

ROS-Gazebo-ErleRover & Evo-ROS

Installation and Usage Guide

Michigan State University



Glen Simon
Philip McKinley
Jared Moore
Anthony Clark

January 17, 2018

Contents

1	Introduction	3
2	Installation	4
2.1	Configuring your Ubuntu Machine	4
2.1.1	Install base packages	4
2.1.2	Install dependencies for MAVProxy	4
2.1.3	Install MAVProxy	4
2.1.4	Download and install ArUco	4
2.1.5	Install ROS Indigo	5
2.1.6	Install Gazebo	5
3	Configuring User Workspace	6
3.1	Download Ardupilot	6
3.1.1	Compile a specific branch of ardupilot	6
3.2	Download ErleRover_Scripts directory	6
3.2.1	Getting permission to push commits to the remote repo	6
3.3	Download ros_gazebo_python directory, which contains the BasicBot work. Optional	6
3.3.1	Getting permission to push commits to the remote repo	6
3.4	Create ROS workspace	6
3.4.1	Make workspace	6
3.4.2	Initialize the workspace	6
3.4.3	Download ros_catkin_ws_src which contains the development work for the MSU rover project	7
3.4.4	Compile the ros_catkin_ws workspace	7
3.5	Download Gazebo models	7
3.6	Configuring .bashrc	7
3.6.1	Add ROS setup to bash	7
3.6.2	Add ros_catkin_ws setup to bash	7
4	Basic Simulation Usage	8
4.1	Basic Erle-Rover simulation	8
4.1.1	Manually starting all needed processes	8
4.1.2	Using scripts to start simulation	9
4.2	Starting MAVProxy via a script	9
4.3	Basic obstacle avoidance simulation using Erle Robotics script	9
4.3.1	Manually starting all needed processes	10
4.3.2	Using scripts to start simulation	10
5	Evo-ROS	12
5.1	Motivation	12
5.2	Usage	13
5.2.1	Selecting and Configuring the GA	13
5.2.2	Evo-ROS Configuration Options	13

5.2.3	Starting Evo-ROS	14
5.2.4	Using Evo-ROS	15
6	Troubleshooting	16
6.1	Rover is not responding to either manual or scripted commands	16

Chapter 1

Introduction

We introduce Evo-ROS, a framework combining evolutionary search and ROS/Gazebo-based simulation. The framework provides researchers in evolutionary robotics access to the extensive support for real-world components and capabilities developed by the ROS community. Conversely, Evo-ROS enables developers in the ROS community, and more broadly robotics researchers, to take advantage of evolutionary search during design. Evolutionary algorithms can be applied to many aspects of development, including optimal configuration and placement of sensors and actuators, generation of compensatory behavior in case of failed/faulty components, and detection of unlikely-but-possible situations that might cause system failure.

The Evo-ROS framework separates the GA and simulation through a socket-based interface. Messages are sent from the GA to ROS/Gazebo instances, where the robot is evaluated. A fitness score is then returned through another socket back to the GA, where the generational loop is processed. In comparison to a typical ROS/Gazebo simulation, the primary extension a user needs to implement is mapping a genome into the robotic system. In our sample code, we show how to implement this approach for a simple Roomba-like robotic system as well as the more complex Erle-Rover robot.

To address performance concerns, Evo-ROS can exploit multiple levels of parallelism, including: deploying to (1) several physical machines; (2) several VMs on the same physical machine; (3) multiple ROS instances in the same VM; (4) multiple Gazebo simulations per ROS master; and (5) multiple, separate evaluations per Gazebo simulation (e.g., two robots completing tasks in separate zones not interacting). The current Github repositories contain implementations of (2) with an Ardupilot based controller and (4) with a state machine based controller.

Chapter 2

Installation

These installation directions are derived from the instructions provided by Erle Robotics found at http://docs.erlerobotics.com/simulation/configuring_your_environment.

It is recommended that this software be used on a machine running Ubuntu 14.04 64 bits.

2.1 Configuring your Ubuntu Machine

These steps only have to be done once per machine.

2.1.1 Install base packages

```
sudo apt-get update
sudo apt-get install gawk make git curl cmake -y
```

2.1.2 Install dependencies for MAVProxy

```
sudo apt-get install g++ python-pip python-matplotlib python-serial python-wxgtk2.8 python-scipy -y
sudo apt-get install python-opencv python-numpy python-pyparsing ccache realpath libopencv-dev -y
```

2.1.3 Install MAVProxy

```
sudo pip install future
sudo apt-get install libxml2-dev libxslt1-dev -y
sudo pip2 install pymavlink catkin_pkg --upgrade
sudo pip install -I MAVProxy==1.5.2
```

2.1.4 Download and install ArUco

1. Download ArUco 1.3.0 from here <https://sourceforge.net/projects/aruco/files/1.3.0/aruco-1.3.0.tgz/download>
2. Install ArUco

```
cd ~/Downloads # Replace this with your Download directory
tar -xvzf aruco-1.3.0.tgz
cd aruco-1.3.0/
mkdir build && cd build
cmake ..
make
sudo make install
```

2.1.5 Install ROS Indigo

Setup your computer to accept software from packages.ros.org, setup your keys and install (make sure your Debian package index is up-to-date):

```
sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu $(lsb_release -sc) main" > /etc/apt
           /sources.list.d/ros-latest.list'
sudo apt-key adv --keyserver hkp://ha.pool.sks-keyservers.net --recv-key 0xB01FA116
sudo apt-get update
```

Install ROS package, build, and communication libraries:

```
sudo apt-get install ros-indigo-ros-base -y
```

Initialize rosdep. Before you can use ROS, you will need to initialize rosdep, which enables you to easily install system dependencies for source you want to compile and is required to run some core components in ROS.

```
sudo rosdep init
rosdep update
```

It's convenient if the ROS environment variables are automatically added to your bash session every time a new shell is launched:

```
echo "source /opt/ros/indigo/setup.bash" >> ~/.bashrc
source ~/.bashrc
```

Get rosinstall and some additional dependencies

```
sudo apt-get install python-rosinstall \
ros-indigo-octomap-msgs \
ros-indigo-joy \
ros-indigo-geodesy \
ros-indigo-octomap-ros \
ros-indigo-mavlink \
ros-indigo-control-toolbox \
ros-indigo-transmission-interface \
ros-indigo-joint-limits-interface \
unzip -y
```

Get RQT graph

```
sudo apt-get install ros-indigo-rqt
sudo apt-get install ros-indigo-rqt-common-plugins
```

2.1.6 Install Gazebo

Setup your computer to accept software from packages.osrfoundation.org

```
sudo sh -c 'echo "deb http://packages.osrfoundation.org/gazebo/ubuntu-stable `lsb_release -c` main" > /etc/apt/sources.list.d/gazebo-stable.list'
```

Setup keys

```
wget http://packages.osrfoundation.org/gazebo.key -O - | sudo apt-key add -
```

Install gazebo7

```
sudo apt-get update
sudo apt-get remove .*gazebo.*' .*sdformat.*' .*ignition-math.*' && sudo apt-get update &&
sudo apt-get install gazebo7 libgazebo7-dev drcsim7 -y
```

Chapter 3

Configuring User Workspace

These steps will have to be done for each user on a machine if they would like their own local copies of the source files.

3.1 Download Ardupilot

The ArduPilot project is an open source autopilot for drones, rovers, and other platforms. We'll be using its code to simulate the Unmanned ground vehicles (UGVs), specifically the Erle-Rover:

3.1.1 Compile a specific branch of ardupilot

```
mkdir -p ~/simulation; cd ~/simulation
git clone https://github.com/erlerobot/ardupilot -b gazebo
```

3.2 Download ErleRover_Scripts directory

This was created to ease the process of starting all of the required processes used to simulate the Erle Rover.

```
cd ~/simulation
git clone https://github.com/gsimon2/ErleRover-Scripts.git
```

3.2.1 Getting permission to push commits to the remote repo

The above ErleRover-Scripts directory is a public repo and can freely be copied, but for access to submit changes please contact Glen Simon at glen.a.simon@gmail.com.

3.3 Download ros_gazebo_python directory, which contains the BasicBot work. Optional

```
cd ~/simulation
git clone https://github.com/jaredmoore/ros_gazebo_python.git
```

3.3.1 Getting permission to push commits to the remote repo

The ros_gazebo_python directory is a public repo and can freely be copied, but for access to submit changes please contact Jared Moore at swiftfoottim@gmail.com.

3.4 Create ROS workspace

3.4.1 Make workspace

```
mkdir -p ~/simulation/ros_catkin_ws/src
```

3.4.2 Initialize the workspace

```
cd ~/simulation/ros_catkin_ws/src
catkin_init_workspace
cd ~/simulation/ros_catkin_ws
catkin_make
source devel/setup.bash
```

3.4.3 Download ros_catkin_ws_src which contains the development work for the MSU rover project

```
cd ~/simulation/ros_catkin_ws
git clone https://github.com/gsimon2/ros_catkin_ws_src.git
```

Delete default src directory and replace with the downloaded one

```
cd ~/simulation/ros_catkin_ws
rm -r src
mv ros_catkin_ws_src src
```

Getting permission to push commits to the remote repo

This is a public repo and can freely be copied, but for access to submit changes please contact Glen Simon at glen.a.simon@gmail.com.

3.4.4 Compile the ros_catkin_ws workspace

```
cd ~/simulation/ros_catkin_ws
source devel/setup.bash
catkin_make --pkg mav_msgs mavros_msgs gazebo_msgs
catkin_make -j 4
```

3.5 Download Gazebo models

```
mkdir -p ~/.gazebo/models
git clone https://github.com/erlerobot/erle_gazebo_models
mv erle_gazebo_models/* ~/.gazebo/models
```

3.6 Configuring .bashrc

3.6.1 Add ROS setup to bash

It's convenient if the ROS environment variables are automatically added to your bash session every time a new shell is launched.

```
echo "source /opt/ros/indigo/setup.bash" >> ~/.bashrc
source ~/.bashrc
```

3.6.2 Add ros_catkin_ws setup to bash

For ROS to find the packages provided in ros_catkin_ws we need to source the setup file every time. This is easier if we also add this to the bash file.

```
echo "source ~/simulation/ros_catkin_ws/devel/setup.bash" >> ~/.bashrc
source ~/.bashrc
```

Chapter 4

Basic Simulation Usage

4.1 Basic Erle-Rover simulation

This process will bring up the Erle-Rover in a blank world and allow you to manually enter throttle and yaw commands via the MAVProxy terminal.

4.1.1 Manually starting all needed processes

The process of starting all processes can be found in more detail at:

http://docs.erlerobotics.com/simulation/vehicles/erle_rover/tutorial_1, but will be covered briefly here.

Executing APMrover2

This process requires two active terminals.

In terminal one enter:

```
source ~/simulation/ros_catkin_ws/devel/setup.bash
cd ~/simulation/ardupilot/APMrover2
../Tools/autotest/sim_vehicle.sh -j 4 -f Gazebo
# once MAVProxy has launched completely, load the parameters
param load /[path_to_your_home_directory]/simulation/ardupilot/Tools/Frame_params/3DR_Rover.
    param
# NOTE: replace [path_to_your_home_directory] with the actual path to your home directory.
# Example: param load /home/john/simulation/ardupilot/Tools/Frame_params/3DR_Rover.param
```

In terminal two enter:

```
source ~/simulation/ros_catkin_ws/devel/setup.bash
roslaunch ardupilot_sitl_gazebo_plugin rover_spawn.launch
```

This should start the Gazebo GUI and you should be able to see that the rover spawned in a blank world appearing similar to figure 4.1.

Controlling Erle-Rover using MAVProxy

Make the rover move forward. In the first terminal execute:

```
# in the MAVProxy prompt:
mode MANUAL
param set SYSID_MYGCS 255
rc 3 1900
```

Or backwards:

```
# in the MAVProxy prompt:
rc 3 1200
```

What we are doing here is overriding the 3rd channel of the RC, which corresponds to the throttle. Values are from 1100 to 1900. 1500 is to stop the throttle; so values above 1500 will make the rover move forward, and values above 1500 backwards. The same principle applies to the yaw, which is in the 1st channel of the RC. Values above 1500 will make it turn right, and below 1500 left. For instance:

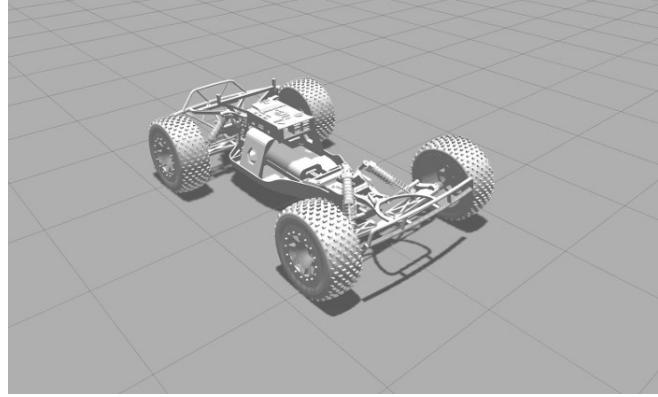


Figure 4.1: Erle-Rover model in Gazebo simulator

```
# in the MAVProxy prompt:
rc 1 1400
```

Note that first we had to use “param set SYSID_MYGCS 255”. This tell MAVProxy where the source for rover commands are coming from. To control the rover manually via the MAVProxy terminal, this value must be set to 255. When using scripts to control the rover, this value must be set to 1.

4.1.2 Using scripts to start simulation

The above simulation can also be started using a script to make the start up process easier. To run the script enter the following commands:

```
cd ~/simulation/ErleRover-Scripts/
./basic_sim.sh
```

This script also runs the “start_FPV” script, which will bring up a window displaying a first person view from the rover’s perspective. The “start_FPV” script can be ran on its own anytime a simulation is running to show the first person view by running the following commands:

```
cd ~/simulation/ErleRover-Scripts/
./start_FPV.sh
```

4.2 Starting MAVProxy via a script

MAVProxy is required when running any simulation of the rover and generally must manually be started prior to anything else. To make starting simulations easier, MAVProxy can be started via a script by using the following commands:

```
cd ~/simulation/ErleRover-Scripts/
./start_MAVProxy.sh
```

This will open an xterm window that sources the setup.bash file required to launch the ArduPilot SiTL software and execute ArduPilot’s sim_vehicle script with the rover parameters already loaded into it. The sim_vehicle script will then launch MAVProxy in the xterm window and ArduPilot in a separate window, generally also xterm, but may vary depending on your system.

4.3 Basic obstacle avoidance simulation using Erle Robotics script

This process will bring up the rover in a basic maze and start a node running a basic obstacle avoidance algorithm provided by Erle Robotics which will lead the rover through the maze without crashing into the walls.

The process of starting all processes can be found in more detail at:
http://docs.erlerobotics.com/simulation/vehicles/erle_rover/tutorial_2, but will be covered briefly here.

4.3.1 Manually starting all needed processes

To run this simulation three processes must be started:

1. MAVProxy - This can be started manually, see terminal one in section 4.1.1, or with a script, see section 4.2.

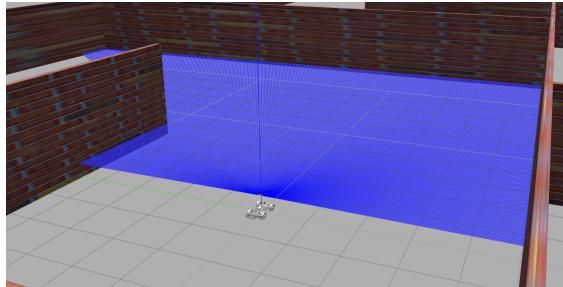
2. The launch file - This can be done with the following command:

```
roslaunch ardupilot_sitl_gazebo_plugin rover_maze.launch
```

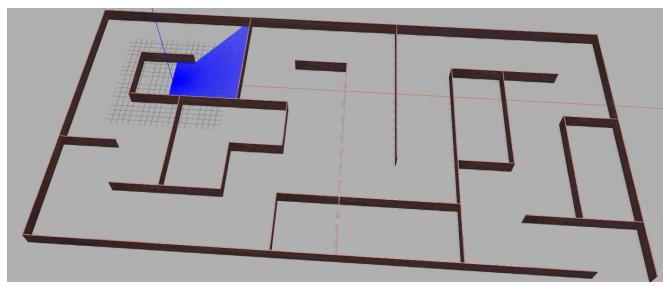
3. The controller node - This can be done with the following command:

```
rosrun erle_rover_explorer erle_rover_explorer.py
```

If this is done correctly you should see the rover spawn in a simple maze as can be seen in figure 4.2a. An overview of the maze can be seen in figure 4.2b.



(a) Erle-Rover in simple maze



(b) Simple maze overview

Notes

1. There is no stopping condition for this simulation and the rover will just continue to drive straight once it has exited the maze.
2. If the everything appears to have started correctly, but the rover is standing still in the maze, check to make sure MAVProxy is listening for commands on the right channel. For help with this issue see section 6.1.
3. The provided obstacle avoidance script also displays a window that represents the lidar scan and shows the current heading of the rover. An example of this can be seen in figure 4.3.
4. A first person view of the rover can be brought up by running the script. See section 4.1.2.

4.3.2 Using scripts to start simulation

The above simulation can also be started using a script to make the start up process easier. To run the script enter the following commands:

```
cd ~/simulation/ErleRover-Scripts/  
./explorer_sim.sh
```

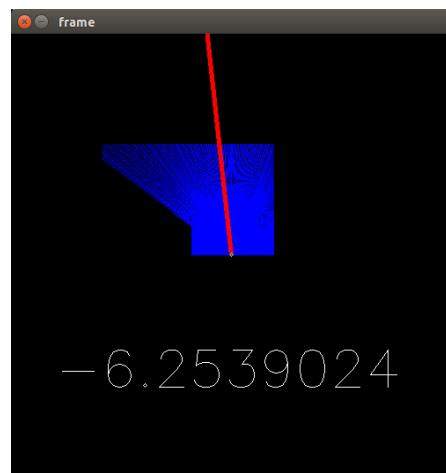


Figure 4.3: Lidar scan representation and current heading

Chapter 5

Evo-ROS

5.1 Motivation

Evolutionary robotics (ER) applies the basic principles of genetic evolution to the design of robots through the application of the genetic algorithm (GA). An artificial genome specifies the robots control system and possibly aspects of its morphology (body). Individuals in a population are evaluated (typically in simulation) with respect to one or more tasks, with the best performing individuals selected to pass their genes to the next generation. Evolutionary approaches have yielded effective controllers and physical designs for a variety of crawling, swimming, and flying robots [1, 5]. Our own research has applied evolutionary algorithms to optimize both morphology and control in aquatic and terrestrial robots. Evolving robot behavior and morphology is interesting in its own right, but from an engineering perspective, a major advantage of evolutionary search is the possible discovery of solutions (as well as potential problems) that the engineer might not otherwise have considered.

Simulation is an essential component of evolutionary robotics, greatly reducing the time to evolve solutions while avoiding possible damage to physical robots. The ER community typically creates one-off simulation environments from a few different physics engines (e.g., ODE, Bullet, VoxCAD, Simulink) to conduct an experiment. Environments are sparse, generally featuring the robot and possibly a few obstacles. Tasks typically comprise locomotion, navigation, and basic problem solving. Robots themselves contain only a few sensors, most often developed for the specific experiment being conducted. Hence, ER tasks are often limited by the scope of the simulation environment and how much time a developer has to code obstacles, sensors, and the platform itself. Models are not necessarily shareable between developers due to a lack of standardization. While many research questions can, and have, been answered by simple simulations, it becomes difficult to address more complex questions in these environments.

In contrast, the broader robotics community can address very complex tasks; the robotic systems utilize many sensor modalities to build a coherent understanding of their environment. By providing tested models of commercially available hardware, ROS/Gazebo provides a platform to study high-level behaviors while saving developer time during the design phase. Additionally, results have been shown to transfer to real robots, potentially addressing the reality-gap often encountered in ER.

In the Evo-ROS project, we have developed an evolutionary framework that integrates ROS-based simulations for robot evaluation. Our current prototype includes ROS, Gazebo, Ardupilot and MAVROS. The primary goals of the project are twofold. First, the framework enables researchers in ER to take advantage ROS and related simulation tools. Second, it enables robot developers to employ evolutionary search during the design process.

5.2 Usage

5.2.1 Selecting and Configuring the GA

There are currently several different GA options that can be selected. For this example we are going to use the script “symmetric_variable_number_sonar_GA_server.py”. This script is used to explore the optimal placement of sonar sensors on an Erle-Rover. This particular does have some limitations to the search space. First, placement of the sensors is limited to the front half of the rover and must be pointed towards the front. This restriction is because for the controller that we are using, the rover will only be driving in the forward direction and thus will have no need for rear facing sensors. The second limitation, is that sensors are limited to the outside border of the front half. This limitation is to allow ample room in the center of the rover for the placement of the battery and other electronics. Lastly, since the evolution of symmetric patterns is seen often in nature, the GA will force symmetry over the Y-axis of the rover for the placement of any sensors. That is, if there is a sensor selected to be placed at the front right corner of the rover at a 10 degree angle, then there will be a matching sensor placed at the front left corner of the rover at a -10 degree angle.

Once the GA is selected there are a few configuration options that should be observed. The configuration file can be opened by using the following commands:

```
cd ~/simulation/ros_catkin_ws/src/evo_ros/config/  
vim default_config.yml
```

This file contains the default configuration options for many aspects of Evo-ROS, but currently we are only interested in those that deal with the GA portion. The “ga_server” section can be found towards the top of the file and contains different parameters for the GA, such as: the tournament size, population size, generation count, log file name, and others.

After you are satisfied with the parameters being used the GA can be started by navigating to the GA directory by using the following commands:

```
cd ~/simulation/ros_catkin_ws/src/evo_ros/GA/
```

Next you can see the running options for the GA by using the “-h” flag as shown below:

```
python symmetric_variable_number_sonar_GA_server.py -h
```

Or you can start the GA in debugging mode with the following command:

```
python symmetric_variable_number_sonar_GA_server.py -d -ip 127.0.0.1
```

It is recommended to start the GA in debugging mode so that extra output is printed to the terminal, which allows to verify parameters and track progression better. Since the GA uses a comparatively low amount of CPU time as compared to the evaluation software, the extra printed output has a negligible effect to running time.

Also note that we specified the IP address that the GA should use to send out the genomes. By default, if this parameter is left blank, the GA will auto-assign it to be the IP address of the current machine.

If all is running correctly, the terminal output should appear close to what is shown in figure 5.2.

5.2.2 Evo-ROS Configuration Options

There are a number of other Evo-ROS configuration options that can be found in the configuration file described in section 5.2.1. The configuration file is broken into sections, each corresponding to a major part of the Evo-ROS framework. These sections include:

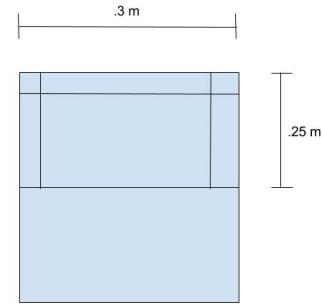


Figure 5.1: The sonar positioning search space for the example GA. The search space is limited to the outside borders of the front half of the rover, so that there is ample room in the center for other sensors. For now, there is no need for sensing behind the rover, since it will only be moving forward.

```

simongle@simongle-XPS-13-V14: ~/simulation/ros_catkin_ws/src/evo_ros/GA
symmetric_variable_number_sonar_GA_server.py -d -ip 127.0.0.1
Configuration file being used:
/home/simongle/simulation/ros_catkin_ws/src/evo_ros/GA/../../config/default_config.yml
GA Host IP = 127.0.0.1

Debugging option has been turned on!

Configuration Settings...
  NUM_EVAL_WORKS: 14
  GA_IP_ADDR: 127.0.0.1
  GA_SEND_PORT: 5000
  GA_RECV_PORT: 5010
  LOG_FILE_NAME: log.dat
  MAX_WAIT_TIME: 1200000
  POP_SIZE: 30
  GEN_COUNT: 25
  MUTATION_PROB: 0.15
  CROSS_OVER_PROB: 0.25
  TOURNAMENT_SIZE: 2

Press Enter when the workers are ready:

```

Figure 5.2: Expected output from starting the GA with the debugging options turned on.

- `sonar_filter`

This process is responsible for collecting all raw data from the sonar sensors in Gazebo and relying the information onto a “/sonar/#_filtered” topic. This allows different filters to be applied to the raw data. One use of this is to apply different failure models to the sensors before the information is passed onto the robot’s controller.

- `software_manager`

The `software_manager` process is responsible for spawning and managing all of the other required processes within Evo_ROS. It is in this set of configuration options that most of the customizations will be made. Here you can select things like: what launch files to use, as well as, which simulation managers and robotic controllers to use.

- `transporter`

The transporter process creates and maintains the connection to the external GA process.

- `sim_manager`

The `sim_manager` (short for simulation manager) is the process that is responsible

5.2.3 Starting Evo-ROS

The process of starting Evo-ROS is relatively simple once the configuration options have been set. For this example, no configuration options have to be changed. To start Evo-ROS, just has to start the “`software_manager`” script and the rest will be managed automatically. The “`software_manager`” script can be started by entering the following command into a new instance of the terminal:

```
roslaunch evo_ros software_manager.py -ip 127.0.0.1 -d -gui
```

Please note that the “-ip” flag is being passed to the `software_manager` with the same IP address that we that gave the GA so that the two will be connected.

The debugging option (“-d” flag) is also set so that each process will spawn in its own window. This allows for easier understanding of what is all happening, but should not be used in production runs as it goes have an impact on the running time.

The “-gui” flag is set as well. This option tell Gazebo that it should bring up its full GUI when it is launched, allowing you to view the rover in the simulated environment. This option should not be used in production runs, as it also impacts the running time.

As with any Evo-ROS process, the `software_manager` can be started with the “-h” help flag to display all possible options.

After the `software_manager` has started, one additional terminal should spawn running the Evo-ROS transporter. At this point you should have terminals open for the GA, `software_manager`, and the transporter. The Evo-ROS processes should be waiting to receive information for the GA, while the GA is waiting for you to give it the okay to start sending information.

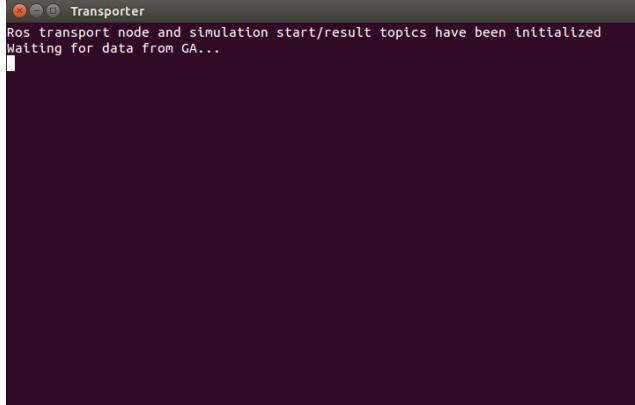


Figure 5.3: Expected output of the transporter terminal.

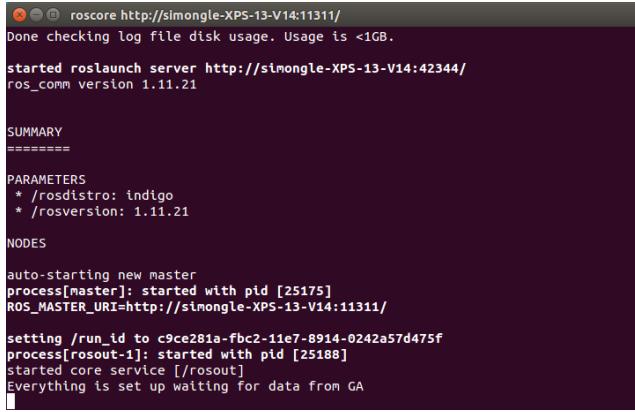


Figure 5.4: Expected output of the software manager terminal.

5.2.4 Using Evo-ROS

Once both the GA and the software_manager have been started, you simply have to give the GA permission to start sending information by pressing “enter” in that terminal. Upon pressing “enter” you should see the genomes from the GA being sent out. In the transporter window you will see one being received and being passed to the software_manager. The software_manager will then begin its process of setting up the simulated evaluation environment. This setup process includes spawning multiple xterm instances as well as a Gazebo instance (assuming the “-d” and “-gui” flags are set). This spawning process could take up to several minutes.

After the set up process has been completed, you should be able to view the rover in the Gazebo environment. The controller for the rover will automatically be started and after 10-20 seconds the rover should start its mission. During this time the simulation_manager process will be monitoring its progression through the maze and monitoring for ending criteria, such as: a crash, mission completion, or a max time limit is reached. When an ending criteria is met, the results of the evaluation will be sent back to the GA, where a fitness can be assigned to the individual.

This instance of Evo-ROS will continue to receive individuals to be evaluated until the GA is either finished or canceled.

Chapter 6

Troubleshooting

6.1 Rover is not responding to either manual or scripted commands

The most common issue for the rover not responding to commands as expected is that MAVProxy is listening for commands on the wrong channel. To check for this enter the following command into the MAVProxy terminal to check which channel is currently being listened to for commands:

```
param show SYSID_MYGCS
```

If the returned value is 1.0, MAVProxy is listening for commands from a scripted controller communicating using a ROS topic.

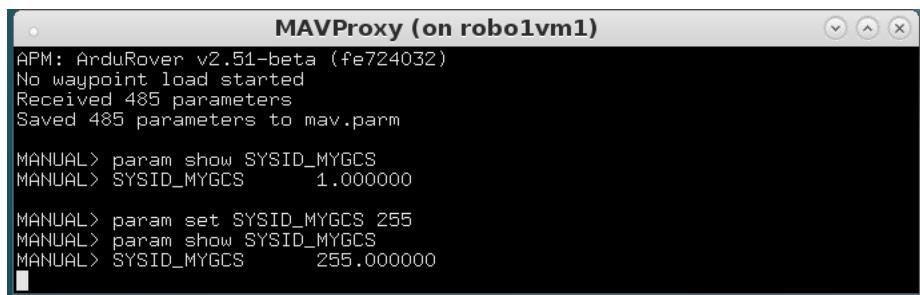
If the returned value is 255.0, MAVProxy is waiting for commands manually entered into the MAVProxy terminal.

To change the channel that MAVProxy is listening on use the following command:

```
param set SYSID_MYGCS {value}
```

where value is between 1 and 255.

Note: MAVProxy must be connected to the rover, either a physical rover or a ROS node acting as the rover, for these commands to work.



```
MAVProxy (on robo1vm1)
APM: ArduRover v2.51-beta (fe724032)
No waypoint load started
Received 485 parameters
Saved 485 parameters to mav.parm

MANUAL> param show SYSID_MYGCS
MANUAL> SYSID_MYGCS      1.000000

MANUAL> param set SYSID_MYGCS 255
MANUAL> param show SYSID_MYGCS
MANUAL> SYSID_MYGCS      255.000000
```

Figure 6.1: Checking MAVProxy command source