ROB 456: Final Project Tips

In order to decide where to send the robot next to fully explore the world, you will need to:

1. Read the current map
2. Compute a waypoint to send to the robot (based on the current map)
3. Wait until the robot reaches the waypoint (or fails to reach the waypoint)
4. Repeat from 1 until all reachable grid cells are observed

The ROS topics that will be of interest to you are:

Topic name	Message type (Documentation)
/map	nav_msgs/OccupancyGrid (http://docs.ros.org/indigo/api/nav_msgs/html/msg/OccupancyGrid.html)
/base_link_goal AND/OR /map_goal	geometry_msgs/Twist (http://docs.ros.org/indigo/api/geometry_msgs/html/msg/Twist.html)
/move_base/result	move_base_msgs/MoveBaseActionResult (http://docs.ros.org/fuerte/api/move_base_msgs/html/msg/MoveBaseActionResult.html)

How to use the documentation

Each of the links will take you to a webpage that looks something like this:

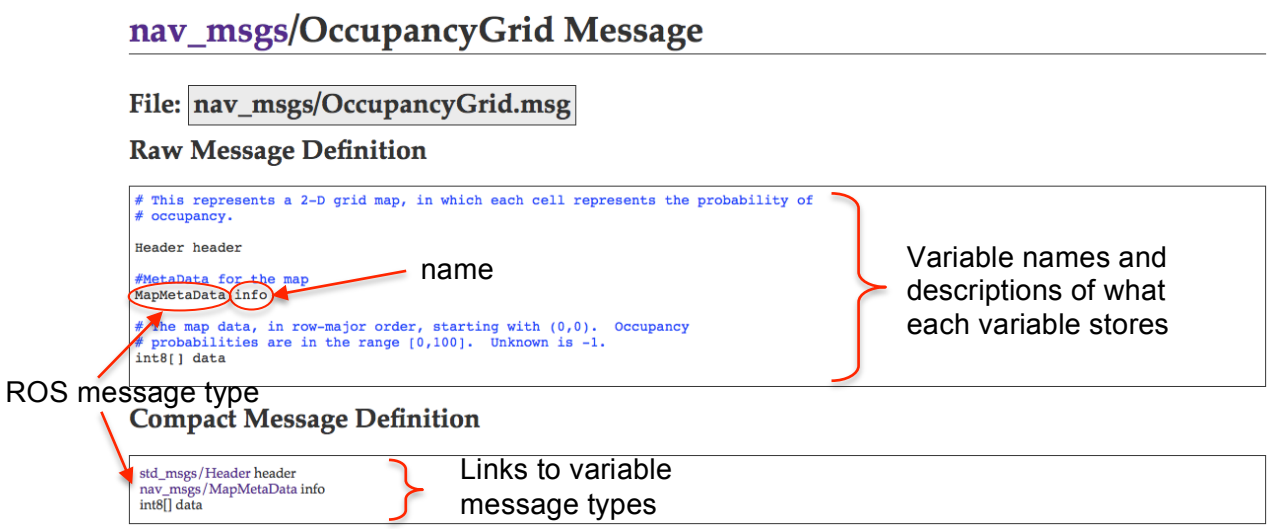
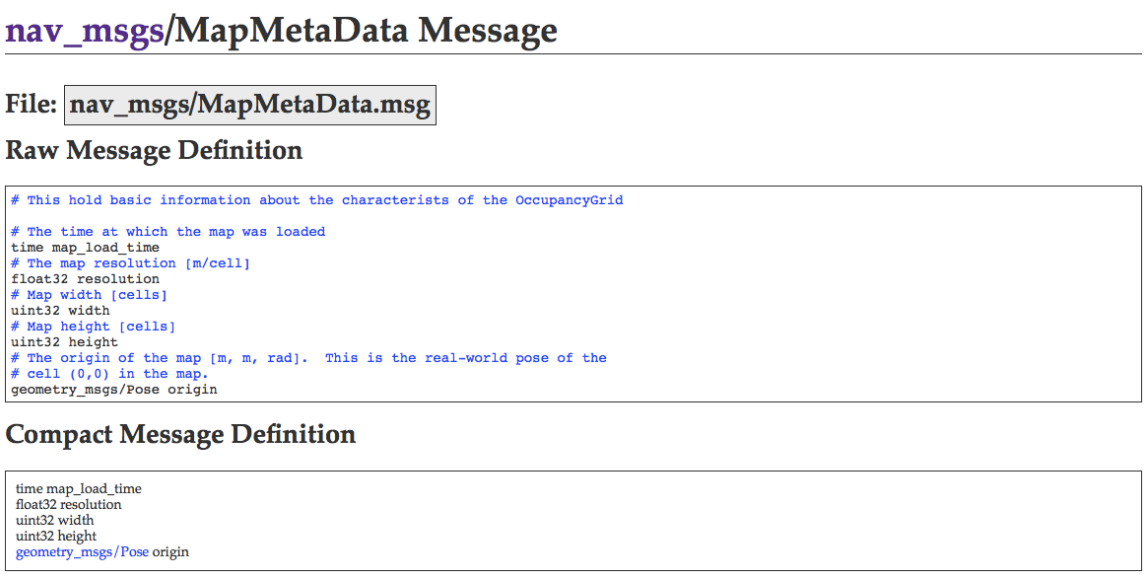


Figure 1 The nav_msgs/OccupancyGrid message definition

The compact message definition provides links when a variable inside a message is itself a ROS message. In the above example, both “header” and “info” are ROS message types and so contain links to the corresponding documentation. In comparison, “data” is an array of integers, which is a basic data type that you can interact with as usual in your code. For example, if I read in the /map topic as a variable called “msg” in Python, then msg.data[0] will return the first element in the array “data”.

Clicking through the nav_msgs/MapMetaData link will take you to a webpage that looks like this:



nav_msgs/MapMetaData Message

File: nav_msgs/MapMetaData.msg

Raw Message Definition

```
# This hold basic information about the characterists of the OccupancyGrid

# The time at which the map was loaded
time map_load_time
# The map resolution [m/cell]
float32 resolution
# Map width [cells]
uint32 width
# Map height [cells]
uint32 height
# The origin of the map [m, m, rad].  This is the real-world pose of the
# cell (0,0) in the map.
geometry_msgs/Pose origin
```

Compact Message Definition

```
time map_load_time
float32 resolution
uint32 width
uint32 height
geometry_msgs/Pose origin
```

Figure 2 The nav_msgs/MapMetaData message definition

Following from the previous example, if you want to access the resolution of the /map topic, in Python you would call msg.info.resolution, which would return a floating point number. For more examples, see how we accessed the LaserScan, Odometry and Twist messages in HW2 and HW3.

Reading the map

Notice that the map data is stored in a 1D array. In Figure 1 we see the comment:
"The map data, in row-major order, starting with (0,0). Occupancy probabilities are in the range [0,100]. Unknown is -1."
This means that to interpret the map as a 2D matrix, we will need to use an element's index in the array, work out what row it would sit on in the matrix, and then what column it belongs to on that row.

For example, map.data may contain:

map.data =

0	1	2	3	4	5	6	7	8

But to work out neighboring cells in the real world, we would like to read it as:

my_map =

0	1	2
3	4	5
6	7	8

Here we can employ integer division and modulo operations to convert from an index in the 1D array to an index in 2D matrix.

The variable map.info.width, tells us how many cells are on each row. Therefore, we can integer divide the element index, *i*, by this value to work out the row number, *r*:

```
r = i / map.info.width
```

Similarly, we can use modulo to work out the column number, *c*:

```
c = i % map.info.width
```

For the example above, our map width is 3. So for element number 5 in map.data, that gets converted to:

```
r = 5 / 3 = 1
c = 5 % 3 = 2
```

And we can confirm in the table above that element 5 corresponds to my_map[1][2].

The second thing to be aware of is that the real-world location of map.data[0] is stored in map.info.origin. Remember this is a Pose message type, so to get to the actual x, y location, you have to go down a couple more levels:

```
x_origin = map.info.origin.position.x
y_origin = map.info.origin.position.y
z_origin = map.info.origin.position.z
```

The orientation of the map (in quaternions) can be accessed in a similar way:

```
qx_origin = map.info.origin.orientation.x
qy_origin = map.info.origin.orientation.y
qz_origin = map.info.origin.orientation.z
qw_origin = map.info.origin.orientation.w
```

Along with the map resolution these will allow you to convert from a cell coordinate (r,c) to an (x,y) location in the world that can be sent as a waypoint to the robot.