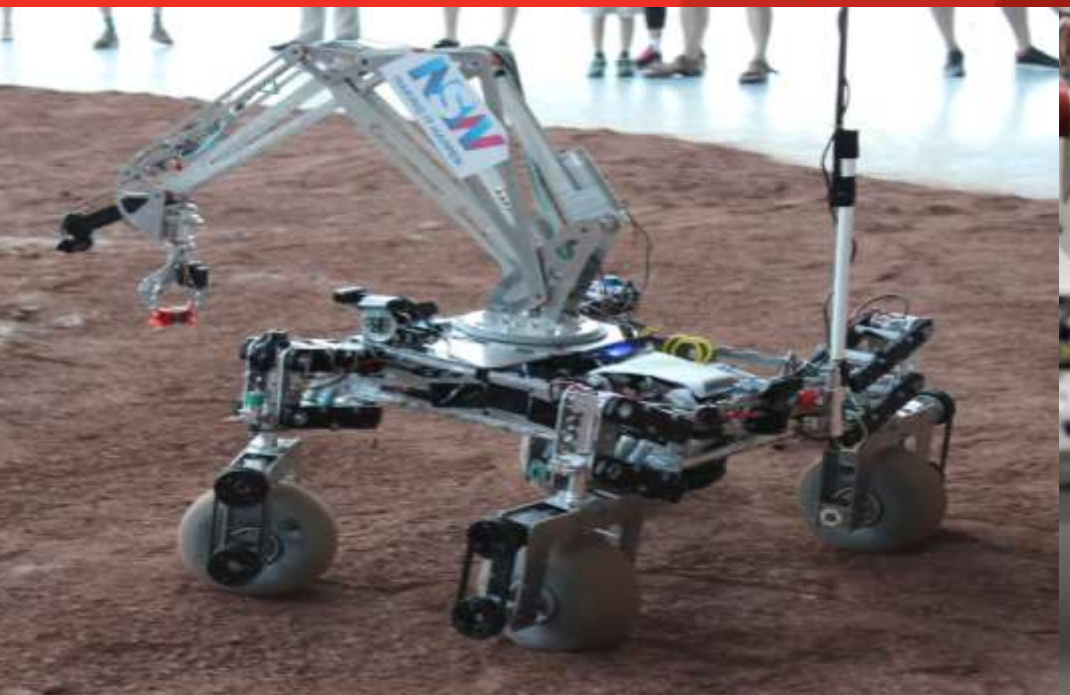


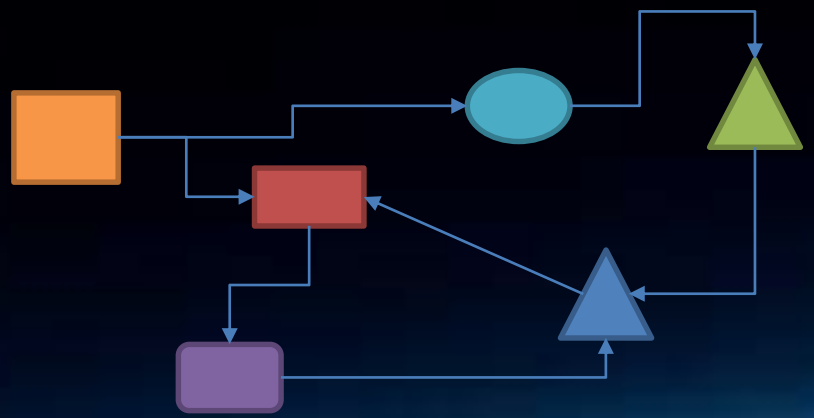

UNSW AUSTRALIA

An Introduction to ROS

Or how to write code that starts fires!

Never Stand Still | Engineering





A Conceptual Overview

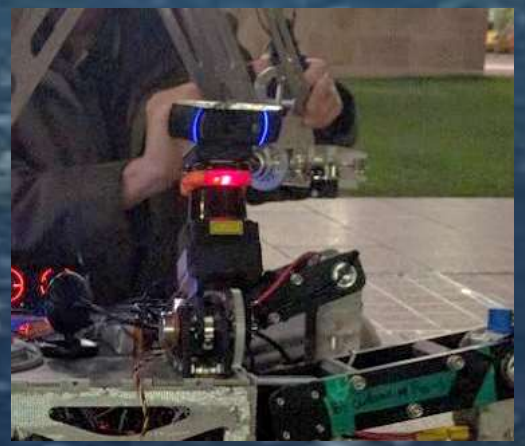
Nodes, Topics and Messages

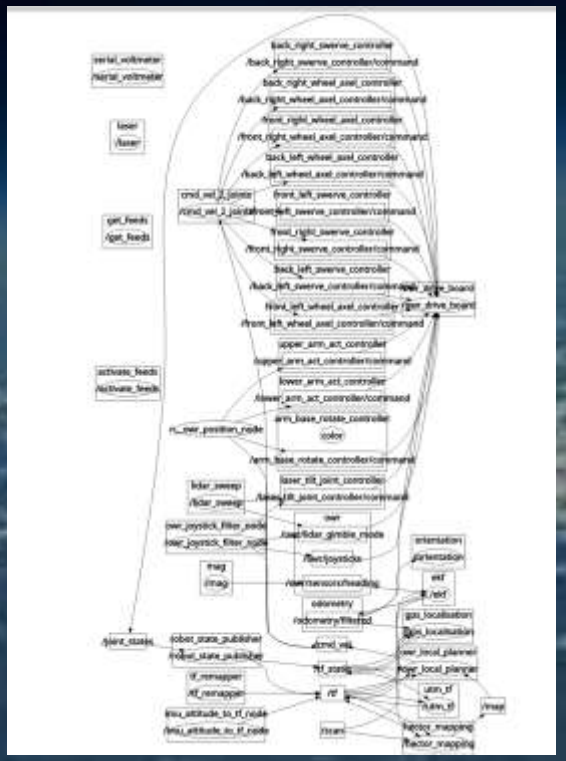
UNSW AUSTRALIA

Never Stand Still | Engineering

What is ROS?

- Stands for Robotics Operating System
- NOT an Operating System.
- Provides many standard tools and libraries for:
  - Navigation
  - Visualisation
  - Interfacing with Hardware
  - Debugging
- Provides a Framework for Modularity



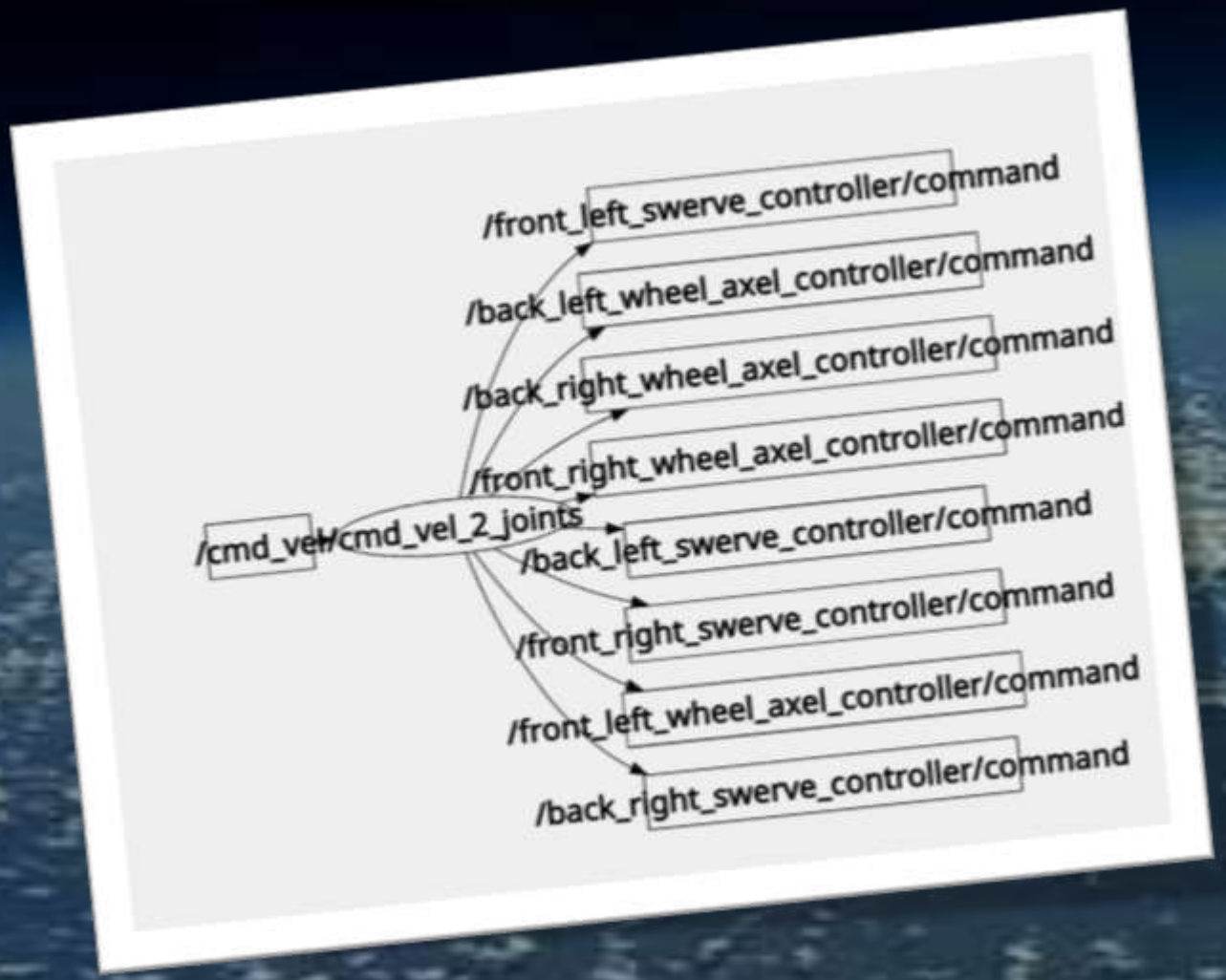


Nodes

- A node is a ROS process/program.
- Use the “rosrun” command to run one:

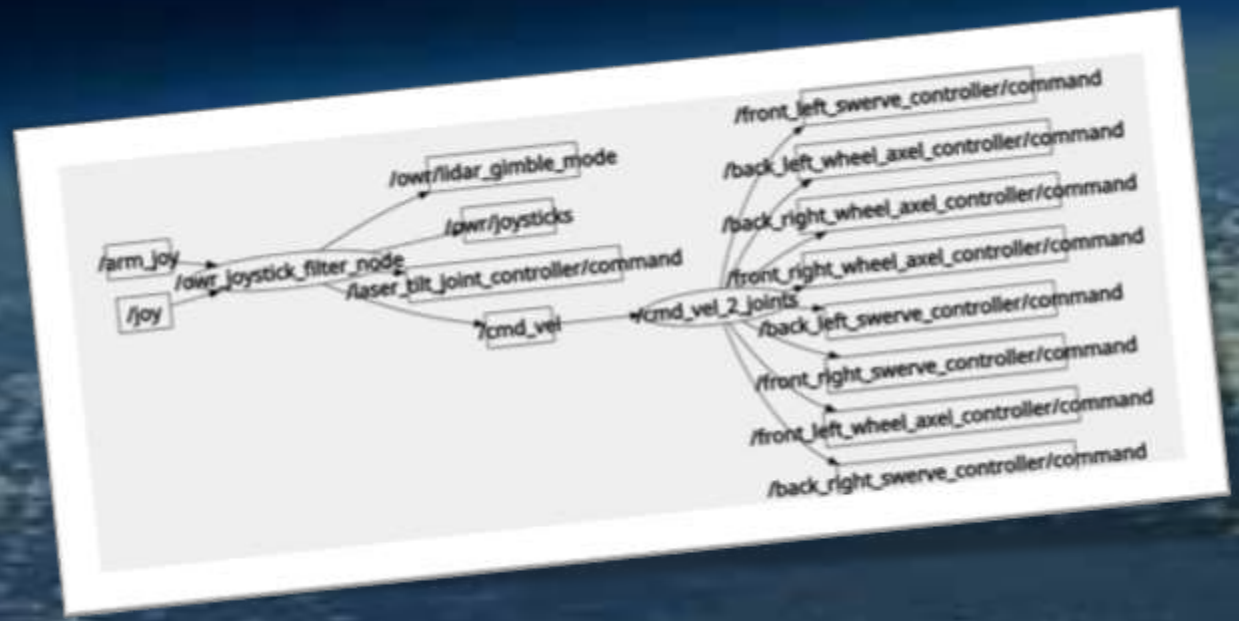
```
rosrun  [package name] [node name]
```

- Publishes and/or Receives Messages.
- Can be something you’ve written yourself or provided by a third party.
- Live in ROS packages.
- The node and package name must be lower case with underscores.



Topics

- Like TV channels or radio bands, allowing for many channels
- Each has a set type of “Message”
- Nodes can:
  - Publish Topics
  - Subscribe Topics
- Topics are named in lower case, with underscores. They can use “/” to signify related topics



Example of a Message: geometry_msgs/Vector3

- Package is “geometry_msgs”
- Message name is “Vector3”
- To use in C++

```
geometry_msgs::Vector3
```

- To #include

```
#include <geometry_msgs/Vector3.h>
```

- To refer it using rostopic

```
geometry_msgs/vector3
```

geometry_msgs/Vector3 Message

File: geometry_msgs/Vector3.msg

Raw Message Definition

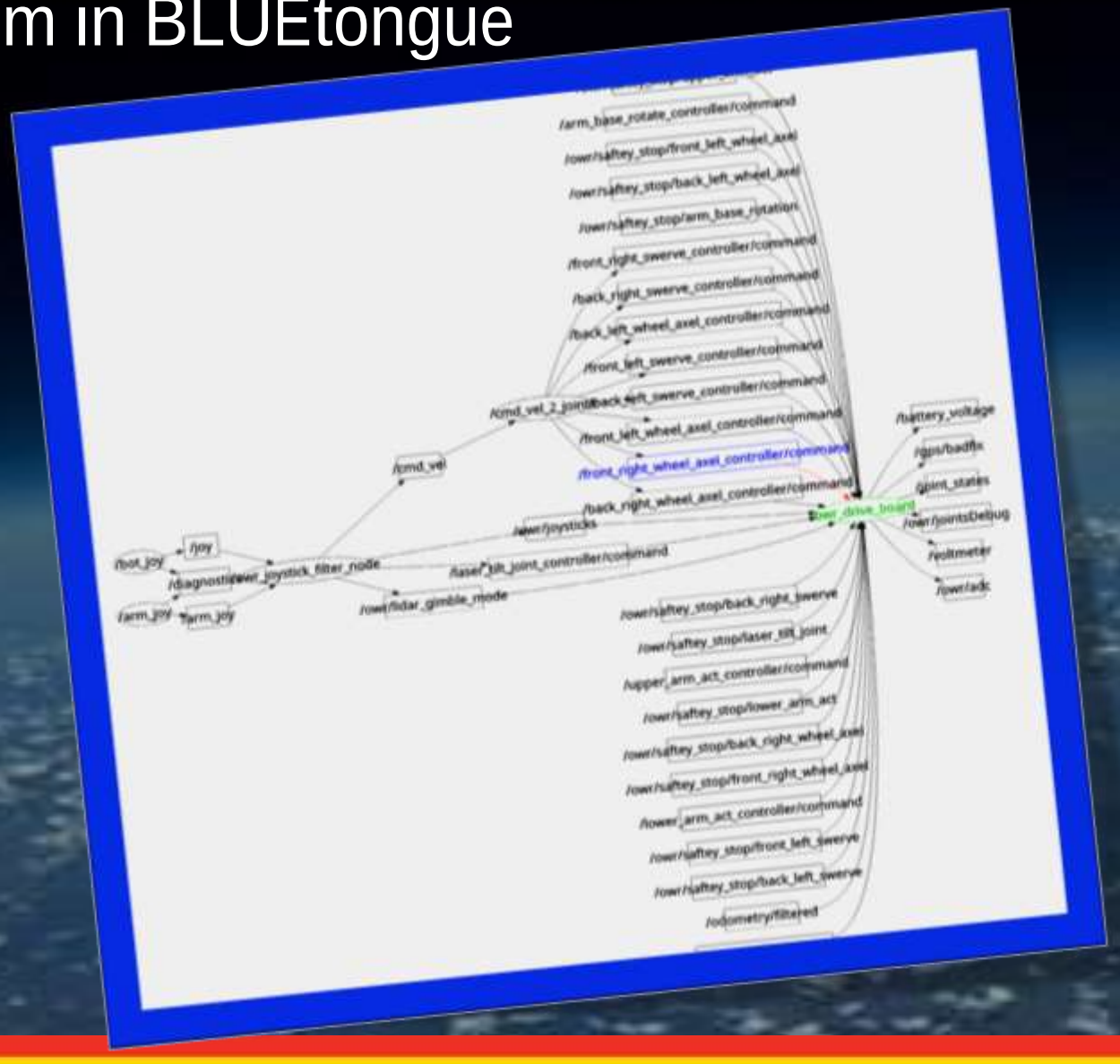
```
# This represents a vector in free space.
# It is only meant to represent a direction. Therefore, it does not
# make sense to apply a translation to it (e.g., when applying a
# generic rigid transformation to a Vector3, tf2 will only apply the
# rotation). If you want your data to be translatable too, use the
# geometry_msgs/Point message instead.

float64 x
float64 y
float64 z
```

Compact Message Definition

```
float64 x
float64 y
float64 z
```

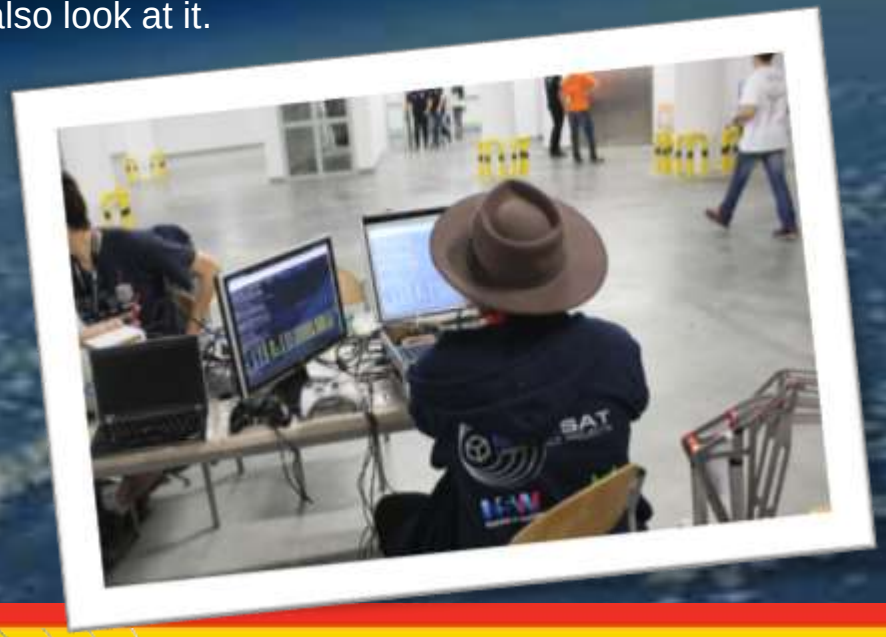
autogenerated on Tue, 17 Jan 2017 20:57:22

How we use them in BLUEtongue



Case Study: The /owr/jointsDebug topic.

- Used to debug the positions of joints, and the command being sent to them
- Send a *owr_messages::board* message.
  - This message contains a *owr_messages::pwm* message.
- Published by the *owr_drive_board* node.
- Subscribers are primarily debugging tools, but the new gui will also look at it.



board.msg

pwm[] joints
Header header

pwm.msg

string joint
int32 pwm
float64 targetVel
float64 currentVel
float64 currentPos
float64 targetPos

UNSW AUSTRALIA

Exercise 1: Simple Subscriber

Or How to Pass Notes in ROS

Never Stand Still | Engineering

Workspaces and Packages

1. Open up your VM.
2. Create a workspace

```
$ mkdir –p ~/tutorial-ws/src
$ cd ~/tutorial-ws
$ catkin_make
[ …lots of output goes here… ]
```

3. Create a package

```
$ cd src
$ catkin_create_pkg my_pubsub roscpp std_msgs
Created file my_pubsub/CMakeLists.txt
Created file my_pubsub/package.xml
Created folder my_pubsub/include/my_pubsub
Created folder my_pubsub/src
Successfully created files in /home/ros/tutorial-ws/src/my_pubsub. Please adjust the values in package.xml.
$ cd my_pubsub
```

Package.xml and CMakeLists.txt

- Two core files that control the build system.
- Specify dependencies.

4. Open package.xml
    1. Add your name as the maintainer
    2. Set the license.
        - (At BLUEsat OWRS we use the MIT license for most our code)
5. Open CMakeLists.txt
    1. Note the dependencies and project name
    2. Have a brief look at the commented out sections. These are examples of how you add different types of files.

A Basic Node

- Each node:
  - needs a header file and a .cpp file.
  - has a reference in its CMakeLists.txt file.
  - has a main function.
- Nodes in this package will be run using the following command format:

```
rosrun  my_pubsub [node name]
```

A Basic Node (cont.)

1. Create the two source files:

```
$ cd  ~/tutorial-ws/src/my_pubsub/
$ touch src/pub_node.cpp include/my_pubsub/pub_node.hpp
```

2. Open up pub_node.hpp in your favourite text editor and copy the file on the next slide

```
/*
 * Date Started: 09/03/2017
 * Original Author: Harry J.E Day
 * Editors:
 * ROS Node Name: pub_node
 * ROS Package: my_pubub
 * Purpose: To demonstrate a simple publisher node.
 * This code is released under the MIT  License. Copyright BLUEsat UNSW, 2017
 */

// These import the main ros libary and the message we are going to send
#include <ros/ros.h>
#include <std_msgs/Float64.h>

class Pub_Node {
   public:
      Pub_Node();
      void spin();

  private:
     ros::NodeHandle nh;
     ros::Publisher pub;

};
```

include/my_pubsub/pub_node.hpp

A Basic Node (cont)

3. Add this at the start of the src/pub_node.cpp file:

```
/*
 * Date Started: 09/03/2017
 * Original Author: Harry J.E Day
 * Editors:
 * ROS Node Name: pub_node
 * ROS Package: my_pubsub
 * Purpose: To demonstrate a simple publisher node.
 * This code is released under the MIT  License. Copyright BLUEsat UNSW, 2017
 */

#include "my_pubsub/pub_node.hpp"
```

A Basic Node – Main Function

4. And this main function

```
int main(int argc, char ** argv) {
    // starts ros, code will block here if roscore is not running
    ros::init(argc, argv, "pub_node");
    ROS_INFO("Hello Robot");
    Pub_Node my_node;
    my_node.spin();

    return EXIT_SUCCESS;
}
```

A Basic Node - Stubs

5. Finally in order to make it compile we need to add the following stub code.

```
Pub_Node::Pub_Node() {}

void Pub_Node::spin() {}
```

A Basic Node – CMakeLists.txt

Now we need to tell the compiler about it:

1. Open up ~/tutorial-ws/src/my_pubsub/CMakeLists.txt in you favourite editor
2. Edit the following:

```
include_directories(
  $(catkin_INCLUDE_DIRS)
)
```

So it looks like this:

```
include_directories(
  $(catkin_INCLUDE_DIRS)
  include
)
```

A Basic Node – CMakeLists.txt (cont)

3. Finally, add the following lines towards the bottom of the file:

```
add_executable(pub_node src/pub_node.cpp)
add_dependencies(pub_node ${${PROJECT_NAME}_EXPORTED_TARGETS} ${catkin_EXPORTED_TARGETS})
target_link_libraries(pub_node ${catkin_LIBRARIES})
```

A Basic Node – Compile and Run

- To compile run the following commands

```
$ cd ~/tutorial-ws
$ catkin_make
[ …lots of output goes here… ]
[100%] Built target pub_node
```

- To run, run the following:

Terminal 1

```
$ roscore
[ …lots of output goes here… ]
```

Terminal 2

```
$ source devel/setup.bash
$ rosrun my_pubsub pub_node
[ INFO] [1489139498.792145514]: Hello Robot!
```

Publisher

- We’re going to publish to the /upper_arm_act_controller/command which is std_msgs:Float64.
- We can visualise this in the gui.

Publisher (cont)

1. Open up ~/tutorial-ws/src/my_pubsub/src/pub_node.cpp
2. Edit the constructor *(Pub_Node:Pub_Node)* so it looks like this:

```
Pub_Node::Pub_Node() {
    pub = nh.advertise<std_msgs::Float64>("/upper_arm_act_controller/command", 1, true);
}
```

3. Then edit the spin function so it looks like this:

```
void Pub_Node::spin() {
    std_msgs::Float64 msg;
    while(ros::ok()) {
        msg.data = 1000;
        pub.publish(msg);
        ros::spinOnce();
    }
}
```

Publisher (cont)

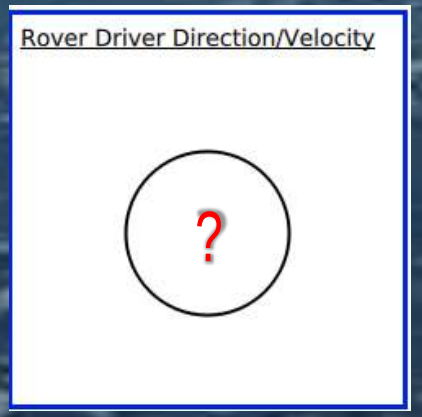
4. Run this:

```
$ cd ~/tutorial-ws
$ catkin_make
[ …lots of output goes here… ]
$  rosrun my_pubsub pub_node
```

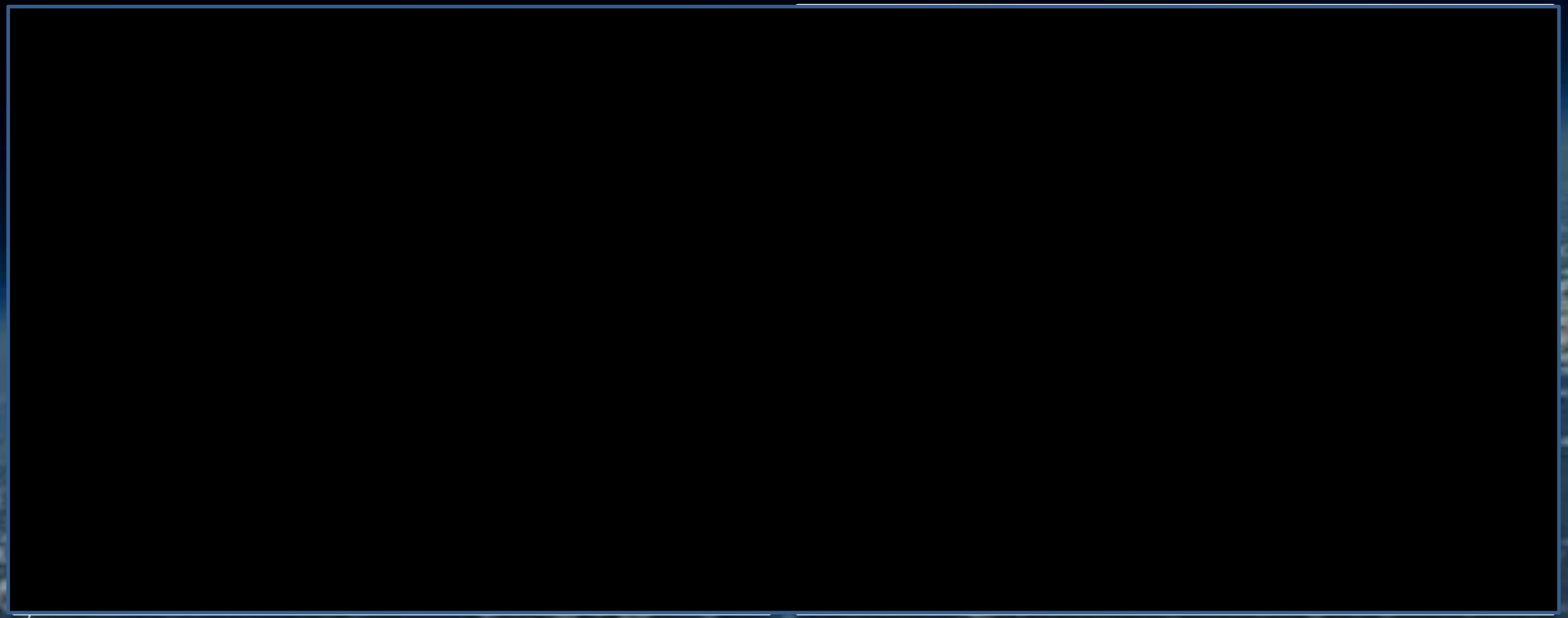
Exercises

1. Modify the publisher so it causes the text to be “retract” rather than “extend”
2. In groups modify the simple publisher so it changes the rover direction widget rather than the arm extension. The topic for this is “/cmd_vel” and the message type is “geometry_msgs/twist” and it uses linear.x and linear.y

   Hint: You’ll need to add the dependencies to the package, and look at the message definition which can be found at: http://docs.ros.org/api/geometry_msgs/html/msg/Twist.html



Solution



And Add These Lines

package.xml

```
<run_depend>geometry_msgs</run_depend>
<build_depend>geometry_msgs</build_depend>
```

CMakeLists.txt (modify)

```
find_package(catkin REQUIRED COMPONENTS
  roscpp
  std_msgs
  geometry_msgs
)
```

UNSW AUSTRALIA

Example 2: A Simple Subscriber

Or how to receive notes in ROS

Never Stand Still | Engineering

Subscriber – Quick Recap

- Reads ROS messages from other nodes.
- What we will build:
  - Can read what we have sent in our publisher
  - Can read the direction of the rover

Subscriber – Initial Setup

1. Create files for the new node

```
$ cd  ~/tutorial-ws/src/my_pubsub/
$ touch src/sub_node.cpp include/my_pubsub/sub_node.hpp
```

2. Add the following to CMakeLists.txt

```
add_executable(sub_node src/sub_node.cpp)
add_dependencies(sub_node ${${PROJECT_NAME}_EXPORTED_TARGETS} ${catkin_EXPORTED_TARGETS})
target_link_libraries(sub_node ${catkin_LIBRARIES})
```

3. Add the stub code on the next slide to the files

Subscriber - Stub

```
/*
 * Date Started: 09/03/2017
 * Original Author: Harry J.E Day
 * Editors:
 * ROS Node Name: sub_node
 * ROS Package: my_pubsub
 * Purpose: To demonstrate a simple subscriber node.
 * This code is released under the MIT  License. Copyright BLUEsat UNSW, 2017
 */

#include "my_pubsub/sub_node.hpp"

int main(int argc, char ** argv) {
    // starts ros, code will block here if roscore is not running
    ros::init(argc, argv, “sub_node");
    ROS_INFO("Hello Robot!");
    Sub_Node my_node;
    my_node.spin();

    return EXIT_SUCCESS;
}

Sub_Node::Sub_Node() {}

void Sub_Node::spin() {}

void Sub_Node::callback(const geometry_msgs::Twist::ConstPtr & msg) {}
```

```
/*
 * Date Started: 09/03/2017
 * Original Author: Harry J.E Day
 * Editors:
 * ROS Node Name: sub_node
 * ROS Package: my_pubsub
 * Purpose: To demonstrate a simple subscriber node.
 * This code is released under the MIT  License. Copyright BLUEsat UNSW, 2017
 */

// These import the main ros libary and the message we are going to recieve
#include <ros/ros.h>
#include <geometry_msgs/Twist.h>

class Sub_Node {
    public:
        Sub_Node();
        void spin();

   private:
       void callback(const geometry_msgs::Twist::ConstPtr & msg);
       ros::NodeHandle nh;
       ros::Subscriber sub;

};
```

Subscriber – The subscriber code

4. Check that everything compiles

```
$ cd  ~/tutorial-ws/
$ catkin_make
[ …lots of output goes here… ]
```

5. Setup the Node to Subscribe

```
Sub_Node::Sub_Node() {
    ros::TransportHints transport_hints = ros::TransportHints().tcp();
    sub = nh.subscribe<geometry_msgs::Twist>("/cmd_vel", 10, &Sub_Node::callback, this, transport_hints);
}
```

6. Setup the spin function

```
void Sub_Node::spin() {
   while(ros::ok()) {
      ros::spinOnce();
   }
}
```

Subscriber – The callback

- Runs every time the node receives a message
- Allows the node to handle the message

7. Implement the callback function

```
void Sub_Node::callback(const geometry_msgs::Twist::ConstPtr & msg)  {
   ROS_INFO("Received Message x: %f y: %f", msg->linear.x, msg->linear.y);
}
```

8. Compile the node

```
$ cd  ~/tutorial-ws/
$ catkin_make
[ …lots of output goes here… ]
```

A Basic Node – Compile and Run

9. Run the following to run

Terminal 1

```
$ roscore
[ …lots of output goes here… ]
```

Terminal 2

```
$ source devel/setup.bash
$ rosrun my_pubsub pub_node
[ INFO] [1489139498.792145514]: Hello Robot!
```

Terminal 3

```
$ source devel/setup.bash
$ rosrun my_pubsub sub_node
[ INFO] [1489139498.792145514]: Hello Robot!
[ INFO] [1489148966.245022580]: Received Message x: 1.000000 y: 0.000000
[ INFO] [1489148966.245160696]: Received Message x: 1.000000 y: 0.000000
```

Exercise

1. In groups modify the subscriber node to make the arm retract whenever the rover moves forward.

Hint: you'll need to add a publisher to the sub_node.

UNSW AUSTRALIA

Where to From Here?

A Map Definitely *not* Created by Our Rover

Never Stand Still | Engineering

More Information

- Talk to one of the senior members.
- ROS Tutorials: http://wiki.ros.org/ROS/Tutorials
- ROS Message Documentation: http://docs.ros.org/api/
  - Easier to google “[package name] [message name]” its usually the first result
- Software Section of the Wiki: https://bluesat.atlassian.net/wiki/display/OWRS/

What's on Confluence

- How To's
  - Add New Nodes to Catkin
  - How to use GDB with ROS
  - Building Gscam
  - Git Commands
  - Qt/QML Tutorial (also on our blog soon)
  - Documentation instructions

Other Seminars we are Running

- Confirmed
  - C++ (possibly in two parts)
  - Git
  - ROS Debugging Tools
- TBC
  - Qt/QML
  - eChronos

That's all Folks

UNSW AUSTRALIA

Never Stand Still | Engineering