/set_pen, /turtle1/teleport_absolute`

```
/* turn the pen again so that we do see a trace when the turtle moves later on */
pen_arguments.request.off = 0;
pen_arguments.request.r = 255; // white
pen_arguments.request.g = 255; // pen
pen_arguments.request.b = 255; // colour
pen_arguments.request.width = 1; // narrow line

success = setpenClient.call(pen_arguments);
if (!success) {
    ROS_ERROR_STREAM("TurtlePen failed to switch on");
}
}
```

# Example Client Program to Use Services

`/clear, /turtle1/set_pen, /turtle1/teleport_absolute`

Build the workspace to compile the program

- Make sure you are in the workspace directory

```
~/workspace/ros/src/agitr/src$ cd ~/workspace/ros
```

```
~/workspace/rosr$
```

- Run `catkin_make`

```
~/workspace/ros$ catkin_make
```

# Example Client Program to Use Services

`/clear, /turtle1/set_pen, /turtle1/teleport_absolute`

If you have not already done it, open a terminal and enter

```
~$ roscore
```

If you have not already done it, open a second terminal and enter

```
~$ rosruntime turtlesim turtlesim_node
```

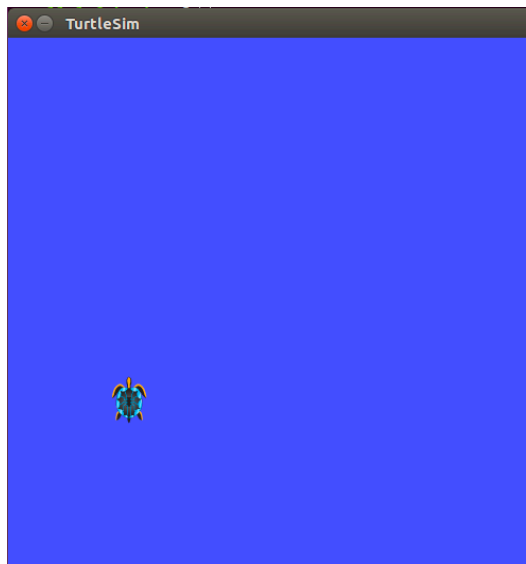
Open a third terminal and enter

```
~$ rosruntime agitr useservices
```

# Example Client Program to Use Services

`/reset, /clear, /turtle1/set_pen, /turtle1/teleport_absolute`

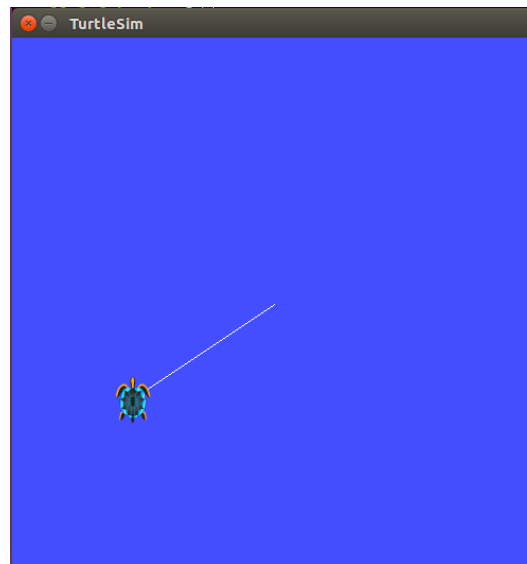
If everything works correctly, you should see the following



# Example Client Program to Use Services

`/reset, /clear, /turtle1/set_pen, /turtle1/teleport_absolute`

If you comment out the first call to the `set_pen` service to turn the pen off, run `catkin_make`, and run the program again, you should see the following



# ROS Resources

Wiki	<a href="http://wiki.ros.org/">http://wiki.ros.org/</a>
Installation	<a href="http://wiki.ros.org/ROS/Installation">http://wiki.ros.org/ROS/Installation</a>
Tutorials	<a href="http://wiki.ros.org/ROS/Tutorials">http://wiki.ros.org/ROS/Tutorials</a>
Tutorial Videos	<a href="http://www.youtube.com/playlist?list=PLDC89965A56E6A8D6">http://www.youtube.com/playlist?list=PLDC89965A56E6A8D6</a>
ROS Cheat Sheet	<a href="http://www.tedusar.eu/files/summerschool2013/ROScheatsheet.pdf">http://www.tedusar.eu/files/summerschool2013/ROScheatsheet.pdf</a>

# Recommended Reading

[http://wiki.ros.org/catkin/Tutorials/create\\_a\\_workspace](http://wiki.ros.org/catkin/Tutorials/create_a_workspace)

<http://wiki.ros.org/ROS/Tutorials/CreatingPackage>

<http://wiki.ros.org/roscpp/Overview/InitializationandShutdown>

<http://wiki.ros.org/roscpp/Overview/NodeHandles>

<http://wiki.ros.org/ROS/Tutorials/BuildingPackages>

[http://wiki.ros.org/ROS/Tutorials/WritingPublisherSubscriber\(c++\)](http://wiki.ros.org/ROS/Tutorials/WritingPublisherSubscriber(c++))

J. M. O’Kane, A Gentle Introduction to ROS, 2014.

<https://cse.sc.edu/~jokane/agitr/>