Introduction to ROS
(continued)
COMP3431/COMP9434
Robot Software Architectures
Turtlebot Setup – Example

The Turtlebot’s Raspberry Pi processor is limited so we want to off-load as much processing as possible to an external workstation (or VM).

Turtlebot Waffle Pi
IP: 192.168.1.10

Workstation/VM
IP: 192.168.1.20
Turtlebot Setup – Step 1

Set ROS_MASTER and ROS_HOSTNAME for each computer.

Turtlebot Waffle Pi
IP: 192.168.1.10
ROS_MASTER_URI=192.168.1.20:11311
ROS_HOSTNAME=192.168.1.10

Workstation/VM
IP: 192.168.1.20
ROS_MASTER_URI=192.168.1.20:11311
ROS_HOSTNAME=192.168.1.20
Turtlebot Setup – Step 2

Spawn master in new terminal on workstation:

```bash
$ roscore
```

*`roscore` spawns master but also parameter server and logging outputs (not shown here).
**Turtlebot Setup – Step 3**

Run turtlebot startup in terminal on Joule:

```
$ roslaunch comp3431 turtlebot.launch
```

What this does:

- Spawns nodes to talk to hardware

---

Turtlebot Waffle Pi
IP: 192.168.1.10
ROS_MASTER_URI=192.168.1.20:11311
ROS_HOSTNAME=192.168.1.10

---

Workstation/VM
IP: 192.168.1.20
ROS_MASTER_URI=192.168.1.20:11311
ROS_HOSTNAME=192.168.1.20
Turtlebot Setup – Step 3

Run turtlebot startup in terminal on Joule:

```
$ roslaunch comp3431 turtlebot.launch
```

What this does:
- Spawns nodes to talk to hardware
- Nodes register with master

![Diagram of Turtlebot Setup](image)

**Turtlebot Waffle Pi**
- IP: 192.168.1.10
- ROS_MASTER_URI=192.168.1.20:11311
- ROS_HOSTNAME=192.168.1.10

**Workstation/VM**
- IP: 192.168.1.20
- ROS_MASTER_URI=192.168.1.20:11311
- ROS_HOSTNAME=192.168.1.20
Turtlebot Setup – Step 3

Run turtlebot startup in terminal on Joule:

```
$ roslaunch comp3431 turtlebot.launch
```

What this does:

- Spawns nodes to talk to hardware
- Nodes register with master
- `base` subscribes to `/cmd_vel` topic

**Turtlebot Waffle Pi**
IP: 192.168.1.10
ROS_MASTER_URI=192.168.1.20:11311
ROS_HOSTNAME=192.168.1.10

**Workstation/VM**
IP: 192.168.1.20
ROS_MASTER_URI=192.168.1.20:11311
ROS_HOSTNAME=192.168.1.20
Turtlebot Setup – Step 4

Run turtlebot teleop in workstation terminal:

$ roslaunch turtlebot_teleop keyboard_teleop.launch

What this does:

- Spawns node to listen to keyboard
Run turtlebot teleop in workstation terminal:

```
$ roslaunch turtlebot_teleop keyboard_teleop.launch
```

What this does:

- Spawns node to listen to keyboard
- Node registers with master

Turtlebot Waffle Pi
IP: 192.168.1.10
ROS_MASTER_URI=192.168.1.20:11311
ROS_HOSTNAME=192.168.1.10

Workstation/VM
IP: 192.168.1.20
ROS_MASTER_URI=192.168.1.20:11311
ROS_HOSTNAME=192.168.1.20
Turtlebot Setup – Step 4

Run turtlebot teleop in workstation terminal:

```
$ roslaunch turtlebot_teleop keyboard_teleop.launch
```

What this does:

- Spawns node to listen to keyboard
- Node registers with master
- `kbd_ctl` publishes to `/cmd_vel` topic
Frames of Reference

- ROS standardises the transformation model between different coordinate frames of reference.
- Right Hand Rule, X forward (XYZ ↔ RGB)
- Tree structure:
  - /map
    - /base_link
      - /base_footprint
      - /laser

- Example: laser detected object is relative to laser frame. Need to transform to map coordinate to know where it is on the map.
ROS Tools and Programs – 1

• Often first thing you run:
  
  ```
  $ roscore
  ```

  – Spawns ROS master – already explained
  – Creates a logging node (listening on topic `/rosout`).
  – Parameter server (http://wiki.ros.org/Parameter%20Server):
    • Shared dictionary for storing runtime parameters
    • Provides flexibility for storing configuration data
    • Hierarchical structure (don't confuse with topic names or frames).
    • Allows private names – configuration specific to a single node.
ROS Tools and Programs – 2

- What is the difference between `roslaunch` and `rosrun`?
- What is going on when I run:

  ```
  $ roslaunch comp3431 turtlebot.launch
  ```

  - If ROS_MASTER is local and no ROS master is running, then run `roscore`.

  - A weird mix of XML and shell scripting
  - ... let's look at `comp3431/launch/turtlebot.launch`
  - `node` tag in `includes/laser.launch` executes rosrun with appropriate parameters.

  ```
  $ rossrun lidar_node lidar_node _frame_id:="/lidar" ...
  ```

  - Note: the "_" - for private parameters.
ROS Tools and Programs – 3

• To debug the connections between nodes use:

  $ rqt_graph
  – Visualises the node graph – and topic connections

• Rviz is the main visualisation tool for ROS:

  $ rosrun rviz rviz
  – Provides plugins architecture for visualising different topics:

    • Videos
    • Map of environment and localised robot
    • Point cloud within the map

• Example: https://www.youtube.com/watch?v=25nnJ64ED5Q
ROS Tools and Programs – 4

• Possible to save the data produced by topics for later analysis and playback:
  
  $ rosbag record -a
  
  – Creates a time stamped bag file in the current directory.
  – Warning: “-a” records all topics so will generate a lot of data.

• Often useful to only record only direct sensor inputs (e.g., laser scans and timing) because the other topics will be generated from processing sensor data.

• To replay:

  $ rosbag play <bagfile>

• Useful if you are testing different interchangeable node (e.g., mapping with gmapping, hector SLAM, or Cartographer).

• Note: SLAM (Simultaneous Localisation and Mapping) algorithms build a map while at the same time localising. Very widely used in robotics.
ROS Tools – Simulator

- Two standard simulators; Stage (2D) and Gazebo (3D)
- For Turtlebot see: http://wiki.ros.org/turtlebot_simulator
- The Gazebo guide - easy guide to get simulator up and running.
- Follow the install instructions, then in different terminals run:

  $ roslaunch turtlebot_gazebo turtlebot_world.launch
  $ roslaunch turtlebot_teleop keyboard_teleop.launch
  $ roslaunch turtlebot_rviz_launchers view_robot.launch

- … see video
Many Different Sensors

- Laser Scanner
- Camera
- IR Cameras
- Depth Cameras
- Motor
- Pressure Sensor
- Compass
- Accelerometer
- IMU (Inertial Measurement Unit) – detects linear acceleration using accelerometer and rotation using gyroscope
- Audio

ROS provides standardised data structures for some of these sensors.
Laser Scanners

- A laser is rotated through a plane
- Distance (and intensity) measurements taken periodically
- 180-270 degrees

```
sensor_msgs/LaserScan

std_msgs/Header header
  uint32 seq
  time stamp
  string frame_id
  float32 angle_min
  float32 angle_max
  float32 angle_increment
  float32 time_increment
  float32 scan_time
  float32 range_min
  float32 range_max
  float32[] ranges
  float32[] intensities
```
Cameras

• Stream images
• Various encodings used (RGB, Mono, UYVY, Bayer)
• ROS has no conversion functions

#include <sensor_msgs/image_encodings.h>
Depth Cameras

- Usually produce Mono16 images
- Typically turned into point clouds
- Depth measurements can be radial or axial

sensor_msgs/PointCloud

std_msgs/Header header
  uint32 seq
  time stamp
  string frame_id

geometry_msgs/Point32[] points
  float32 x
  float32 y
  float32 z

sensor_msgs/ChannelFloat32[] channels
  string name
  float32[] values
Motor Positions

• Many motors report their positions

• Used to produce transformations between frames of reference

```
sensor_msgs/JointState
std_msgs/Header header
   uint32 seq
time stamp
   string frame_id
string[] name
float64[] position
float64[] velocity
float64[] effort
```
Lab Exercise

- Modify simple publisher and subscriber from Lecture 1:
  - Class member function callbacks.
  - Use Timer to publish at a specific rate.