Velocity Commands

- To make a robot move in ROS we need to publish Twist messages to the topic cmd_vel
- This message has a linear component for the (x,y,z) velocities, and an angular component for the angular rate about the (x,y,z) axes

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

A Move Turtle Node

- For the demo, we will create a new ROS package called my_turtle

```
$ cd ~/catkin_ws/src
$ catkin_create_pkg my_turtle std_msgs rospy roscpp
```

- In Eclipse add a new source file to the package called Move_Turtle.cpp
- Add the following code

MoveTurtle.cpp

```
#include "ros/ros.h"
#include "geometry_msgs/Twist.h"

int main(int argc, char **argv)
{
    const double FORWARD_SPEED_MPS = 0.5;

    // Initialize the node
    ros::init(argc, argv, "move_turtle");
    ros::NodeHandle node;

    // A publisher for the movement data
    ros::Publisher pub = node.advertise<geometry_msgs::Twist>("turtle1/cmd_vel", 10);

    // Drive forward at a given speed.  The robot points up the x-axis.
    // The default constructor will set all commands to 0
    geometry_msgs::Twist msg;
    msg.linear.x = FORWARD_SPEED_MPS;

    // Loop at 10Hz, publishing movement commands until we shut down
    ros::Rate rate(10);
    ROS_INFO("Starting to move forward");
    while (ros::ok()) {
        pub.publish(msg);
        rate.sleep();
    }
}
```

Launch File

- Add move_turtle.launch to your package:

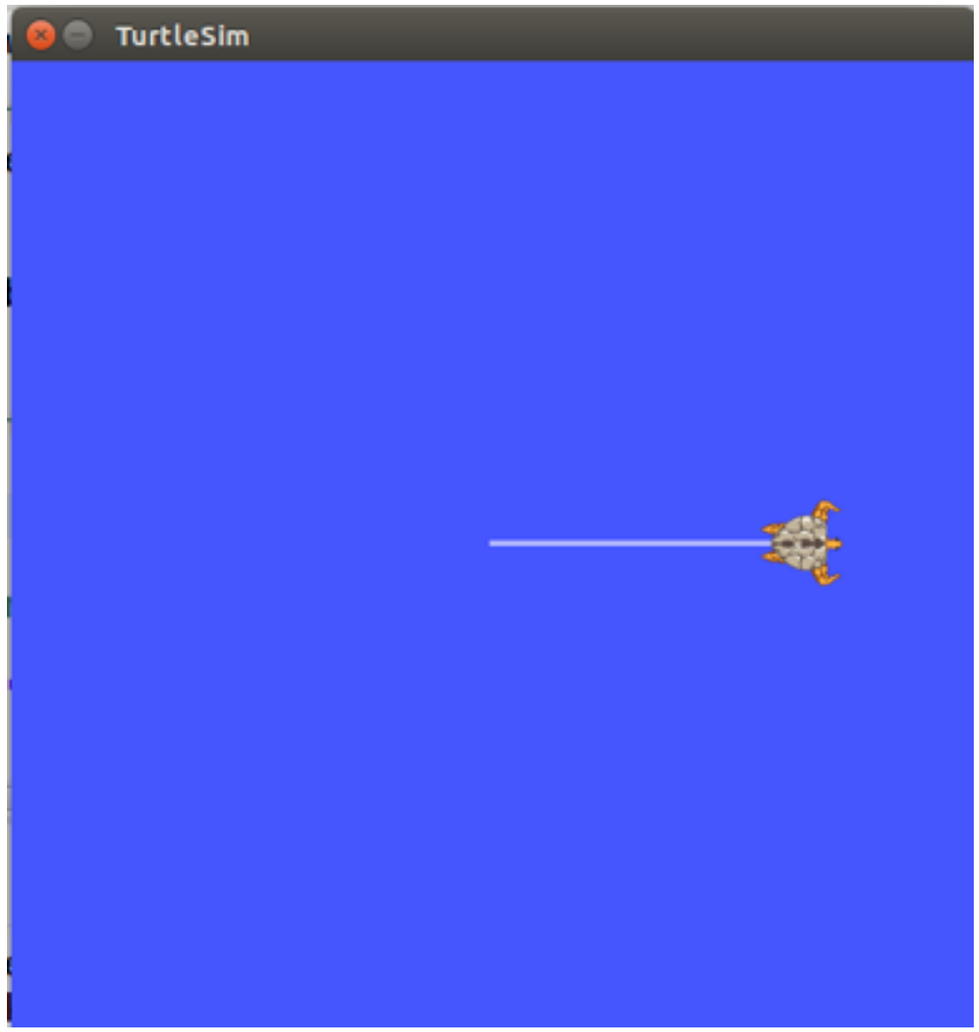
```
<launch>
  <node name="turtlesim_node" pkg="turtlesim" type="turtlesim_node" />
  <node name="move_turtle" pkg="my_turtle" type="move_turtle"
output="screen" />
</launch>
```

- Run the launch file:

```
$ roslaunch my_turtle move_turtle.launch
```

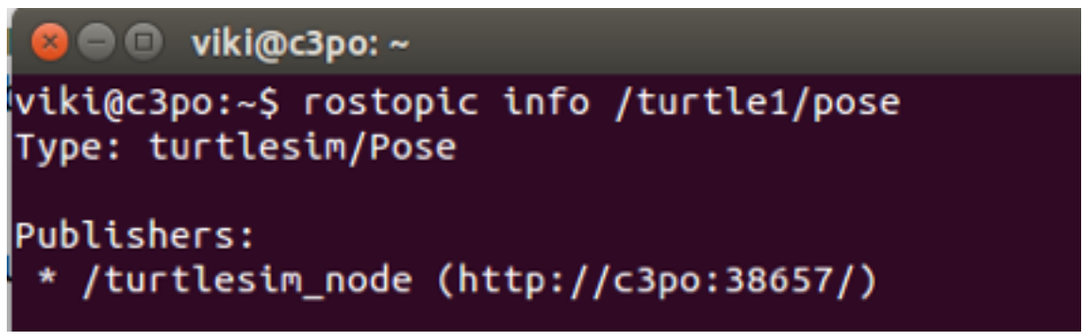
Move Turtle Demo

- You should see the turtle in the simulator constantly moving forward until it bumps into the wall



Print Turtle's Pose

- In order to print the turtle's pose we need to subscribe to the topic /turtle1/pose
- First, we find the message type of the topic by running the command rosmsg type /turtle1/pose



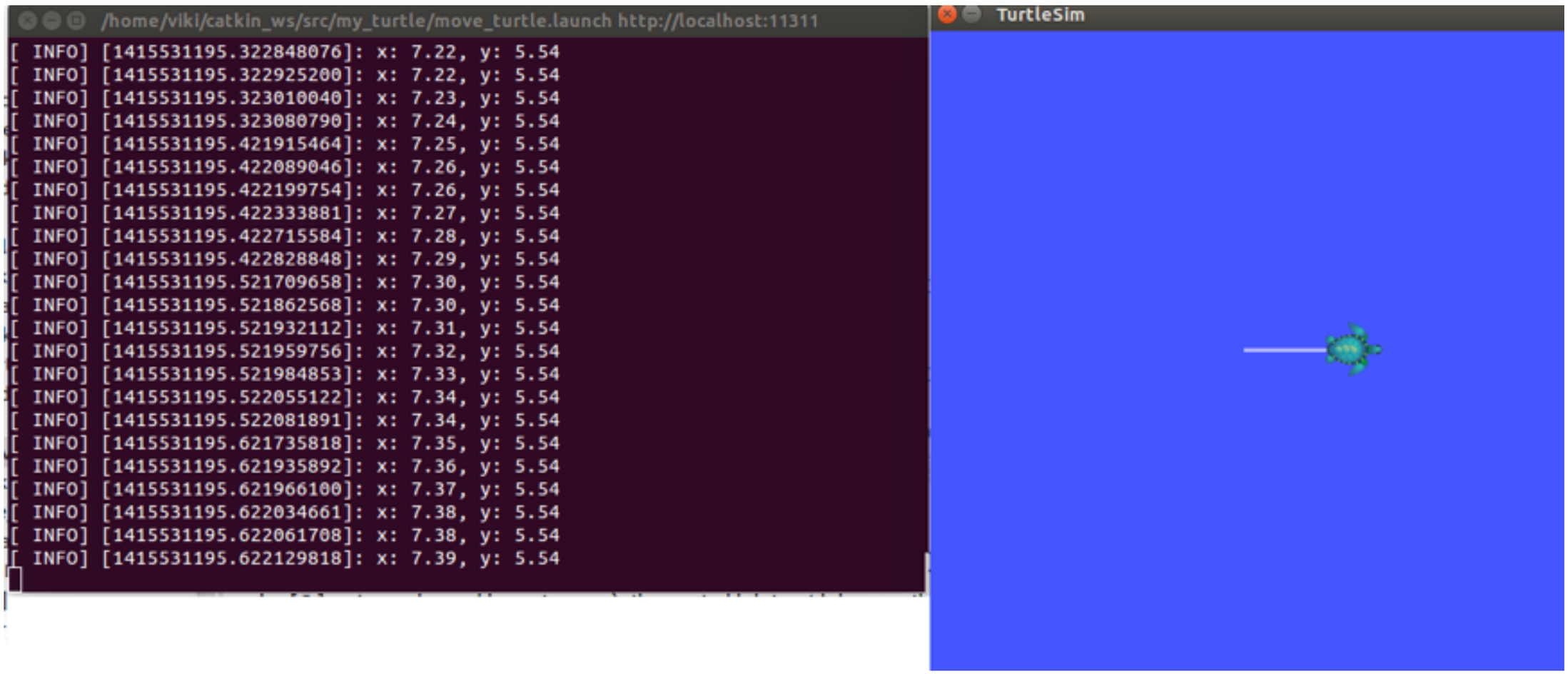
- Message type is turtlesim/Pose
- This is a specific message in turtlesim package, thus we need to include the header "turtlesim/Pose.h" in order to work with message of this type

MoveTurtle.cpp

```
#include "ros/ros.h"
#include "geometry_msgs/Twist.h"
#include "turtlesim/Pose.h"

// Topic messages callback
void poseCallback(const turtlesim::PoseConstPtr& msg)
{
    ROS_INFO("x: %.2f, y: %.2f", msg->x, msg->y);
}

int main(int argc, char **argv)
{
    const double FORWARD_SPEED_MPS = 0.5;

    // Initialize the node
    ros::init(argc, argv, "move_turtle");
    ros::NodeHandle node;

    // A publisher for the movement data
    ros::Publisher pub = node.advertise<geometry_msgs::Twist>("turtle1/cmd_vel", 10);

    // A listener for pose
    ros::Subscriber sub = node.subscribe("turtle1/pose", 10, poseCallback);
    // Drive forward at a given speed.  The robot points up the x-axis.
    // The default constructor will set all commands to 0
    geometry_msgs::Twist msg;
    msg.linear.x = FORWARD_SPEED_MPS;

    // Loop at 10Hz, publishing movement commands until we shut down
    ros::Rate rate(10);
    ROS_INFO("Starting to move forward");
    while (ros::ok()) {
        pub.publish(msg);
        ros::spinOnce(); // Allow processing of incoming messages
        rate.sleep();
    }
}
```

Print Turtle's Pose

- roslaunch my_turtle move_turtle.launch



Passing Arguments To Nodes

- In the launch file you can use the args attribute to pass command-line arguments to node
- In our case, we will pass the name of the turtle as an argument to move_turtle

```
<launch>
  <node name="turtlesim_node" pkg="turtlesim" type="turtlesim_node" />
  <node name="move_turtle" pkg="my_turtle" type="move_turtle"
args="turtle1" output="screen"/>
</launch>
```

MoveTurtle.cpp

```
#include "ros/ros.h"
#include "geometry_msgs/Twist.h"
int main(int argc, char **argv)
{
    const double FORWARD_SPEED_MPS = 0.5;
    string robot_name = string(argv[1]);

    // Initialize the node
    ros::init(argc, argv, "move_turtle");
    ros::NodeHandle node;

    // A publisher for the movement data
    ros::Publisher pub = node.advertise<geometry_msgs::Twist>(robot_name + "/cmd_vel", 10);

    // A listener for pose
    ros::Subscriber sub = node.subscribe(robot_name + "/pose", 10, poseCallback);

    geometry_msgs::Twist msg;
    msg.linear.x = FORWARD_SPEED_MPS;

    ros::Rate rate(10);
    ROS_INFO("Starting to move forward");
    while (ros::ok()) {
        pub.publish(msg);
        ros::spinOnce(); // Allow processing of incoming messages
        rate.sleep();
    }
}
```