Pollution Mapping with the MMP

Tasks completed

- `slam_gmapping`, `amcl`, and `move_base` are working on the MMP
- basic pollution mapping from gas sensor data

Things to be fixed

- `move_base` parameters need to be tweaked in order to produce faster goal achievement
- Chrony needs to be installed to permanently fix timing issues between the MMP and workstation computer
- gas sensors do not produce reliable results (possible change in resistor values)

Connecting to the MMP

The MMP's computer must be accessed via ssh from a workstation computer. It is connected to the workstation through a wireless network, provided by the base station.

1. Change the workstation computer's ROS master to the MMP's computer (the MMP IP may be 192.168.0.105 or 192.168.0.104).
   Temporarily (for the single terminal):
   ```
   $ export ROS_MASTER_URI=http://192.168.0.105:11311
   ```
   Permanently:
   ```
   $ echo export ROS_MASTER_URI=http://192.168.0.105:11311 >>
     ~/.bashrc
   ```
2. Leave the `ROS_HOSTNAME` for the workstation as the workstation IP.
3. Turn on the base station and connect to the CPRTEC wireless network. Ping the MMP's computer to confirm the connection.
   ```
   $ ping 192.168.0.105
   ```
4. You may need to add a route from the workstation to the robot.
   ```
   $ sudo route add -net 192.168.0.105 netmask 255.255.255.255
     gateway 192.168.0.105
   ```
5. Communicate with the MMP via the workstation.
   ```
   $ ssh administrator@192.168.0.105
   ```

- Note that transferring files between the MMP and the workstation works the same way. The connection is opened by:
  ```
  $ sftp administrator@192.168.0.105
  ```
  Files are transferred by using the `get` or `put` commands.

Teleoperation

The MMP joystick has a USB component (stored in the battery compartment) which must be plugged into the MMP. The controller buttons correspond as follows:

- **X**: Toggles the headlights.
- **A**: Enables wheel movement. Must be held down during teleoperation of MMP.
- **B**: Enables autonomous movement. Must be held down to allow `move_base` to move the robot.
- **Left joystick**: Controls movement of robot.

Where to run your nodes

Most of the necessary background MMP nodes start up when the MMP is turned on. Some of these may conflict with other nodes that you choose to run. For example, `ekf_localization` starts up automatically, but if you would like to use `robot_pose_ekf` instead, ensure that you kill `ekf_localization` first.

The MMP's computer cannot handle several nodes running at once before the CPU stalls. If the CPU stalls, published messages will not be output at their expected frequencies. This results in extrapolation errors caused by mismatched timestamps between ROS messages. As such, the navigation stack should be run on the workstation, while the nodes which require direct access to the MMP's hardware should be run on the MMP.

MMP	**Workstation**
`teleop`	`slam_gmapping`
`ekf_localization / robot_pose_ekf`	`amcl`
`serial_node`	`pmapping`
	`rviz`

Note that, while `ekf_localization` starts up with the MMP, both scouts have been set up to use `robot_pose_ekf`. In order to set up the MMPs with `ekf_localization`, the odometry frames have to be changed in the include files for `gmapping` and `amcl`, and topics have to be remapped for `move_base` (`robot_pose_ekf` uses `odom_combined` while `ekf_localization` uses `odom`).

Starting up the MMP

1. Kill the `ekf_localization` node.
   ```
   $ rosnode kill ekf_localization
   ```
   You can check that this was successful by viewing a list of running nodes.
   ```
   $ rosnode list
   ```
2. If Chrony has not been installed, match the time between the MMP and the workstation from the MMP's computer.
   ```
   $ sudo ntpdate <IP address of computer>
   ```
   If you do not know the IP address, check via the following command on the workstation computer:
   ```
   $ ifconfig
   ```
3. Launch teleoperation of the MMP.
   ```
   $ roslaunch tec_scout_bringup teleop.launch
   ```
4. Launch `robot_pose_ekf`. Note that this provides the transform between the odometry frame (`odom_combined`) and the `base_footprint` frame of the robot. Additionally, it publishes a pose estimation by filtering data from the wheel encoders, IMU, and GPS (not configured) through a Kalmann filter.
   ```
   $ roslaunch tec_scout_bringup ekf.launch
   ```

Connecting to the Arduino

1. Before connecting the Arduino, check which port the Arduino is connected to by:
   ```
   $ cd /dev
   $ ls tty*
   ```
   This should output a list of files. Plug in the Arduino and check this list again. The difference in the lists is the port with the newly attached Arduino. This should be `ttyACM1` or `ttyACM2`. The `ttyACM0` port is connected to another MMP Arduino responsible for the encoder output.
2. Launch `rosserial` to allow the Arduino to publish to topics. The Arduino should be programmed to initialize a node and a subscriber, read gas sensor output, scale the data, and publish the data to the "`MQ`" topic.
   ```
   $ rosrun rosserial_python serial_node.py /dev/<Arduino port>
   ```

- For the Arduino code, see *ROS with Arduino* below.

Pollution mapping

Both the `gmapping` and the `amcl` launch files also run the `move_base` node. The pollution mapping node works by taking the ROS PoseWithCovarianceStamped message published by `robot_pose_ekf`, the map published by `gmapping`, and the gas sensor data from the Arduino and forming a corresponding gas map (array in row-major order) filled with the sensor data scaled to size. These values are then converted to colour values in an image which displays both the pollution sensor readings and the map obstacles.

1. Launch `slam_gmapping` or `amcl` in order to provide the map for pollution mapping.
   ```
   $ roslaunch cear_jenni gmapping_mmp.launch
   ```
   or
   ```
   $ roslaunch cear_jenni amcl_mmp.launch
   ```
2. Launch rviz in order to view the laserscans, obstacle map, and pollution map.
   ```
   $ roslaunch cear_jenni view_mmp.launch
   ```
3. If the gas sensors are not working, use fake sensor data (published to topic "`MQ`").
   ```
   $ rosrun cear_jenni tmp_mq
   ```
4. Launch pollution mapping node.
   ```
   $ rosrun cear_jenni pmapping
   ```

- Note that you must press the 'B' button on the joystick in order to allow `move_base` to move the robot. This means teleoperation must be running.
- You can set a `move_base` goal via the command line, but you must know the map coordinates of the goal in order to set it.
  ```
  $ rostopic pub /move_base_simple/goal geometry_msgs/PoseStamped
    '{ header: { frame_id: "map" }, pose: { position: { x: 1.0, y:
    0, z: 0 }, orientation: { x: 0, y: 0, z: 0, w: 1 } } }'
  ```
  For this reason, you should set the `2d Nav Goal` in rviz by point and click. This determines the coordinates of the PoseStamped message goal and publishes them to `/move_base_simple/goal` without command line usage.

Gas Sensors

There are 5 different types of gas sensors which have already been soldered and burned in.

MQ	Gas	Quantity
2	flammable gas, smoke	1
3	alcohol	1
6	propane, LPG, isobutene	1
7	carbon monoxide	2
8	hydrogen	1

The datasheets for the sensors have been included with the `cear_jenni` package. The wiring for these sensors can be found here and has already been done. Note that one of the modules was bought preassembled (MQ7) and uses a variable resistor. You must desolder the other modules in order to change the resistor values.

ROS with Arduino

In order for ROS to work with Arduino, the `rosserial` package must be downloaded and the `ros_lib` library must be imported to Arduino.

1. Install `rosserial`.
   ```
   $ sudo apt-get install ros-hydro-rosserial-arduino
   $ sudo apt-get install ros-hydro-rosserial
   ```
   You can also `git clone` the package from here.
2. Create the `ros_lib` library.
   ```
   $ cd <Arduino sketchbook directory>/libraries
   $ rosrun rosserial_arduino make_libraries.py
   ```
   If the library already exists, and has to be regenerated, you must first remove the `ros_lib` folder before running `make_libraries.py`.
3. Import the library to the Arduino IDE.

After importing the library, you must include `ros.h` as well as any message types used in the code (see lines 1 and 2). ROS node and publisher initialization works in much the same way as with C++ scripts.

```
1   #include <ros.h>
2   #include <std_msgs/Int64.h>
3
4   ros::NodeHandle nh;
5   std_msgs::Int64 input;
6
7   // Create publisher
8   ros::Publisher MQ("MQ", &input);
9
10  void setup() {
11
12    // Initialize node and publisher
13    nh.initNode();
14    nh.advertise(MQ);
15  }
16
17  void loop() {
18    // Scale and store data
19    input.data = (float)analogRead(A0)*127.0/1023.0;
20
```

```
   // Publish data
   MQ.publish(&input);
   nh.spinOnce();
   delay(10);
}
```

Errors and Debugging

The following extrapolation error typically occurs when `robot_pose_ekf` is not functioning properly and the transformation between the odometry frame and `base_footprint` is not being provided.

```
[ WARN] [1436263397.911729770]: Costmap2DROS transform timeout. Current time: 1436263397.9095, global_pose stamp: 1436263388.0972, tolerance: 1.0000
[ERROR] [1436263398.192083647]: Extrapolation Error looking up robot pose: Lookup would require extrapolation into the past. Requested time 1436263388.097209750 but the earliest data is at time 1436263388.178999652, when looking up transform from frame [base_footprint] to frame [map]
```

This could be caused by a network timing error (especially if the extrapolation error is into the future). You should ensure clock synchronicity between computers:

```
$ sudo ntpdate <IP address of computer>
```

It is more likely that this is caused by a problem with `robot_pose_ekf`. You can verify this by reviewing your current transform tree.

```
$ cd <directory for storage of tf tree pdf file>
$ rosrun tf view_frames
```

If your transform tree is missing the transformation from the odometry frame to `base_footprint` then you know that `robot_pose_ekf` is not working correctly. Check to see that the IMU and encoder have both been activated, and the Kalmann filter has been initialized. You can also check the encoder by reading the messages published by the `/encoder` topic.

```
$ rostopic echo /encoder
```

If no messages appear, the encoder is not activated. The encoder sometimes does not start (especially when additional Arduinos have been connected) and the robot may need to be restarted.

This error can also occur when the computer does not have the processing power to run all the nodes at once and large delays occur between published messages. See *Where to run your nodes* for more information. You can check the frame transformation frequency via the following:

```
$ rosrun tf tf_monitor
```

The following warning occurs when the robot cannot update or publish at the expected frequency. These rates are set in the `move_base` parameters found in the 'config' directory of `cear_jenni`. If this warning occurs frequently, then the expected frequencies are too high for the computer to handle and can affect the performance of the robot.

```
[ WARN] [1436263590.879563461]: Map update loop missed its desired rate of 4.0000Hz... the loop actually took 0.4396 seconds
[ WARN] [1436263590.885458972]: Control loop missed its desired rate
```

```
of 5.0000Hz... the loop actually took 0.5562 seconds
```

This warning occurs when `move_base` does not know the current pose of the robot. The problem typically originates with `robot_pose_ekf`. See the previous errors for more information.

```
[ WARN] [1436263592.342165723]: Unable to get starting pose of robot,
unable to create global plan
```

This error appears when networking between computers is not working correctly.

```
[ERROR] [1436263398.192083647]: Couldn’t find an AF_INET address for …
```

It can be fixed by adding the name and IP address of the other computer to the hosts file.

```
$ cd etc
$ sudo nano hosts
```

Add the IP address and name under the current computers home address and name.