ROS indigo and Gazebo2
Software-in-the-Loop Pioneer3dx Simulation
Ubuntu 14.04 LTS (Trusty Tahr)

Jen Jen Chung

May 6, 2016

Abstract

This document outlines the setup of the software-in-the-loop simulation for the Pioneer-3dx robots. This setup incorporates Gazebo2 as the physics simulator on a base station computer and uses a ROS concert framework for communication between the base station and the individual laptops controlling each robot. This setup has been tested on Ubuntu 14.04 LTS (Trusty Tahr) machines with the ROS indigo distribution, which interfaces with Gazebo2. Future ROS distributions may require different versions of Gazebo, users are encouraged to check online for the latest information at http://wiki.ros.org and http://gazebosim.org.

1 Installation

For ROS and Gazebo2 installation instructions see the relevant online documentation at http://wiki.ros.org/indigo/Installation/Ubuntu and http://gazebosim.org/tutorials?tut=install&ver=2.2&cat=install. A compiled set of instructions including details for downloading custom ROS packages specific to this simulation can be found at http://web.engr.oregonstate.edu/~chungje/Code/Pioneer3dx%20simulation/ros-indigo-gazebo2-pioneer.pdf.

Standard ROS packages that will also need to be installed include:

```
sudo apt-get install ros-indigo-navigation ros-indigo-gmapping ros-indigo-slam-gmapping ros-indigo-ros-
    control ros-indigo-ros-controllers ros-indigo-rviz ros-indigo-rocon*
```

1.1 Simulation Packages

Custom packages required for robot localisation, waypoint navigation and system networking:

- **pioneer_gazebo_ros**: Gazebo simulation nodes, includes robot model, world files and scripts for launching nodes.
- **pioneer_2dnav**: All **move_base** configuration files.
- **nav_bundle**: Launch files for localisation and navigation.
- **simple_navigation_goals**: Nodes for waypoint navigation, interfaces with **move_base**.
- **rocon_flip**: Scripts for advertising/pulling and flipping topic connections in a robotics-in-concert framework.
- **pioneer_networking**: Launch files and scripts for setting up robotics-in-concert framework.

The **pioneer_networking** package can be found at https://github.com/duchowpt, all other packages can be found at https://github.com/JenJenChung. Git clone these repositories into your ~/catkin_ws/src folder and run `catkin_make`.

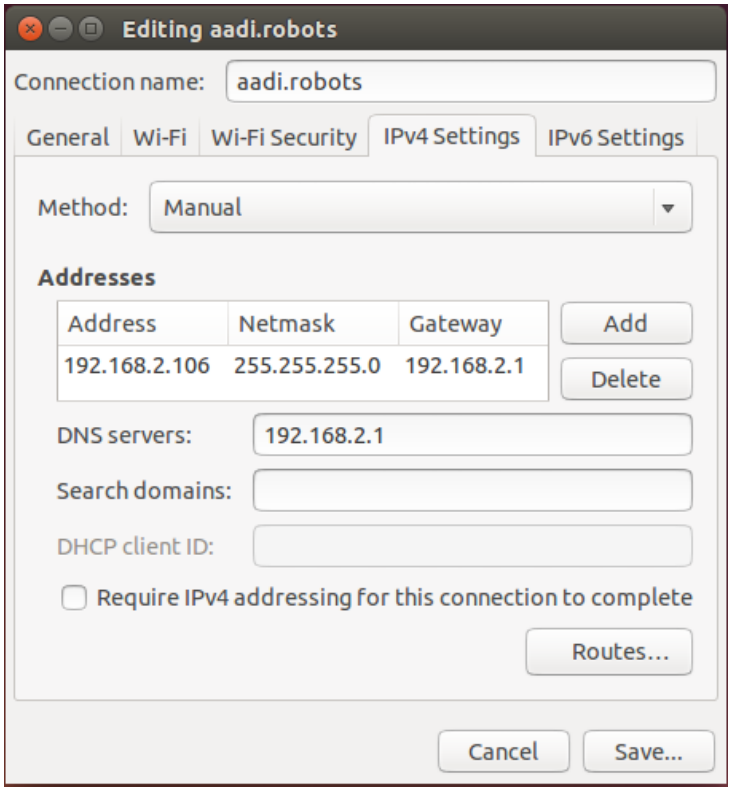

Figure 1: How to set up a static IP on the aadi.robots network.

2 Network Setup

The network consists of a designated base station computer running the Gazebo physics simulator. The base station will communicate relevant state information to the remote computers that are assigned to run localisation and control algorithms for the individual robots within the simulation. The remote computers communicate the commanded control actions to the base station computer for execution in the Gazebo simulation. In addition, the remote computers may also communication relevant topics for visualisation in rviz to the base station computer, such as corrected pose estimate (/tf output of the **amcl** package), current goal and planned trajectory.

2.1 Wireless Network

A private WiFi network **aadi.robots** is set up for use with the AADI laptops. The Gazebo simulation has a conservative estimated memory requirement of 1GB per additional robot and all network traffic will be directly between the remote computers and the base station. Thus, it is recommended that you use the aadiworkstation computer as the base station with a direct ethernet connection to the aadi.robots router.

All computers in the network must have a static IP, you can edit this by navigating to **All Settings→Network→aadi.robots→Settings→IPv4 Settings**, selecting **Manual** for the method and entering the relevant **Address** details, see Fig. 1. Currently, each AADI laptop address is 192.168.2.1XX, where XX is replaced by the laptop number (e.g. 06 for aadi6) and the workstation computer IP is 192.168.2.2.

Enter the static IP addresses of each machine along with the associated machine name in the /etc/hosts file of each network computer. This can be done by typing `nano /etc/hosts` in a terminal and directly inserting the relevant details after the local host and home addresses, see Fig. 2.

2.2 Robotics-in-Concert (rocon)

The ROS communication network is set up using the robotics-in-concert multimaster framework, sparse documentation can be found via the ROS wiki `http://wiki.ros.org/rocon` and a number of tutorials

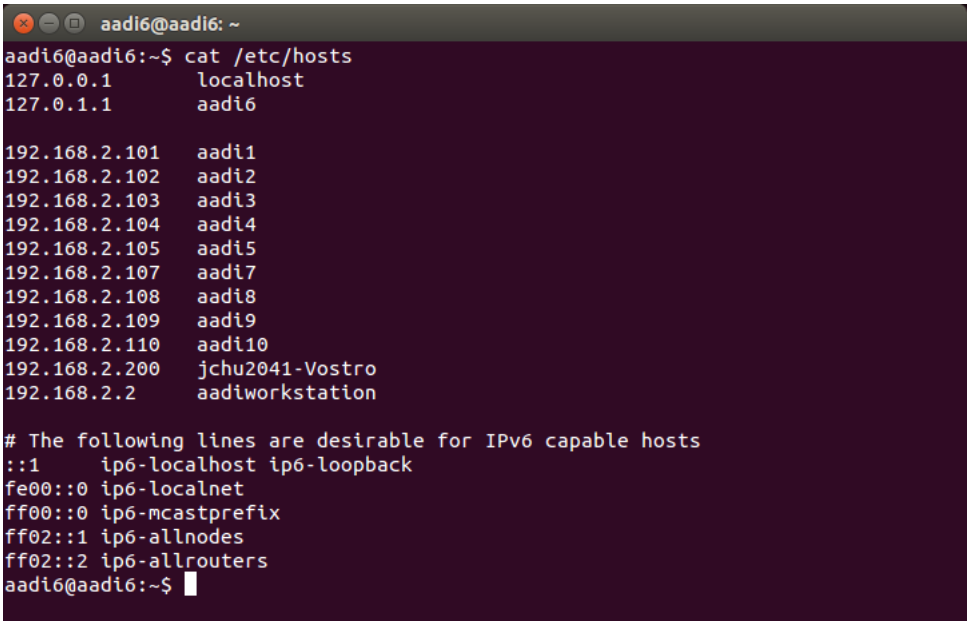

Figure 2: Including networked computers in the /etc/hosts file.

are available at http://wiki.ros.org/rocon_tutorials/Tutorials.

The concert framework requires a *Hub* that hosts the concert and allows the creation and registration of *gateways* from which individual ROS masters can access the concert network. Critically, the Hub itself does not have access to the data available in the network, all traffic within the network must originate from a gateway connection. The scripts in the **rocon_flip** package assume that the Hub will be instantiated on the base station computer from which the Gazebo simulation will also be executed.

In the **pioneer_networking** package, the **run-pioneer-hub** script launches a multimaster concert hosted by Pioneer Hub and initialises a gateway called Hub_Gateway from two separate ports, each with their own ROS masters. Executing the **pioneer_gateway.launch** file from any laptop connected to the same wireless network as the Hub will create a new gateway with its own ROS master on that laptop, and will register the gateway on Pioneer Hub. The gateway will be called aadiX_gateway, where X is replaced by the laptop number.

2.3 Communicating ROS Topics

Any ROS nodes that require incoming or outgoing topics that are discoverable on the network must be executed under the ROS master associated with the local gateway. This means that the Hub_Gateway ROS master will manage the Gazebo simulation, while the aadiX_gateway ROS masters will each run the localisation and navigation nodes.

There are two connection methods for communicating topics across the network: advertising/pulling and flipping. Advertising and pulling is similar to the way that the ROS publish/subscribe framework operates, whereby any node on a connected gateway can advertise (publish) a topic on the network and any node on a connected gateway can pull (subscribe to) a topic on the network. In comparison, a flip connection is a direct request to share a topic between two nodes on different gateways. Either gateway can instantiate the flip connection, regardless of whether the local node is the subscriber or the publisher. Setting up the connection requires the name of the remote gateway, the topic name, the name of the local node, and specification of whether the local node is subscribing to or publishing the topic. Note that the local node can be specified as an empty string, this allows the connected topic to be visible to/from all nodes on the local gateway.

Listings 1 and 2 provide an example of how to advertise and pull a topic connection. Both callback functions take a single string input for the topic name. Note that the advertiser does not need to specify a remote gateway since it is advertising to the entire network. On the other hand, the pull request must specify the gateway from which the advertisement originates. In both scripts, the local node (`rule.node`) is specified as an empty string. For the advertiser, this means that if multiple nodes are publishing that topic, the most recent message will always be advertised. If a specific node is listed in the rule, then

only the most recently published topic message from that node will be advertised. Similarly, the pull request can specify a single node that can access the pulled topic by entering a non-empty string in the `rule.node` variable. The purpose of this is to prevent data contamination if multiple topics on the network have the same name.

```
#!/usr/bin/python
import rospy
import sys
import gateway_msgs.msg as gateway_msgs
import gateway_msgs.srv as gateway_srvs
import rocon_gateway

def advertise(topic):
  rospy.init_node('advertise_hub_publisher')

  rospy.wait_for_service('/gateway/advertise')
  advertise_service=rospy.ServiceProxy('/gateway/advertise', gateway_srvs.Advertise)
  req = gateway_srvs.AdvertiseRequest()
  rule = gateway_msgs.Rule()
  rule.name = '/' + topic
  rule.type = gateway_msgs.ConnectionType.PUBLISHER
  rule.node = ""
  req.rules.append(rule)
  rospy.loginfo("Advertise : [%s,%s,%s]."%(rule.name, rule.type, rule.node))

  resp = advertise_service(req)
  if resp.result != 0:
      rospy.logerr("Advertise : %s"%resp.error_message)

  rospy.loginfo("Finished advertising connection.")

if __name__=='__main__':
  try:
    if len(sys.argv) < 2:
      print('Usage: advertise_hub_publisher.py topic')
    else:
      advertise(str(sys.argv[1]))
  except rospy.ROSInterruptException:
    pass
```

Listing 1: advertise_hub_publisher.py: Python script for advertising a topic connection

```
#!/usr/bin/python
import rospy
import sys
import gateway_msgs.msg as gateway_msgs
import gateway_msgs.srv as gateway_srvs
import rocon_gateway

def pull(topic):
  rospy.init_node('pull_hub_publisher')

  rospy.wait_for_service('/gateway/pull')
  remote_gateway = "Hub_Gateway"
  pull_service = rospy.ServiceProxy('/gateway/pull', gateway_srvs.Remote)
  req = gateway_srvs.RemoteRequest()
  req.cancel = False
  req.remotes = []
  rule = gateway_msgs.Rule()
  rule.name = '/' + topic
  rule.type = gateway_msgs.ConnectionType.PUBLISHER
  rule.node = ""
  req.remotes.append(gateway_msgs.RemoteRule(remote_gateway, rule))
  rospy.loginfo("Pull : [%s,%s,%s,%s]."%(remote_gateway, rule.name, rule.type, rule.node))

  resp = pull_service(req)
  if resp.result != 0:
      rospy.logerr("Pull : %s"%resp.error_message)

  rospy.loginfo("Finished pulling connection.")

if __name__=='__main__':
```

```
  try:
    if len(sys.argv) < 2:
      print('Usage: pull_hub_publisher.py topic')
    else:
      pull(str(sys.argv[1]))
  except rospy.ROSInterruptException:
    pass
```

Listing 2: pull_hub_publisher.py: Python script for pulling a topic connection

Listing 3 provides a script to flip a direct topic connection between two gateways. This particular example is a flip request from the Hub_Gateway to an aadiX_gateway, where the local node on the Hub_Gateway is subscribing to a topic published by the remote gateway. The flip connection setup is similar to that of a pull connection.

```
#!/usr/bin/python
import rospy
import sys
import gateway_msgs.msg as gateway_msgs
import gateway_msgs.srv as gateway_srvs
import rocon_gateway

def flip(namespace, topic, node, gate):
  rospy.init_node('flip_hub_subscriber')

  remote_gateway = gate + '_gateway'
  flip_service = rospy.ServiceProxy('/gateway/flip', gateway_srvs.Remote)
  req = gateway_srvs.RemoteRequest()
  req.cancel = False
  req.remotes = []
  rule = gateway_msgs.Rule()
  if topic[0] == '/':
    rule.name = topic
  else:
    rule.name = '/' + namespace + '/' + topic
  rule.type = gateway_msgs.ConnectionType.SUBSCRIBER
  rule.node = '/' + node
  req.remotes.append(gateway_msgs.RemoteRule(remote_gateway,rule))
  rospy.loginfo("Flip : [%s,%s,%s,%s]."%(remote_gateway, rule.name, rule.type, rule.node))

  resp = flip_service(req)
  if resp.result != 0:
      rospy.logerr("Flip : %s"%resp.error_message)

  rospy.loginfo("Finished flipping connection.")

if __name__=='__main__':
  try:
    if len(sys.argv) < 5:
      print('Usage: flip_hub_subscriber.py namespace topic node gate')
    else:
      flip(str(sys.argv[1]),str(sys.argv[2]),str(sys.argv[3]),str(sys.argv[4]))
  except rospy.ROSInterruptException:
    pass
```

Listing 3: flip_hub_subscriber.py: Python script for flipping a topic connection

Once a connection rule is set up from a gateway, it will remain active until the associated gateway is disconnected from the network. An error message will appear if you attempt to create a new rule that is identical to an existing rule on the same gateway. The new rule will not be created and the existing rule will continue to operate. Note that a common source of user error derives from typographical mistakes in the `remote_gateway`, `rule.name`, and `rule.node` variables, make sure to check these first when debugging connection failures.

3 How To Run Basic Software-in-the-Loop Simulation

The steps for running the basic software-in-the-loop simulation code is provided in pioneer_gazebo_ros/ pioneer_gazebo/how_to_launch_rocon_multi_pioneer.txt and is repeated below:

0. Directly connect the Hub computer to the aadi.robots router via ethernet and make sure to use the aadi.robots connection settings. Make sure all remote laptops are also connected to the aadi.robots WiFi network.

1. Hub computer (designated to run Gazebo physics simulation)

```
rosrun pioneer_networking run-pioneer-hub
```

2. Remote computers (designated to run Pioneer navigation)

```
export HOSTNAME
roslaunch pioneer_networking pioneer_gateway.launch
```

3. Hub gateway terminal (ctrl+shift+t)

```
rosrun pioneer_gazebo run-multi-pioneer-gazebo-master
```

4. Remote computers (ctrl+shift+t)

   If necessary edit ROBOT_NAME variable in run-pioneer-nav-local line 6 to reflect pioneer number, currently should match aadi computer number

```
rosrun pioneer_gazebo run-pioneer-nav-local
```

5. Hub gateway terminal (ctrl+shift+t)

```
roslaunch pioneer_description multi_pioneer_rviz.launch
```

3.1 The Steps Explained

3.1.1 Hub Concert Initialisation

The run-pioneer-hub script initialises a hub concert under the name “Pioneer Hub” in a new terminal. In a separate terminal with a new ROS master, it creates a gateway under the name “Hub_Gateway”.

3.1.2 Launching Gateways

The HOSTNAME environment variable is used by the remote computers to define the gateway name and so must be exported to the current terminal. The pioneer_gateway.launch file creates a local gateway under the name “aadiX_gateway” where X is replaced by the laptop number. The gateway should connect to the Pioneer Hub if both computers are all on the same network, check that a successful connection is reported in the terminal from which the launch file was called.

3.1.3 Launching Gazebo Simulation

New terminals opened from the gateway terminal (ctrl+shift+t) will be connected to the gateway ROS master. The run-multi-pioneer-gazebo-master script shown in Listing 4 launches the Gazebo world (multi_pioneer_world.launch), loads the robot URDF model including the initial positions of each robot (robot_description.launch), and runs a loop to initialise each robot under the appropriate namespace (generic_pioneer.launch). The map is also loaded into the Hub_Gateway ROS master since it will be shared across all robots. Make sure that the loaded map file (e.g. utm_0) matches the Gazebo world that is launched. Finally, the script advertise-from-hub advertises the simulation topics that must be shared across the network (clock, map, tf, tf_static), and flip-from-hub creates a flip subscription request for localisation and navigation topics from each remote gateway to enable visualisation in rviz on the workstation computer (see Listing 5).

```
#!/bin/bash

PATH=$(echo $PATH | sed 's|/home/aadi-workstation/anaconda3/bin:||g')
export PATH=$PATH

my_pid=$$
echo "My process ID is $my_pid"

echo "Launching Gazebo..."
roslaunch pioneer_gazebo multi_pioneer_world.launch &
pid=$!

echo "Loading initialisation parameters..."
sleep 10s
roslaunch pioneer_description robot_description.launch

echo "Launching Pioneers in Gazebo stack..."
sleep 5s
for i in `seq 1 5`;
do
  roslaunch pioneer_description generic_pioneer.launch name:=pioneer$i pose:="-x $(rosparam get /pioneer$i
      /x) -y $(rosparam get /pioneer$i/y) -Y $(rosparam get /pioneer$i/a)" &
  pid="$pid $!"
  sleep 7s
done

echo "Launching map server..."
sleep 5s
roslaunch nav_bundle map_server.launch map_name:=utm_0 &
pid="$pid $!"

echo "Advertising topics..."
sleep 5s
rosrun rocon_flip advertise-from-hub

echo "Flipping connections to rviz..."
rosrun rocon_flip flip-from-hub

trap "echo Killing all processes.; kill -2 TERM $pid; exit" SIGINT SIGTERM

sleep 24h
```

Listing 4: run-multi-pioneer-gazebo-master: Script to run Gazebo simulation on the Hub Gateway

```
#!/bin/bash

namespace=(pioneer1 pioneer2 pioneer3 pioneer4 pioneer5)
remotes=(aadi1 aadi2 aadi3 aadi4 aadi5)
sub_topics=(/tf move_base/current_goal move_base/local_costmap/costmap move_base/local_costmap/
     costmap_updates move_base/NavfnROS/plan move_base/TrajectoryPlannerROS/local_plan)
sub_nodes=(rviz)
sub_robot_len=${#namespace[@]}
sub_topic_len=${#sub_topics[@]}

for (( i=0; i<${sub_robot_len}; i++ )); do
  for (( j=0; j<${sub_topic_len}; j++ )); do
    rosrun rocon_flip flip_hub_subscriber.py ${namespace[$i]} ${sub_topics[$j]} ${sub_nodes[0]} ${remotes[
        $i]}
  done
done
```

Listing 5: flip-from-hub: Script to flip subscription connection request to remote gateways

3.1.4 Launching Navigation

The run-pioneer-nav-local script sets up the necessary topic connections across the network and runs the navigation launch file. Only run this on the laptops after the Hub_Gateway script has successfully completed on the base station computer. The ROBOT_NAME environment variable should be listed as "pioneerX" where X is replaced by the laptop number, if you are assigning mismatching laptop numbers to the robots, ensure that you make the appropriate changes to the ROBOT_NAME variable.

Each remote computer first pulls the required advertised topics from the Hub_Gateway that will enable synchronisation at initialisation for the **amcl** and **move_base** packages (pull-from-hub). The flip-from-remote script sets up subscription connections to base_scan and odom from the simulator as well as a publisher connection to send locally computed pioneer/cmd_vel messages to the simulator. The topic connections must be established for the navigation bundle to launch successfully. The base_scan and odom messages may take some time to communicate across the network and timeout warnings may appear; however, once they are received you will see "`odom received!`" appear as the final message printed to the terminal.

```
#!/bin/bash

my_pid=$$
echo "My process ID is $my_pid"

export ROBOT_NAME="pioneer"

echo "Pulling advertised topics..."
rosrun rocon_flip pull-from-hub

echo "Flipping topics from local machine..."
rosrun rocon_flip flip-from-remote

echo "Loading robot parameters..."
roslaunch pioneer_description robot_description.launch
rosparam set /use_sim_time true

echo "Launching robot navigation packages..."
namespace=($ROBOT_NAME)
roslaunch nav_bundle single_navigation.launch robot_name:=${namespace[0]} x:="$(rosparam get /${namespace
    [0]}/x)" y:="$(rosparam get /${namespace[0]}/y)" yaw:="$(rosparam get /${namespace[0]}/a)" &
pid=$!

trap "echo Killing all processes.; kill -2 TERM $pid; exit" SIGINT SIGTERM

sleep 24h
```

Listing 6: run-pioneer-nav-local: Script to launch gateway connections and navigation bundle

3.1.5 Launching rviz

The call to launch **rviz** is separated from the launch script in Listing 4 since it has typically failed to load correctly or completely when launched prior to setting up all the remote topic connections. The loaded rviz configuration file will show five Pioneer robots, their base scans, local costmaps, local and global plans, as well as their current goals.

3.2 Executing Waypoint Commands

Each aadi laptop runs a map_goal_client node that subscribes to waypoint commands issued on the map_goal topic. The map_goal_client manages the list of commanded waypoints and forwards the next waypoint to the move_base node after the current waypoint is reached. To send a new waypoint, publish a geometry_msgs/Twist message to the map_goal topic. For the Pioneer platforms, only the linear.x, linear.y and angular.z values are relevant. Also note that the commanded heading angle (angular.z) is in degrees. An example command line publication is:

```
rostopic pub -1 /pioneer1/map_goal geometry_msgs/Twist '[1.0, 2.0, 0.0]' '[0.0, 0.0, 90.0]'
```

This will command the robot to move to $(1, 2)$ with a heading of 90° in the /map frame.

The move_base node computes the instantaneous command velocities according to the planner specifications in base_local_planner_params.yaml, and these are scaled by the controller parameters in the pioneer_ros node. The published topic pioneer/cmd_vel is flipped to the Hub_Gateway, which simulates the outcome of executing the command.

3.3 Network Schematic

A schematic of the Pioneer Hub network is provided in Figure 3. Of particular note is the difference between advertise/pull connections and flip connections. In general, an advertised topic can be thought

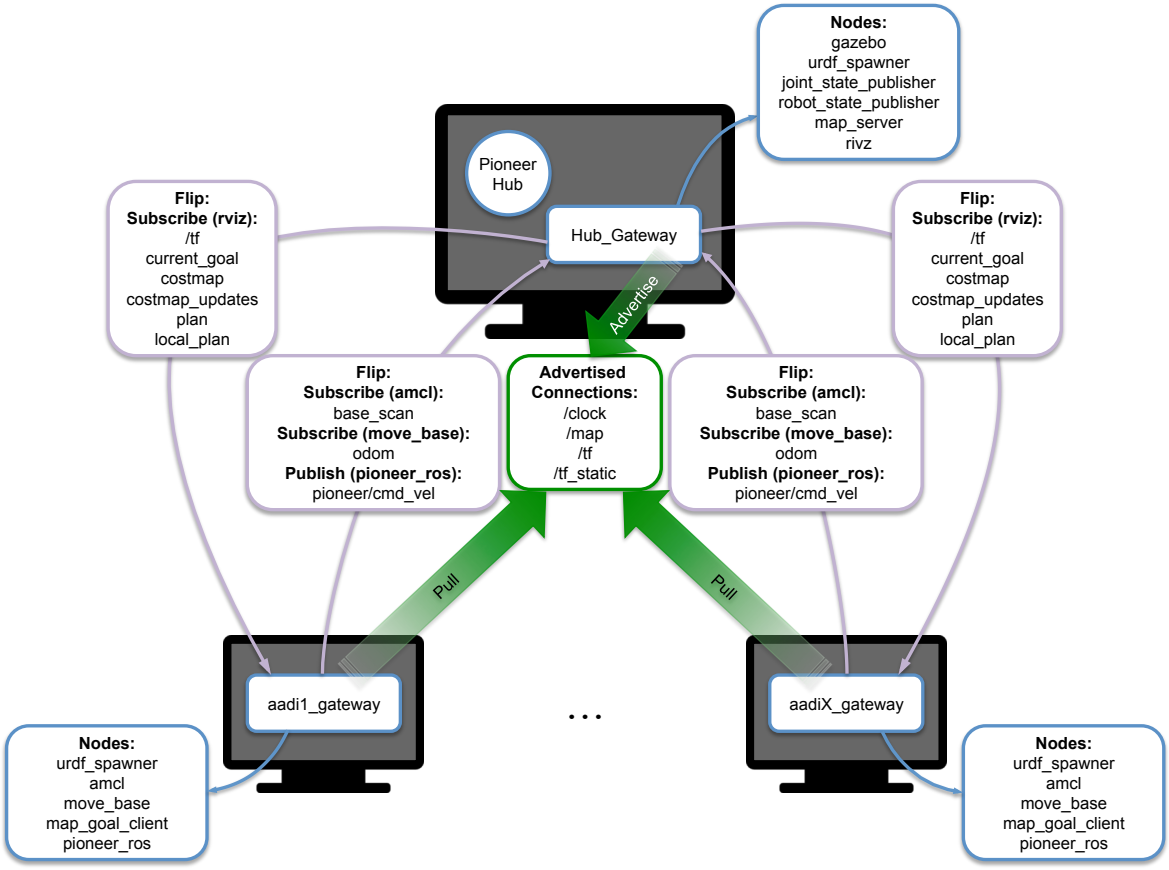

Figure 3: Network schematic for the ROS multimaster software-in-the-loop simulation. Each gateway is associated with a local ROS master. Although the hub (which also has its own ROS master) can exist on any computer connected to the same WiFi network, it is customary to run the hub on the base station computer. Flip connection requests are shown as purple arrows that originate from the gateway making the request. Advertise and pull requests are shown as green arrows that originate from the gateway making the request. Note that while the purple flip requests are a direct connection between the two specified gateways, advertised connections can be pulled from any gateway connected to the hub.

of as a 'broadcast' message (although the two are functionally different since gateways can choose to not pull advertised connections), and flip connections can be thought of as 'peer-to-peer' communication. These loose definitions can be used to guide the connection type most suitable to a particular topic.

When disconnecting from the hub, hold ctrl+c in the terminal from which the aadiX_gateway was launched. When shutting down the hub, hold ctrl+c in the original terminal from which you executed run-pioneer-hub. If you are shutting down the entire network, you can terminate these in any order. Note that error messages will appear in the gateway terminals if the hub is disconnected first.

Appendix A: Caveats, Corner Cases and Things I Don't Have an Elegant General Solution for Yet

Namespaces are the only thing preventing data contamination across the different robots. All the pioneers publish and subscribe to the same topics separated only by the robot name, which is appended as a prefix to the topic name at initialisation (e.g. /pioneer1/base_scan, /pioneer2/base_scan, etc.). If there is a communication error, for example if messages are not being communicated across the network, check that all the topic names are complete with the necessary namespace prefixes.

Linux machines can have multiple **python versions**, however some versions such as anaconda don't have the required libraries to support the built in ROS packages. The version installed by calling

```
sudo apt-get install python-rosinstall
```

is recommended and will typically act as the default python engine. However the path to this version may become masked in the `PATH` environment variable. You can check this by typing:

```
echo $PATH
```

Lines 3 and 4 of Listing 4 demonstrate one way of removing paths to other python versions from the `PATH` environment variable. Another option is to add an explicit reference to the python version you want to use by appending its directory to the start of `PATH`.

Blacklisted topics loaded by pioneer_gateway.launch from blacklist_multi_pioneer.yaml cannot be flipped or advertised/pulled from the gateway that blacklisted the topics. Note that it is still possible to set up a connection request on the network however no topic data will be communicated and *no error or warning messages will appear.*