Swarmie User Manual: A Rover Used for Multi-agent Swarm Research

Gilbert Montague
Kennedy Space Center
Physics
Center Innovation Fund (CIF) Swarming Robot Research
Fall Session
18 11 2014

Swarmie User Manual: A Rover Used for Multi-agent Swarm Research

Gilbert Montague[1]
Baldwin Wallace University, Berea, Ohio, 44017, USA

The ability to create multiple functional yet cost effective robots is crucial for conducting swarming robotics research. The Center Innovation Fund (CIF) swarming robotics project is a collaboration among the KSC Granular Mechanics and Regolith Operations (GMRO) group, the University of New Mexico Biological Computation Lab, and the NASA Ames Intelligent Robotics Group (IRG) that uses rovers, dubbed "Swarmies", as test platforms for genetic search algorithms. This fall, I assisted in the development of the software modules used on the Swarmies and created this guide to provide thorough instructions on how to configure your workspace to operate a Swarmie both in simulation and out in the field.

Nomenclature

Algorithm	= a process or set of rules to be followed in calculations or problem-solving operations
Genetic Algorithm	= a software algorithm that mimics the process of natural selection
Swarmie	= the name of the rover used for the swarming robotics research project
AprilTags	= the target detection system chosen for the swarming robotics research project
ROS	= Robot Operating System. A set of software libraries used to build robot applications
Gazebo	= Free, open source robot simulation software
BASH	= Bourne-again shell. A UNIX shell.
GUI	= Graphical user interface

I. Introduction

The CIF Swarming Robotics Research Project is a collaboration among the KSC Granular Mechanics and Regolith Operations (GMRO) group, the University of New Mexico Biological Computation Lab, and the NASA Ames Intelligent Robotics Group (IRG) and aims to develop genetic software algorithms for a variety of different tasks including robotic mining. This project focuses on developing autonomous robots that if placed in a foreign environment will be able to learn the environment and share the data about the terrain to other agents in the swarm. The robots use genetic algorithms to adapt their behavior in respect to power management and are capable of such tasks as route planning and dynamic navigation. The Swarmie rover was designed to provide researchers with a cost effective physical test platform to test the genetic algorithms rather than only relying on computer simulations.

One of my major tasks this fall was to aid in the development of software used by the Swarmies. I created this technical document, the Swarmie User Manual, to provide a comprehensive paper that clearly details the configuration process for a Swarmie rover. This document is designed so that anyone with a basic understanding of electronics and software can setup and operate a Swarmie.

II. Configuring the ROS workspace

The Robot Operating System (ROS) was chosen as the underlying framework to handle the individual Swarmie processes. According to the ROS website, "[ROS] is a collection of tools, libraries, and conventions that aim to simplify the task of creating complex and robust robot behavior across a wide variety of robotic platforms."[1] Most notably, ROS provides

[1] Undergraduate Student, Department of Physics and Astronomy, Kennedy Space Center, Baldwin Wallace University

a system for small software modules to communicate via a common messaging bus. This functionality was one of the key selling points for using the ROS framework.

ROS is optimized for the Linux Operating System and therefore it makes sense that the development and implementation of the ROS software modules are conducted on a Linux Operating System, specifically Ubuntu 12.04LTS. The following details how to install and configure a version of ROS (Hydro) on your Ubuntu 12.04LTS Linux developer workstation.

A. Installing ROS Hydro

The following is a condensed version of the installation guide from the wiki.ros.org [2]. Please refer to the ROS website for detailed information regarding ROS installation, configuration, and implementation.

You will first need to setup your workstation to accept software from packages.ros.org. To do this, open a new terminal window and enter:

```
$ sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu precise main" > /etc/apt/sources.list.d/ros-latest.list'
```

After that command, enter the following to set up your keys for installing ROS Hydro:

```
$ wget https://raw.githubusercontent.com/ros/rosdistro/master/ros.key -O - | sudo apt-key add -
```

Now, make sure your Debian package index is up-to-date by entering the following command:

```
$ sudo apt-get update
```

Finally, install ROS Hydro by entering the following command:

```
$ sudo apt-get install ros-hydro-desktop-full
```

This will install the full desktop version of ROS Hydro which includes rqt, a QT-based framework for graphical user interface (GUI) development for ROS. Rqt is used as the primary GUI for operating the Swarmies as it provides an easy to use interface for inputting commands and parameters.

Before you can use ROS, you need to initialize rosdep, a tool that enables you to easily install system dependencies that allow you to run certain core ROS components. To initialize rosdep, enter the following commands in your active terminal window:

```
$ sudo rosdep init

$ rosdep update
```

The final steps of the ROS Hydro installation are setting up your workstation's environment variables so that you can use all of the packages ROS has to offer. To do this, execute the following commands:

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

$ source ~/.bashrc
```

This will source the correct ROS environment variables automatically every time you start a new BASH session. This is very convenient when you are using ROS.

B. Creating a workspace

The following is a condensed version of the installation guide from the wiki.ros.org [3]. Please refer to the ROS website for detailed information regarding ROS installation, configuration, and implementation.

The two catkin workspaces for the Swarmie project are *rover_onboard_workspace* and *rover_driver_workspace*. This section will go through the steps to create the *rover_onboard_workspace* catkin workspace. The steps detailed here are identical for the *rover_driver_workspace* catkin workspace. All source code for the workspaces are available upon request.

First, create and initialize a workspace directory in your home directory by executing the following commands. These commands will create a *rover_onboard_workspace* workspace directory and then initialize that directory so that catkin, the official build system of ROS, can build the workspace.

```
$ mkdir ~/rover_onboard_workspace

$ cd ~/rover onboard workspace
```

With the *rover_onboard_workspace* initialized, you need to create the packages in the workspace to ensure that everything in the workspace is linked correctly with ROS. To do this, run the following command in the top level of your catkin workspace:

```
#This is an example, do not try to run this:
$ catkin_create_pkg <package_name> [depend1] [depend2] [depend3]
```

Where <*package name*> is the package you want to add to the workspace and *depends#* variable is the dependency you want that package to have. For the Swarmies, you would run the following command and replace the <*package name*> variable with the actual package name (*rover_onboard_abridge, rover_onboard_health_status*, *rover_onboard_localization*, etc.)

```
$ cd ~/rover_onboard_workspace

$ catkin_create_pkg <package_name> std_msgs rospy roscpp
```

Then, with all of the packages created, copy and paste the *src/* directory into the workspace that contains all of the source code necessary for the Swarmie modules. Contact information for Swarmie source directories are available in the References section.

III. Swarmie Software

A. Software Description

Before building the software, review the following tables and become familiar with what each software package does and where it is located.

Table 1: rover_onboard_workspace/src/ Package Descriptions

Package	Description	Executables
rover_onboard_abridge	Arduino bridge that publishes data from the microcontroller as ROS message types.	abridge
rover_onboard_health_status	Collects and publishes health and status data from the onboard computer.	health_status
rover_onboard_localization	Notifies Swarmie of its location and attitude in 3D space by reading intertial measurement unit (IMU) data and global positioning system (GPS) data.	localization
rover_onboard_mapping	Creates and updates real-time maps based on data collected from the obstacle detection, target detection, and localization modules.	mapping
rover_onboard_mobility	Takes input from the GUI, joystick, or path planner to move the Swarmie. Also contains the logic for the random walk routine.	mobility
rover_onboard_obstacle_detection	Notifies the Swarmie of obstacles by reading ultrasonic sensor data. Outputs obstacle data to the map.	obstacle
rover_onboard_path_planning	Sends a real-time path to the Swarmie based on information from the mission parameters and mapping module.	path
rover_onboard_target_detection	Opens webcam, executes AprilTag target detection algorithm, and notifies the Swarmie of targets.	camera, target

Table 2: rover_driver_workspace/src/ Package Descriptions		
Package	**Description**	**Executables**
rover_driver_gazebo_launch	Contains the necessary scripts, executables, and files to launch a Gazebo session	*.launch files for the simulation rovers
rover_driver_ptp	Rqt plugin for point-to-point functionality.	Rover PTP (RQT GUI)
rover_driver_rqt_mcp	Mission control rqt plugin. It is the GUI for connecting to a Swarmie and inputing mission parameters.	Rover MCP (RQT GUI)
rover_driver_rqt_motor	Rqt plugin for maunal teleoperation of a Swarmie	joystick_driver, Rover Motor (RQT GUI)
rover_driver_world_state	Keeps track of the number of found tags and the universal map	world_state

B. Software Build

As stated before, the *src/* directories for the Swarmie research project can be obtained by contacting one of the researchers listed in the References section. After acquiring the *src/* directories, copy them into their respective workspaces. Once copied over, execute the following commands to build the workspaces. Note that since the Swarmie project uses more than one catkin workspace, they must be "chained" together to work as intended and thus it is imperative that the chronological order of the following commands are kept.

```
$ cd ~/rover_onboard_workspace/

$ catkin_make

$ source rover_onboard_workspace/devel/setup.bash

$ cd ~/rover_driver_workspace

$ catkin_make

$ cd ~/rover_onboard_workspace/

$ catkin_make
```

For your convenience, you should update the last few lines of your *~/.bashrc* file with the following lines. This will source the necessary environment variables automatically every time you start a new BASH session. Remember to source your *~/.bashrc* file after making the changes. Add these lines to your *~/.bashrc* file:

```
source /opt/ros/hydro/setup.bash
source ~/rover_driver_workspace/devel/setup.bash
source ~/rover_onboard_workspace/devel/setup.bash
```

IV. Gazebo Robot Simulator

The following is a condensed version of the installation guide from the gazebosim.org [4]. Please refer to the Gazebo website for detailed information regarding Gazebo installation, configuration, and implementation.

Gazebo is a free, open source robot simulator that is widely used in the robotics community and is the simulator of choice for the Swarmie project. Similar to the ROS Hyrdo installation, you must first setup your workstation to accept software from packages.osrfoundation.org. In a terminal window, run the following:

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

Setup your keys.

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

Install Gazebo.

```
$ sudo apt-get update
$ sudo apt-get install gazebo4
```

With Gazebo successfully installed, add the following line to your *~/.bashrc* file:

```
source /usr/share/gazebo-1.9/setup.sh
```

A. rover_misc_workspace and Inserting Models into Gazebo

The directory rover_misc_workspace is aptly named because it contains all of the miscellaneous directories and files necessary for the Swarmie project and most notably contains all of the Gazebo model files. To make sure that Gazebo knows where to "look" for the model files, add this line to your *~/.bashrc* file:

```
export GAZEBO_MODEL_PATH=/home/USER/rover_misc_workspace/src/gazebo/models
```

You will need to change USER to the username of the machine you are on. This addition will allow you to insert the correct models into Gazebo.

V. Workflow Automation

The majority of testing using the simulation requires a significant amount of time starting various processes and inserting the correct models into Gazebo. Since the majority of the time testing in the simulation should be spent conducting tests and not setup, several BASH scripts were created to automate the more tedious tasks.

A. software_load

The *software_load* script is a program that will automatically load the necessary software onto a Swarmie. This script is located in the *rover_misc_workspace* workspace in the *rover_scripts* directory. To invoke the script, enter the following commands:

```
$ cd ~/rover_misc_workspace/src/rover_scripts/

$ chmod +x software_load.sh

$ ./software_load [USERNAME] [HOSTNAME] [SWARMIE USERNAME] [SWARMIE HOSTNAME]
```

Where the **USERNAME** and **HOSTNAME** arguments are your current workstation's username and hostname and **SWARMIE USERNAME** and **SWARMIE HOSTNAME** are the username and hostname of the Swarmie you are loading the software onto. Contact the developers for Swarmie username and hostname information.

B. driver_launch

Similarly, the *driver_launch* script is found in the *rover_misc_workspace* workspace in the *rover_scripts* directory. Also, *driver_launch*, as the name alludes to, should be located and executed from the laptop driver station. To invoke the script, execute the following commands:

```
$ cd ~/rover_misc_workspace/src/rover_scripts/

$ chmod +x driver_launch.sh

$ ./driver_launch
```

This script will automatically start the necessary driverStation processes to operate a Swarmie for physical field trials and testing. Specifically, it will launch a *roscore* session, *rqt, rover_driver_world_state*, and ask if you would like to use the joystick to teleoperate the Swarmie.

C. rover_driver_gazebo_launch

The script rover_driver_gazebo_launch is located in the rover_driver_workspace workspace under the rover_driver_gazebo_launch directory and is responsible for starting up everything you need to conduct test simulations in Gazebo. To invoke the script, enter the following commands in your terminal window:

```
$ cd ~/rover_driver_workspace/src/rover_driver_gazebo_launch/src

$ chmod +x rover_driver_gazebo_launch.sh

$ ./rover_driver_gazebo_launch.sh
```

The script will first prompt you to choose the AprilTag distribution you would like to implement for your test. AprilTags, the tag detection system developed by researchers at the University of Michigan, are used to simulate resources such as water or minerals [5]. They are used both in simulation and out in the field.

Table 3: AprilTag Distribution Descriptions	
Uniform	Spawns n number of randomly distributed single AprilTags (where n is a positive integer value determined by user input).
Clustered	Spawns four randomly distributed groups of 64 AprilTags.
PowerLaw	Spawns one pile of 64 AprilTags, four piles of 16 AprilTags, 16 piles of four AprilTags, and 64 single AprilTags

After inputting the desired AprilTag distribution, you will be prompted for the name of the rover you wish to spawn first. To avoid unexpected results, the script will not let you proceed until you have entered a valid Swarmie name and that the Swarmie model file exists in *rover_misc_workspace/src/gazebo/models*. The script will then ask you how many obstacles you would like for your simulation. The obstacles are blocks that are 25 centimeters cubed and are randomly distributed throughout the simulation.

With the correct name, obstacle, and tag variables set, the script will then start a *roscore* session and the software modules associated with the Swarmie you chose to insert first. After that, it will then prompt for how many more Swarmies you would like in your simulation and what their names should be. Similarly to the first Swarmie, the script will start the necessary software for each selected Swarmie. If you do not wish to add any additional Swarmies, simply input a “0”.

The script, after starting the Swarmie processes, will start a Gazebo session, an *rqt* session, and *rover_driver_world_state* and wait for the user to select where to insert the first rover and any additional rovers you chose to add.

Finally, after spawning the Swarmies, the script will spawn the AprilTags and obstacles into the simulation environment and wait for the user to press “q” to close all of the processes started by the script. At this point, the user can use *rqt* to control the Swarmie(s) in the simulation environment.

VI. Graphical User Interface

The GUI used to control the Swarmies is *rqt*. Figure 1 is a screenshot of a typical *rqt* session while operating one rover.

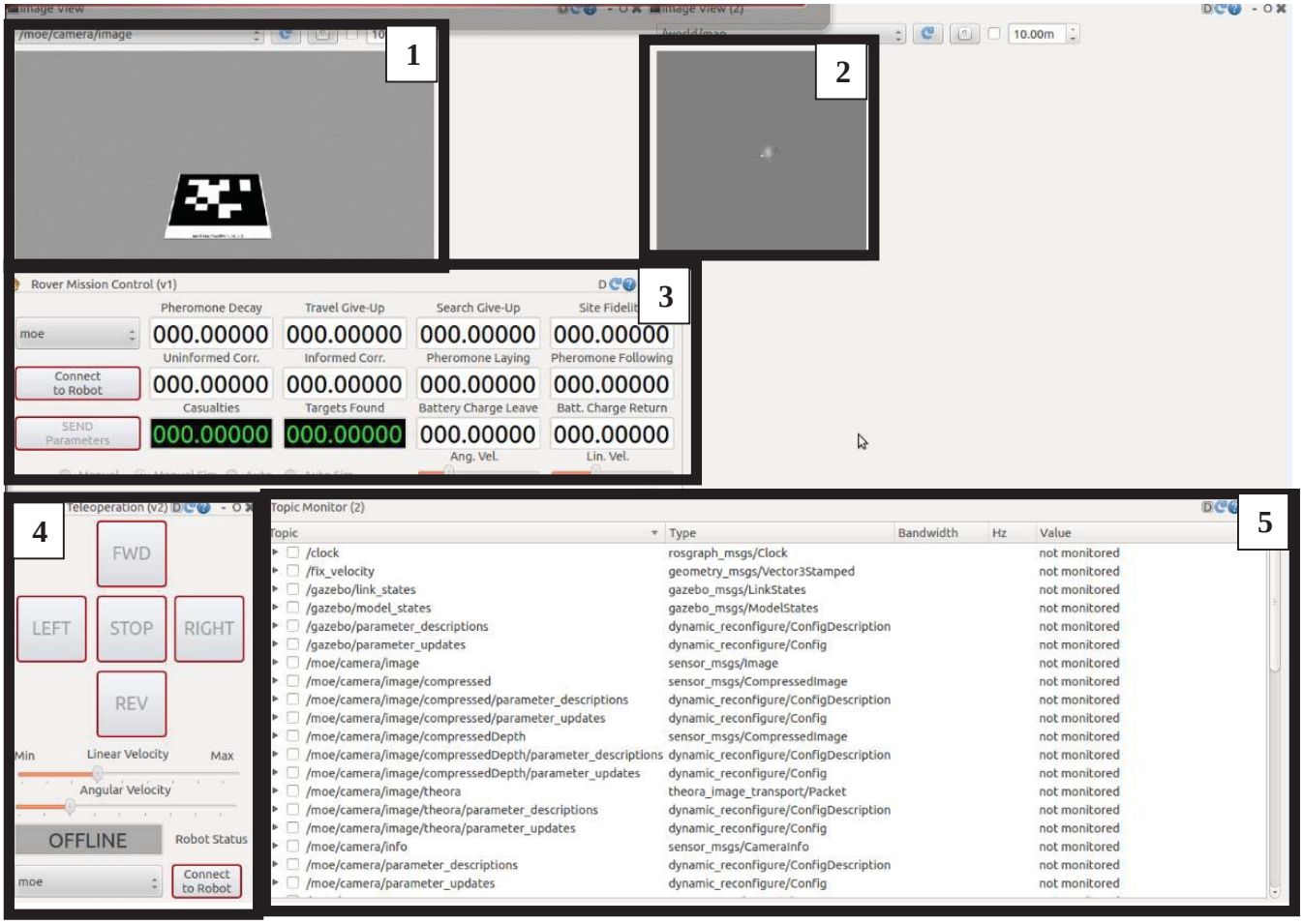

Figure 1: *A screenshot of the rqt graphical user interface used to operate the Swarmies. Essentially, the command center, the rqt GUI is where the user can monitor a Swarmie's camera feed, (window 1), monitor the world map, (window 2) input commands and mission parameters to the Swarmie, (window 3), manually teleoperate the Swarmie, (window 4), and also monitor the published message topics (window 5).*

A. Parameters

The following table describes the parameters generated by the genetic algorithm. The parameters can be sent to the Swarmie by entering them into the *rqt* plugin *rover_mcp* and clicking the "Send Parameters" button.

Parameter Description	
Travel Give-up	Percent chance Swarmie will give up travelling in this time slice
Search Give-up	Percent chance the Swarmie will give up the search and return home
Battery Recharge	Percent chance the Swarmie gives up the search to return home to recharge its batteries
Uninformed Correlation	Window of degrees the Swarmie will choose from when moving towards a target
Informed Correlation	Exponential decay rate at which the Swarmie's search pattern decays into the walking search pattern
Site Fidelity	Percent chance that a Swarmie who just found a target will come back to this target to gather more data
Pheromone Laying	Percent chance a Swarmie who just found a target will lay pheromones on the way home
Pheromone Following	Percent chance a Swarmie will follow a pheromone trail when found

VII. Operating in the Field

One of the primary goals of this Swarmie project is to test the software and genetic algorithms on actual physical rovers. The process of the physical trial setup is as follows:

1. Plug the router into a power source and wait for the network to come up.
2. Turn on the Swarmie(s).
3. Wait for the GPS to lock on the Swarmie(s). A small red light emitting diode (LED) on the GPS module on the microcontroller board will start blinking when it has a successful connection. If there is no lock after three minutes, push the reset button on the microcontroller.
4. Start up the driverStation laptop and connect to the wireless network.
5. Invoke the aforementioned *driver_launch* script located in the *rover_misc_workspace* workspace and in the *rover_scripts* directory. This will start *roscore*, *rqt*, and *world_state*.
6. Each Swarmie must have its own instance of Rover Driver MCP open in *rqt* to operate. Opening additional *rqt* windows for the camera feed and map are recommended but not necessary.
7. Secure Shell (SSH) into a Swarmie and invoke the *rover_onboard_node_launch.sh* script located in the home directory of the Swarmie. This will automatically start all of the necessary onboard nodes for the Swarmie. Repeat this step for each Swarmie you have on.
8. After the test, kill off all of the onboard processes and quit out of the processes started by the *driver_launch* script.
9. Make sure to completely power off the Swarmies before putting them away.

Eventually, when the team is past the development and debugging stages of the project, the onboard processes will start automatically when the Swarmie is powered on, eliminating the need for the SSH connection. For now, the onboard processes are started manually to aid in the development process.

VIII. Other Work

In addition to writing this high level users guide, I was responsible for a variety of other software development and hardware tasks on the Swarmie project this fall. I wrote the majority of the BASH scripts that automated the workflow including the script that starts up a trial run in Gazebo. I also integrated the C version of the AprilTag library into the onboard software which reduced the central processing unit (CPU) usage of the target detection software module by almost half. I was involved in making physical modifications to the Swarmies including adding a felt deck under the ultrasonic sensors to filter the ground noise to the ultrasonic sensors and thus reducing the number of false positive obstacle detections. I was heavily involved in other aspects of the onboard software development including testing and debugging the software in the Gazebo simulation and in the physical Swarmie rovers. I am very grateful to have had such a supportive team that provided me with tasks that were crucial to the success of the project and not just "busy work". I feel very privileged to have had worked on important software and hardware tasks that I believe will benefit mankind.

IX. Conclusion

When operating any piece of equipment, a comprehensive guide is necessary to limit confusion and reduce the learning curve as much as possible. This Swarmie User Manual provides a complete and informative guide to successfully setup and run a group of virtual and physical Swarmies. The detailed instructions, annotated pictures, informative tables, and helpful hints provide the reader with a useful tool set to begin operating their rover swarm with minimal difficulty. This will reduce the precious time spent setting up the Swarmies and allow more time to focus on the ground-breaking research being conducted.

Acknowledgements

I would like to thank the team members who have mentored me during my internship: Cheryle Mako, Kurt Leucht, Lien Moore, Matt Nugent, and Karl Stolleis. They have been fantastic mentors to me and I consider them all lifelong colleagues and friends. I would also like to thank the folks in the Launch Control Center (LCC): Adam Niev, Kelvin Ruiz, Allan Villorin, Michael McDonough, and Leandro James for putting up with my calls for an escort into and out of the LCC. Finally, thank you to all the people over at the Operations Support Building II (OSB II) especially Mr. Greg Clements, Caylyne Shelton, and Michael Duffy.

References

[1] *"Robot Operating System,"* http://www.ros.org/about-ros/, retrieved 11/2014
[2] *"Ubuntu install of ROS Hydro,"* http://wiki.ros.org/hydro/Installation/Ubuntu, retrieved 11/2014
[3] *"Creating a workspace for catkin"* http://wiki.ros.org/catkin/Tutorials/create_a_workspace/, retrieved 11/2014
[4] *"Download and Install Gazebo,"* http://gazebosim.org/tutorials?tut=install, retrieved 11/2014
[5] Olson, E. AprilTag: A robust and flexible visual fiducial system. Proceedings of the IEEE International Conference on Robotics and Automation (ICRA), 3400-3407.

A. Developer Contact Information

Kurt W. Leucht
Command and Control Software Developer
NASA Control and Data Systems
Kurt.Leucht@nasa.gov

Karl Stolleis
University of New Mexico
stolleis@unm.edu