User Manual
Minesweeper

Version 0.2

Editor: Elin Näsholm
Date: November 30, 2017

Status

Reviewed	ReviewerName	Date1
Approved	ApproverName	Date2

Project Identity

Linköping University, Department of Electrical Engineering (ISY)
Autumn 2017

Client:	Martin Lindfors, Linköping University **Phone:** +46 13 28 13 65, **E-mail:** martin.lindfors@liu.se
Customer:	Torbjörn Crona, Saab Dynamics **E-mail:** torbjorn.crona@saabgroup.com
Course Examiner:	Daniel Axehill, Linköping University **Phone:** +46 13 28 40 42, **E-mail:** daniel.axehill@liu.se
Project Manager:	Hampus Andersson
Advisors:	Per Boström-Rost, Linköping University **E-mail:** per.bostrom-rost@liu.se Erik Ekelund, Saab Dynamics **E-mail:** erik.ekelund@saabgroup.com Axel Reizenstein, Saab Dynamics **E-mail:** axel.reizenstein@saabgroup.com

Group members

Name	LiU-id	Phone number	Responsibility
Andreas Hägglund	andha796	072-713 38 70	Head of Hardware
Andreas Lundgren	andlu901	070-022 56 44	Head of Design
Elin Näsholm	elina044	073-094 94 03	Head of Documentation
Fredrik Gustafsson	fregu856	070 578 63 48	Head of Slam Implementation
Fredrik Tormod	freto995	073-775 05 36	Head of Control and Route Planning
Hampus Andersson	haman657	073-675 93 56	Project Manager
Jonathan Jerner	jonje173	070 296 61 50	Head of Software
Mattias Andreasson	matan461	070-822 53 67	Head of Testing

Group E-mail: tsrt10-minrojning@googlegroup.com
Homepage: TBA

Mail to each individual can be sent to "LiU-id"@student.liu.se.

Contents

1 Introduction **1**
1.1 System Description . . . 1

2 Definitions **2**

3 Balrog **3**
3.1 Hardware . . . 3
3.2 Software . . . 5
3.2.1 Setup of the Balrog Software Stack . . . 6
3.2.2 Setup of the Balrog Simulation Stack . . . 10
3.2.3 Configure Linux Computer to Compile RPI Code . . . 15
3.2.4 Compile & Transfer RPI Code . . . 15
3.3 BaseStation . . . 15
3.3.1 Setup of GUI . . . 16
3.3.2 GUI Usage . . . 16
3.4 Charging . . . 17
3.5 RC Controller . . . 17
3.6 Usage . . . 19
3.6.1 Initial Setup . . . 19
3.6.2 Running Autonomous Mission . . . 20
3.6.3 Running SLAM & Manually Control Balrog . . . 21
3.6.4 Manually Control Balrog . . . 21
3.6.5 Usage of the Balrog Simulator . . . 21

4 Sauron **23**
4.1 Hardware . . . 23
4.1.1 Remote Controller . . . 23
4.1.2 Drone . . . 24
4.1.3 BaseStation . . . 26
4.1.4 Router . . . 27
4.2 Software . . . 27
4.2.1 Sauron . . . 27
4.2.2 BaseStation . . . 30
4.2.3 GUI . . . 32
4.3 Usage . . . 32
4.4 Getting Started Quick Guide . . . 33

Document History

Version	Date	Changes made	Sign	Reviewer
0.1	28/11-17	First draft.	-	EN
0.2	30/11-17	Minor changes	-	EN

1 Introduction

This document is a user manual for the autonomous minesweeping system. The purpose of the document is to provide users and developers with all practical information needed to deploy the current system and further improve on it in the future. The document is an extension of last year's user manual [1].

1.1 System Description

The minesweeping system consists of three major components: Balrog, Sauron and BaseStation. Balrog is a tracked robot equipped with various sensors and computing hardware, designed to autonomously map, cover and secure an assigned area on the ground. Sauron is a UAV equipped with additional onboard computing hardware and a camera, designed to autonomously track Balrog in order to provide an aerial view of the assigned area. Both platforms can also be manually controlled by a human user via RC hand controllers and they are both connected to BaseStation over WiFi. BaseStation is a computer running ROS that communicates with the platforms and displays relevant sensor data in separate GUIs. Balrog and Sauron can be seen together in Figure 1 below.

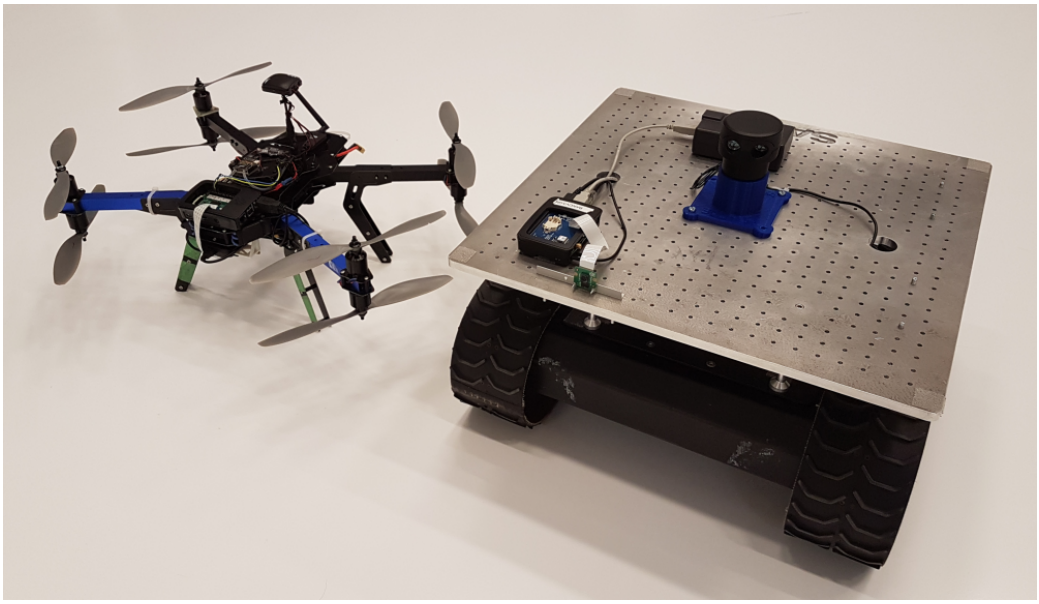

Figure 1: Sauron and Balrog, the two main components of the minesweeping system.

2 Definitions

AprilTags	A matrix of bar codes used to identify predefined objects.
Balrog	The ground vehicle aimed for minesweeping used in the project.
EKF	Extended Kalman Filter.
GUI	Graphical user interface.
HAL	Hardware abstraction layer.
MAV-link	Micro Air Vehicle Link.
Mine	An object or image on the ground acting as a mine.
Object	Any physical item in the testing area such as a mine, an obstacle or a wall.
Obstacle	An object in the test area that acts as an obstacle for the Balrog.
ROS nodes	Units of code that can be run independently of each other and communicate via ROS topics.
ROS topics	Named data channels that ROS nodes can send messages to by publishing on the topic, and read messages from by subscribing on the topic.
ROS	Robot Operating System. A set of open source software libraries and tools enabling communication between different modules in robotic systems.
Route	A set of way-points created by the user or the platform itself.
rqt	Qt-based framework for GUI development in ROS.
rviz	Visualization tool for ROS.
Sauron	The drone used in the project for collaborating with the ground vehicle.
Search area	The area that has not yet been searched for mines by Balrog.
Searched area	The area that has been visited by Balrog.
SLAM	Simultaneous localization and mapping.
TCP	Transmission Control Protocol.
Test area	An indoor predefined area where the platform will be tested.
The project group	The group of students involved in the project.
UAV	Unmanned Aerial Vehicle.
UDP	User Datagram Protocol.
User	The human interacting with the system in any way.

3 Balrog

In this section all practical information needed to setup, use and further develop the Balrog subsystem is provided.

3.1 Hardware

All computing and sensor hardware components mounted on Balrog are listed in Table 1 and Table 2 below, respectively.

Computer	An Ubuntu computer running ROS placed on the platform to run the SLAM, navigation and control modules, to communicate with *RPI* and to communicate with BaseStation over WiFi.
RPI	A Raspberry Pi 3 model B+ that is the main communication hub on Balrog. It communicates with *Arduino-Int* to send control signals and read wheel encoder data. It receives control signals from the RC controller via *Arduino-Ext* in manual mode and from *Computer* in autonomous mode. It also has a camera attached for video streaming and detection of AprilTags.
Arduino-Int	An Arduino that reads control signals from *RPI* and transforms these into motor commands which it sends to the platform motors. It also reads data from the wheel encoder sensors and forwards this to *RPI*.
Arduino-Ext	An Arduino used for manual control of Balrog. It reads data from the RC controller and forwards this to *RPI*.

Table 1: Computing hardware on Balrog.

LIDAR	A Scanse Sweep v1 360° scanning LIDAR mounted on top of the robot platform. A range sensor that functions by transmitting a laser beam towards a target and detecting the reflected light. The sensor has a maximum range of 40 m, a rotation frequency of 5 Hz and a sample rate of 1000 samples per second. Used as the primary input sensor for SLAM.
Wheel encoders	Two rotational encoders mounted on each of the two front wheels that turns the tracks of the platform. Used to estimate the distance that each track has travelled, which is utilized in the SLAM module.
Camera	A forward-facing Raspberry Pi Camera Module v2 that is mounted on Balrog and connected directly to *RPI*. It captures video to be used for AprilTag detection and streaming to BaseStation.
IMU	An MTi 100 Inertial Measurement Unit mounted at the midpoint of the robot platform. It communicates with *RPI* via USB and operates at a frequency of 50 Hz. Currently only used to run *RPI* at a fixed frequency, the sensor data is not used.

Table 2: Sensor hardware on Balrog.

The Arduino-Int, wheel encoders and IMU are mounted inside the chassis. The RPI, Arduino-Ext, LIDAR and camera are mounted on top of the metal plate. The computer is placed between the metal plate and the chassis.

To setup the hardware, one needs to make sure that the components are connected according to Figure 2 below. The camera is directly connected to the RPI via a CSI cable whereas the Arduino-Ext, Arduino-Int and IMU are connected to the RPI via USB cable. The computer needs to be connected to the RPI via an Ethernet cable (marked with "1. Connection to RPI" in Figure 2) and to the LIDAR via an USB cable (marked with "2. Connection to LIDAR" in Figure 2).

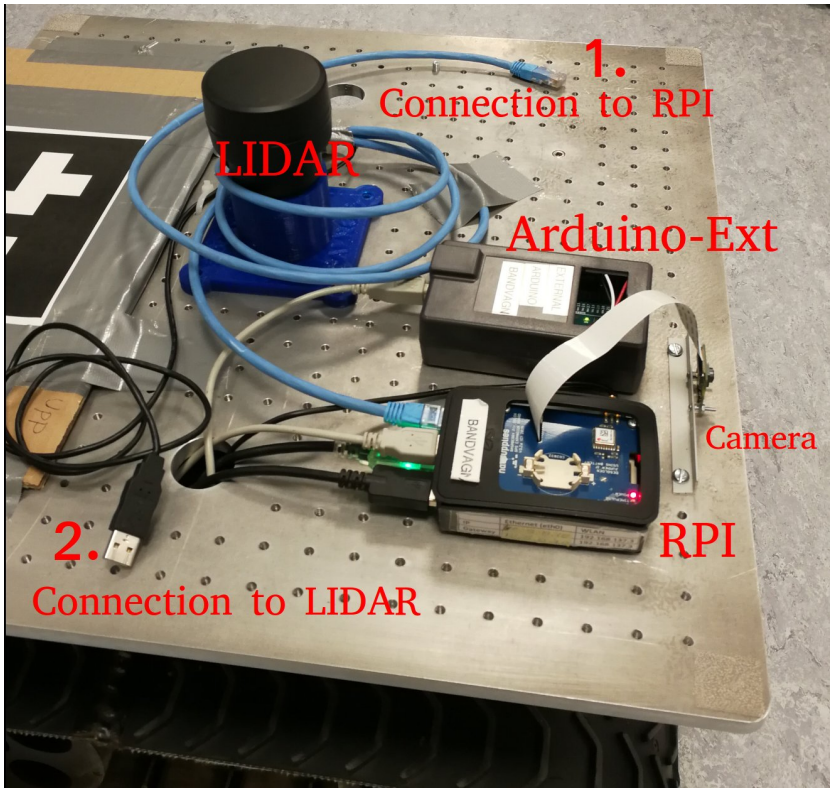

Figure 2: Top view of Balrog with labels for the main components

3.2 Software

All code related to Balrog is found in the minesweeper git repository[1]. The code of main interest, especially for future development, is found in *minesweeper/balrog* which contains all code for the Ubuntu computer that runs the SLAM, navigation and control modules. In this year's project, one of the group members' personal laptop was used for this task. Step-by-step instructions on how to setup the complete software stack on a different computer running Ubuntu 16.04.3 LTS are thus provided in Section 3.2.1 below. These steps have been followed on the industrial PC that has been used in the project in the past, currently there are however some missing parts needed to mount this PC on Balrog.

minesweeper/balrog_sim contains all code and model files needed to run the complete autonomous mission in a simulated Gazebo environment. Most code files found in this directory are simply copies of files from *minesweeper/balrog*. Step-by-step instructions on how to setup any computer running Ubuntu 16.04.3 LTS to be able to run the simulator are provided in Section 3.2.2. These steps have also been followed on the industrial PC, it might however be convenient to have the simulator running on a separate computer as well.

The code related to the RPI is located in various subdirectories of the minesweeper directory. This code was developed in previous years' projects and have only been slightly modified by the current project group. The main file of interest is *minesweeper/main/application.m*. In this file various functions are called, which mostly are located in *minesweep-*

[1] `https://gitlab.ida.liu.se/tsrt10_2017/minesweeper`

er/toolbox. For changes to this code to take effect, one needs to compile the code on a configured Linux computer and then transfer the resulting binaries to the RPI. Instructions on how to configure a computer running Ubuntu 16.04.3 LTS are found in Section 3.2.3, whereas instructions on how to compile and transfer the modified code are provided in Section 3.2.4.

3.2.1 Setup of the Balrog Software Stack

Initial setup:

```
$ sudo apt install git

$ sudo apt-get install python-scipy

$ sudo apt-get install ros-kinetic-image-transport-plugins

$ cd --

$ git clone https://gitlab.ida.liu.se/tsrt10_2017/minesweeper.git
```

Setup ROS and catkin:

```
Install ROS kinetic (not all packages seem to be available for lunar):

    $ 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:80
      --recv-key 421C365BD9FF1F717815A3895523BAEEB01FA116

    $ sudo apt-get update

    $ sudo apt-get install ros-kinetic-desktop

    $ sudo rosdep init

    $ rosdep update

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

    $ source ~/.bashrc

Install catkin (http://wiki.ros.org/catkin):

    $ sudo apt-get install ros-kinetic-catkin

Create a directory called TSRT10 in your home directory:

    $ cd --

    $ mkdir TSRT10

Create a catkin workspace:

    $ cd ~/TSRT10

    $ mkdir catkin_ws

    $ cd catkin_ws
```

```
    $ mkdir src

    $ catkin_make

$ sudo nano ~/.bashrc

Add the line "source ~/TSRT10/catkin_ws/devel/setup.bash" to the
bottom of this file (Ctrl+Shift+v to paste the line,
Ctrl+x - y - Enter to save the file)
```

Setup needed software for the LIDAR:

```
Install the libsweep library from sweep-sdk:

    $ cd ~/TSRT10

    $ git clone https://github.com/scanse/sweep-sdk

    $ cd sweep-sdk/libsweep

    $ mkdir -p build

    $ cd build

    $ cmake .. -DCMAKE_BUILD_TYPE=Release

    $ cmake --build .

    $ sudo cmake --build . --target install

    $ sudo ldconfig

    $ sudo apt-get install ros-kinetic-pcl-conversions

Install the sweep-ros package:

    $ cd ~/TSRT10/catkin_ws/src

    $ git clone https://github.com/scanse/sweep-ros.git

    $ cd ~/TSRT10/catkin_ws

    $ catkin_make

Make sure that your username is in the dialout group in /etc/group
(otherwise you won't have permission to open /dev/ttyUSB0):

    $ sudo nano /etc/group

    In our case (the used username is 'fregu856'), we had to change the line
    "dialout:x:20:" to "dialout:x:20:fregu856"

    Restart the computer ($ sudo reboot)

Now, you should be able to launch sweep.launch (after connecting the
LIDAR to the computer):

    $ roslaunch sweep_ros sweep.launch

sweep2scan.launch and view_sweep_laser_scan.launch will however not work,
for this we need to install the package pointcloud_to_laserscan:
```

```
    $ sudo apt-get install ros-kinetic-pointcloud-to-laserscan

Now, you should be able to launch sweep2scan.launch:

    $ roslaunch sweep_ros sweep2scan.launch

    (you should start receiving a lot of data if you now open a new
    terminal window and type "rostopic echo /scan")

You should also be able to launch view_sweep_laser_scan.launch:

    $ roslaunch sweep_ros view_sweep_laser_scan.launch

    (this should launch rviz where you can see a visualization of the scans)
```

Install the OpenKarto SLAM package:

```
Install the ROS navigation stack in order to get the costmap_2D package:

    $ sudo apt-get install ros-kinetic-navigation

Install SuiteSparse:

    $ sudo apt-get install libsuitesparse-dev

Install Eigen3:

    $ cd ~/TSRT10

    $ wget http://bitbucket.org/eigen/eigen/get/3.2.10.tar.gz

    $ tar -xvzf 3.2.10.tar.gz

    $ rm 3.2.10.tar.gz

    $ sudo ln -s /usr/include/eigen3/Eigen /usr/local/include/Eigen

$ cd ~/TSRT10/catkin_ws/src

$ git clone https://github.com/skasperski/navigation_2d.git

$ cd ~/TSRT10/catkin_ws

$ catkin_make
```

Create and setup the balrog package in the catkin workspace:

```
$ cd ~/TSRT10/catkin_ws/src

$ catkin_create_pkg balrog std_msgs roscpp rospy

$ cd ~/TSRT10/catkin_ws

$ catkin_make

Create a python_scripts directory in the package (it's in this
directory we will place all python ROS scripts):

    $ cd ~/TSRT10/catkin_ws/src/balrog

    $ mkdir python_scripts
```

Every python script that one writes and places in python_scripts (e.g. test.py) must be made executable:

```
$ cd ~/TSRT10/catkin_ws/src/balrog/python_scripts

$ chmod a+x test.py
```

You should always also build the package (this is quite often needed even for python scripts since we use C++ messages):

```
$ cd ~/TSRT10/catkin_ws

$ catkin_make
```

Place all files in ~/minesweeper/balrog/python_scripts in ~/TSRT10/catkin_ws/src/balrog/python_scripts

Make them all executable:

```
$ cd ~/TSRT10/catkin_ws/src/balrog/python_scripts

$ chmod a+x *
```

Copy the directory ~/minesweeper/balrog/launch and place it in ~/TSRT10/catkin_ws/src/balrog

Copy the directory ~/minesweeper/balrog/param and place it in ~/TSRT10/catkin_ws/src/balrog

Copy the directory ~/minesweeper/balrog/src and place it in ~/TSRT10/catkin_ws/src/balrog

Copy the directory ~/minesweeper/balrog/rviz and place it in ~/TSRT10/catkin_ws/src/balrog

Replace the file ~/TSRT10/catkin_ws/src/balrog/CMakeLists.txt with ~/minesweeper/balrog/CMakeLists.txt

Build the package:

```
$ cd ~/TSRT10/catkin_ws

$ catkin_make
```

Install the aprilTags package:

```
$ cd ~/TSRT10/catkin_ws/src

$ git clone https://github.com/RIVeR-Lab/apriltags_ros.git

$ cd ~/TSRT10/catkin_ws

$ catkin_make
```

Configure ethernet connection with the RPI:

```
Connect the computer to the RPI via Ethernet cable

$ ifconfig

Find the entry corresponding to the RPI (disconnect the cable
and run ifconfig again, check which entry disappears) and note its
HWaddr (e.g.: HWaddr 58:00:e3:66:a0:d3)

Click on the Wifi symbol in the toolbar --> "Edit connections..."

Click on "Add" --> choose "Ethernet" --> click "create..."

Enter the noted HWaddr in the "Cloned MAC address" field

Click the "IPv4 Settings" tab, set Method to "Manual"

Click "Add" to add an IP address, enter:

    Address: 10.0.0.20

    Netmask: 255.255.255.0

    Gateway: leave empty

Click "Save" to save the connection and connect to it

Check if you can ping the RPI:

    $ ping 10.0.0.10
```

Setup the ROS IP addresses:

```
(BaseStation is Master and connected to the MINESWEEPER router via
cable, the computer is a slave and connected to the MINESWEEPER
wifi (2.4 GHz))

Connect to the MINESWEEPER wifi

Find the computer's IP address by running ifconfig and looking
for "inet addr", we call this XXX.XXX.X.XX

$ sudo nano ~/.bashrc

Add the following two lines to the bottom of the file:

    export ROS_MASTER_URI=http://192.168.1.59:11311 # (BaseStation)

    export ROS_HOSTNAME=XXX.XXX.X.XX

$ source ~/.bashrc
```

3.2.2 Setup of the Balrog Simulation Stack

Note that some steps in the instructions below, which are all especially marked, are identical to those found in Section 3.2.1 above. Those steps does of course not have to be completed twice.

Initial setup (identical to 3.2.1):

```
$ sudo apt install git

$ sudo apt-get install python-scipy

$ sudo apt-get install ros-kinetic-image-transport-plugins

$ cd --

$ git clone https://gitlab.ida.liu.se/tsrt10_2017/minesweeper.git
```

Setup ROS and catkin (identical to 3.2.1):

```
Install ROS kinetic (not all packages seem to be available for lunar):

    $ 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:80
      --recv-key 421C365BD9FF1F717815A3895523BAEEB01FA116

    $ sudo apt-get update

    $ sudo apt-get install ros-kinetic-desktop

    $ sudo rosdep init

    $ rosdep update

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

    $ source ~/.bashrc

Install catkin (http://wiki.ros.org/catkin):

    $ sudo apt-get install ros-kinetic-catkin

Create a directory called TSRT10 in your home directory:

    $ cd --

    $ mkdir TSRT10

Create a catkin workspace:

    $ cd ~/TSRT10

    $ mkdir catkin_ws

    $ cd catkin_ws

    $ mkdir src

    $ catkin_make

$ sudo nano ~/.bashrc

Add the line "source ~/TSRT10/catkin_ws/devel/setup.bash" to the
bottom of this file (Ctrl+Shift+v to paste the line,
Ctrl+x - y - Enter to save the file)
```

Install the OpenKarto SLAM package (identical to 3.2.1):

```
Install the ROS navigation stack in order to get the costmap_2D package:
    $ sudo apt-get install ros-kinetic-navigation
Install SuiteSparse:
    $ sudo apt-get install libsuitesparse-dev
Install Eigen3:
    $ cd ~/TSRT10
    $ wget http://bitbucket.org/eigen/eigen/get/3.2.10.tar.gz
    $ tar -xvzf 3.2.10.tar.gz
    $ rm 3.2.10.tar.gz
    $ sudo ln -s /usr/include/eigen3/Eigen /usr/local/include/Eigen
$ cd ~/TSRT10/catkin_ws/src
$ git clone https://github.com/skasperski/navigation_2d.git
$ cd ~/TSRT10/catkin_ws
$ catkin_make
```

Install the aprilTags package (identical to 3.2.1):

```
$ cd ~/TSRT10/catkin_ws/src
$ git clone https://github.com/RIVeR-Lab/apriltags_ros.git
$ cd ~/TSRT10/catkin_ws
$ catkin_make
```

Install the simulator:

```
$ sudo apt-get install ros-kinetic-turtlebot
$ sudo apt-get install ros-kinetic-turtlebot-apps
$ sudo apt-get install ros-kinetic-turtlebot-interactions
$ sudo apt-get install ros-kinetic-turtlebot-simulator
```

Set the ROS IP addresses to localhost:

```
(to be able to run the simulator locally, without having to
 connect to any specific network)
Change the lines with ROS_MASTER_URI and ROS_HOSTNAME in ~/.bashrc to:
    export ROS_MASTER_URI=http://localhost:11311
    export ROS_HOSTNAME=localhost
$ source ~/.bashrc
```

Clone and install the repo needed to simulate turtlebot in Gazebo:

```
$ cd ~/TSRT10/catkin_ws/src

$ git clone https://github.com/StanfordASL/asl_turtlebot.git

$ cd ~/TSRT10/catkin_ws

$ catkin_make

Add "export GAZEBO_MODEL_PATH=~/TSRT10/catkin_ws/src/asl_turtlebot/models"
to the bottom of ~/.bashrc ($ sudo nano ~/.bashrc to edit)

$ source ~/.bashrc
```

Test the simulator:

```
$ roslaunch asl_turtlebot turtlebot_sim.launch
```

If everything works as expected, Gazebo should launch and you should see the environment shown in Figure 3. On some computers there seems to be some difficulties launching the simulator, when this happens you might have to try again a couple of times.

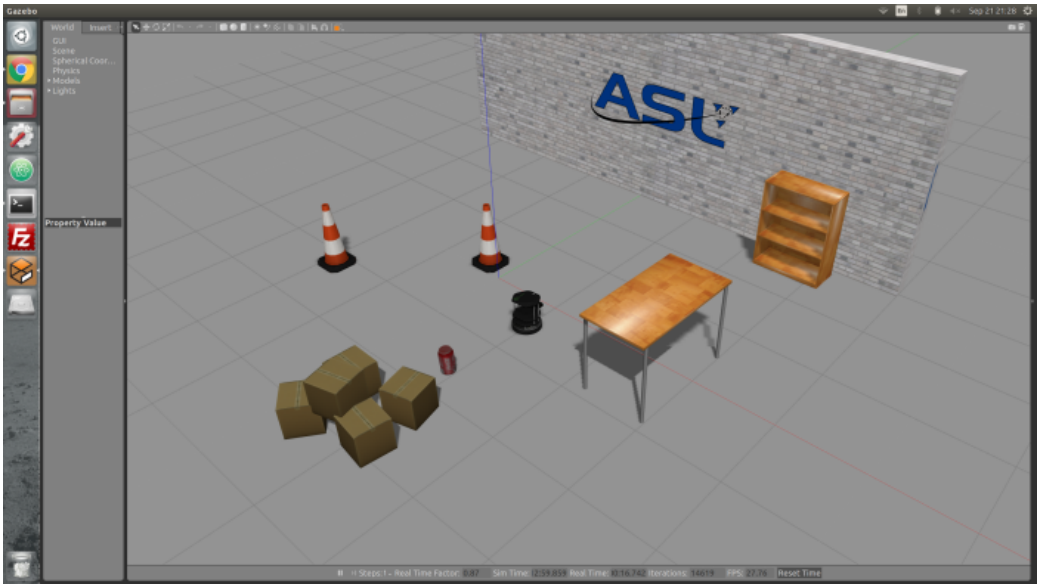

Figure 3: Successful launch of the simulator "Hello world!" environment.

Create and setup the balrog_sim package in the catkin workspace:

```
$ cd ~/TSRT10/catkin_ws/src

$ catkin_create_pkg balrog_sim std_msgs roscpp rospy

$ cd ~/TSRT10/catkin_ws

$ catkin_make
```

Create a python_scripts directory in the package (it's in this directory we will place all python ROS scripts):

```
$ cd ~/TSRT10/catkin_ws/src/balrog_sim

$ mkdir python_scripts
```

Every python script that one writes and places in python_scripts (e.g. test.py) must be made executable:

```
$ cd ~/TSRT10/catkin_ws/src/balrog_sim/python_scripts

$ chmod a+x test.py
```

You should always also build the package (this is quite often needed even for python scripts since we use C++ messages):

```
$ cd ~/TSRT10/catkin_ws

$ catkin_make
```

Place all files in ~/minesweeper/balrog_sim/python_scripts in ~/TSRT10/catkin_ws/src/balrog_sim/python_scripts

Make them all executable:

```
$ cd ~/TSRT10/catkin_ws/src/balrog_sim/python_scripts

$ chmod a+x *
```

Copy the directory ~/minesweeper/balrog_sim/launch and place it in ~/TSRT10/catkin_ws/src/balrog_sim

Copy the directory ~/minesweeper/balrog_sim/param and place it in ~/TSRT10/catkin_ws/src/balrog_sim

Copy the directory ~/minesweeper/balrog_sim/world and place it in ~/TSRT10/catkin_ws/src/balrog_sim

Copy the directory ~/minesweeper/balrog_sim/robots and place it in ~/TSRT10/catkin_ws/src/balrog_sim

Copy the directory ~/minesweeper/balrog_sim/rviz and place it in ~/TSRT10/catkin_ws/src/balrog_sim

Build the package:

```
$ cd ~/TSRT10/catkin_ws

$ catkin_make
```

Install the ros-keyboard package:

```
$ cd ~/TSRT10/catkin_ws/src

$ git clone https://github.com/lrse/ros-keyboard.git

$ cd ~/TSRT10/catkin_ws/

$ catkin_make
```

3.2.3 Configure Linux Computer to Compile RPI Code

```
$ cd --

$ git clone https://gitlab.ida.liu.se/tsrt10_2017/minesweeper.git

$ cd ~/minesweeper/dependencies/cross_compiler

$ sh install_cross_compiler

Reboot the computer
```

3.2.4 Compile & Transfer RPI Code

To apply any changes made to the RPI code, the following steps have to be performed (on a computer configured according to Section 3.2.3):

Power on the RPI

Either connect the computer to the RPI via Ethernet cable, or connect the computer to the Bandvagn wifi

```
[terminal 1] $ ssh pi@10.0.0.10

[terminal 1] (password: cdio2016)

[terminal 1] $ rm main

[terminal 1] $ rm Communicator

[terminal 2] $ cd ~/minesweeper

[terminal 2] $ make

[terminal 2] $ make install

[terminal 2] $ scp -rp target/* pi@10.0.0.10:/home/pi/

[terminal 2] (password: cdio2016)
```

Reboot the RPI

If yo*u do not get the expected behaviour on reboot, you can ssh into the RPI, edit the file startup_ root.sh* such that the main and Communicator programs are not automatically launched on boot, reboot the RPI, ssh into the RPI again and manually start the programs (./main, ./Communicator). You will now see all messages sent to standard output, which highly simplifies debugging.

3.3 BaseStation

BaseStation is used to run the Balrog GUI, which is needed to display sensor data and the SLAM module output, and in order to initialize Balrog's autonomous mode. A picture of the GUI is seen in Figure 4.

3.3.1 Setup of GUI

The following steps, which already have been completed on BaseStation, have to be performed in order to setup the GUI on any Ubuntu computer running ROS:

```
$ cd --

$ git clone https://gitlab.ida.liu.se/tsrt10\_2017/minesweeper.git

Take ~/minesweeper/balrog/rviz/balrog.rviz and save this file as
   "default.rviz" in ~/.rviz ($ cd ~/.rviz, $ nano default.rviz,
   paste everything from balrog.rviz)
```

3.3.2 GUI Usage

The GUI is started by running the following command:

```
$ rqt --perspective-file ~/minesweeper/balrog/gui/balrog.perspective
```

The two plots in the upper right corner of Figure 4 display the wheel encoder sensor data that is transmitted from the RPI. The left half of the GUI contains an rviz window in which all data related to the SLAM module can be displayed, including the map, laser scans and the estimated 2D robot pose. In the bottom left corner one can also choose to display a live video feed from the forward-facing camera on Balrog, including a visualization of all detected AprilTags.

In the *Message Publisher* section, the user can send basic commands to Balrog by publishing strings on two different topics. When put in autonomous mode, Balrog will not initialize the mission until "start" is published on *start_topic*. This is done by clicking the small square next to *start_topic* in the topic list. In this mode, the user can also adjust the speed of Balrog by publishing on the *change_velocity* topic.

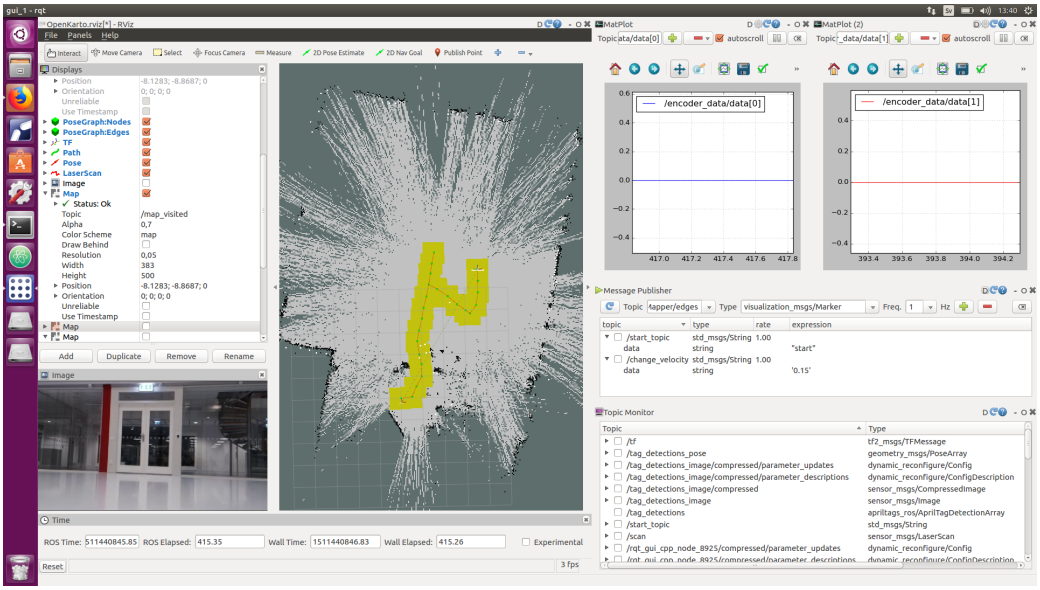

Figure 4: The Balrog GUI.

3.4 Charging

Balrog is charged by connecting the supplied charger to a socket on the rear side of the chassis next to the power-switch, see Figure 5. The socket and charger can only be connected in one way and are both coloured red/black to indicate polarity. The charger has a third connector with a yellow cord which is not used. An LED on the charger will indicate with red when Balrog is actively charging, and with green when Balog is fully charged.

Figure 5: Balrog seen from the rear, with the power switch and charging socket.

3.5 RC Controller

An RC controller is used for manual control of Balrog and to toggle between manual and autonomous mode. The RC data receiver on Balrog is connected to the Arduino-Ext. To turn on the RC controller the switch on the top side, see Figure 6, is moved either to the right ("PWR ON") to see the status of the connection to Balrog and the input readings, or to the left ("DISP") to change the settings.

From previous years' project, Balrog is programmed to always require the RC controller to be turned on to function. This is a safety feature, since if anything unexpected happens, Balrog can always be immediately stopped by turning off the RC controller, i.e. by moving the switch to the "OFF" position.

Figure 6: The RC controller with the ON-OFF switch marked.

A successful connection to Balrog has been established when "No ID" changes and the voltage on the receiver is displayed, see Figure 7. Note that this is not the battery voltage, only the voltage on the Arduinio-Ext which should be constant around 5 V.

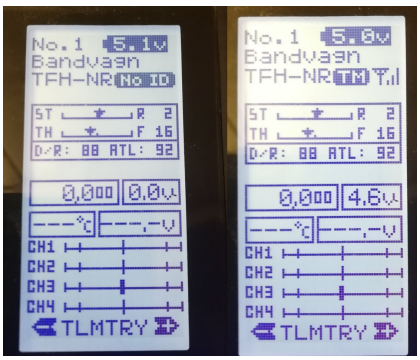

Figure 7: The display of the RC controller with no connection (left) and a successful connection to Balrog (right).

The RC controller has three components being used to send commands to Balrog, see Figure 8. 1 and 2 are used for manual control, where 1 changes the desired angular velocity and 2 the linear velocity. An understanding of how these function is best obtained by testing and keeping in mind that Balrog is sensitive to fast movements. The wheel (3) is used to toggle between manual and autonomous mode. Balrog is started in manual mode and each click of the wheel toggles the mode.

Figure 8: The RC controller seen from the side with the key components marked. 1: Angular velocity (steering) 2: Linear velocity (throttle) 3: Wheel to toggle between manual and autonomous mode.

3.6 Usage

Depending on the scenario and desired mode, the procedure to get Balrog running will differ somewhat. In this section, step-by-step instructions are provided for the most common cases. Known problems are that Balrog does not start correctly which mainly was detected as problem using the RC controller or odometry data seems to be inverted on either sensor. A reboot of the system by powering of Balrog with the switch on the rear side for a few seconds solved all these problems.

3.6.1 Initial Setup

1. Power on Balrog with the switch on the rear side of the chassis. A solid green light next to the switch indicates that the power is on. All components are powered and

all necessary scripts on the RPi are started automatically. Note! The camera feed can only be connected to once, if a script connecting to the feed is restarted, Balrog thus also has to be restarted using the power switch.

2. Balrog takes roughly 15-30 seconds to start. Ensure that the RPI has connection with the chassis by looking at the USB cables attached to the RPI. One has a green light that will flash when data is transferred, see Figure 9.
3. Connect the computer on Balrog to the MINESWEEPER WiFi.
4. Connect BaseStation to the router via Ethernet cable. WiFi works as well, however the bandwidth might be a limiting factor.
5. Make sure that the correct ROS IP addresses are entered in the bashrc file on **both** the computer on Balrog and BaseStation. See Section 3.2.1, *Setup the ROS IP addresses*, for instructions.
6. Connect the computer on Balrog to the RPI via Ethernet cable.
7. Connect the USB cable from the LIDAR to any port on the Balrog computer and wait a few seconds for the LIDAR to obtain full rotation speed.

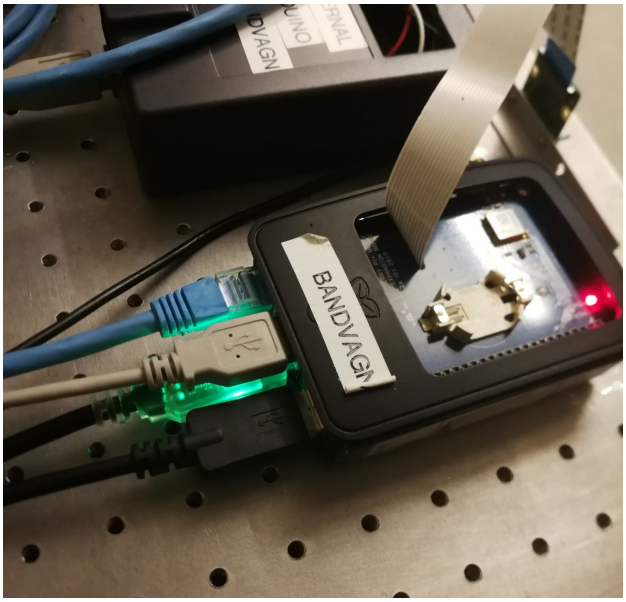

Figure 9: Flashing green light indicating connection between the RPI and IMU.

3.6.2 Running Autonomous Mission

In a complete autonomous mission, Balrog will map the considered area (making sure that there are no unknown, reachable grid cells in the map), cover all free parts of the area (making sure that Balrog has been close to each free grid cell in the map), visit each mine (= AprilTag) that has been detected during mapping and covering, before finally returning to the start position. To launch an autonomous mission, the following steps have to be performed (where "computer" refers to the Ubuntu computer on Balrog):

```
Follow the steps in "Initial Setup"
```

```
[BaseStation] $ roscore

[computer] $ roslaunch balrog balrog.launch

[computer] $ rosrun balrog main.py

[BaseStation] $ rqt --perspective-file ~/minesweeper/balrog/gui/balrog.perspective

Turn on the RC controller and select autonomous mode

[BaseStation] Mark "start_topic" in the Message Publisher section
              of the GUI to start the mission
```

3.6.3 Running SLAM & Manually Control Balrog

If one wants to evaluate the performance of the SLAM module by manually controlling Balrog in different environments, one can simply follow the steps in 3.6.2 above and then put Balrog in manual mode using the RC controller. In this scenario, the process can however be somewhat simplified:

```
Follow the steps in "Initial Setup"

[BaseStation] $ roscore

[computer] $ roslaunch balrog balrog.launch

[BaseStation] $ rqt --perspective-file ~/minesweeper/balrog/gui/balrog.perspective

Turn on the RC controller and control Balrog
```

3.6.4 Manually Control Balrog

If one simply wants to manually control Balrog, in order to e.g. test Sauron's tracking capability, the process described in the above sections can be considerably simplified:

1. Power on Balrog with the switch on the rear side of the chassis (a solid green light next to the switch indicates that the power is on).
2. Wait for the green light on the RPi to start flashing (see Figure 9).
3. Turn on the RC controller and control Balrog (see Section 3.5).

3.6.5 Usage of the Balrog Simulator

A simulated environment has been developed in which the complete autonomous mission can be evaluated, see Figure 10. The user can also manually control the robot in the simulated environment in order to evaluate the SLAM module. It is highly recommended to run the simulator on a computer with a devoted graphics card.

Running autonomous mission:

```
$ roslaunch balrog_sim balrog_auto.launch

$ rosrun balrog_sim main.py
```

Running SLAM and manually control Balrog:

```
$ roslaunch balrog_sim balrog_manual.launch

Click on the small "ROS keyboard input" window and control the robot using WASD
```

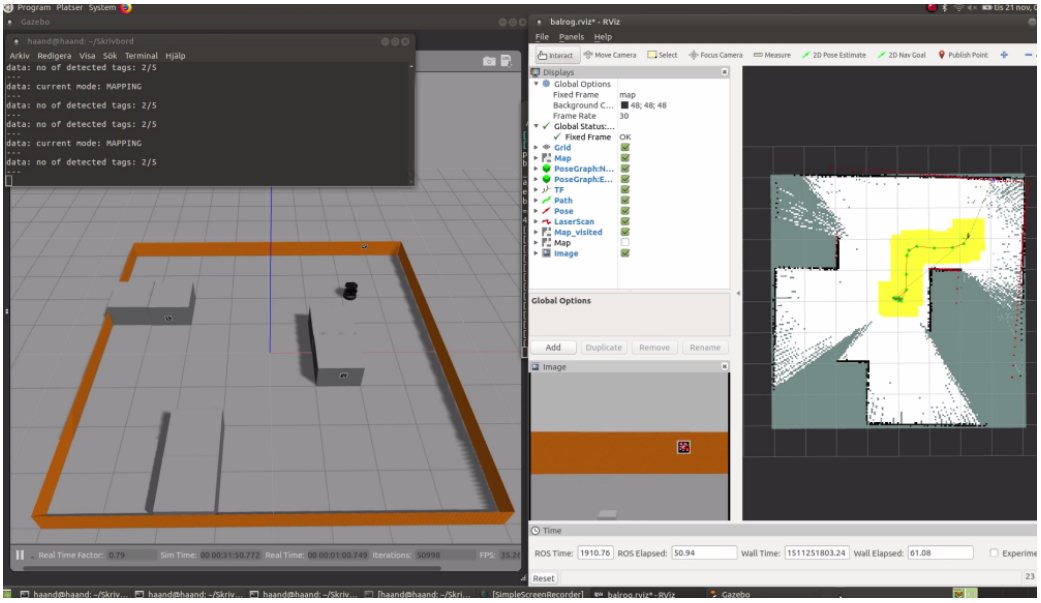

Figure 10: The simulation environment in which the complete mission can be tested.

4 Sauron

In this section all practical information needed to setup, use and further develop the Sauron subsystem is provided.

4.1 Hardware

This section treats the process of setting up the Sauron hardware for flight. An overview of Sauron is seen in Figure 11.

Figure 11: Sauron seen from above.

4.1.1 Remote Controller

A picture of Sauron's remote controller with labels for the used sticks and switches is seen in Figure 12 below. Channel 1 through 4 is used for control of aileron, elevator, throttle and rudder for controlling roll, pitch, throttle and yaw angles. Channel 7 is used for setting Sauron into "Landing Mode". Channel 8 is used to toggle between automatic and manual mode. If the switch is downward when the remote controller is lying flat on the ground, automatic mode is active. Channel 5 is used by the switch operational mode. This switch toggles the modes labeled ”STD”, ”LTR” and ”AUTO”, and the switch next to it ”RTL”, both sharing channel 5. The rest of the switches and channels are not used.

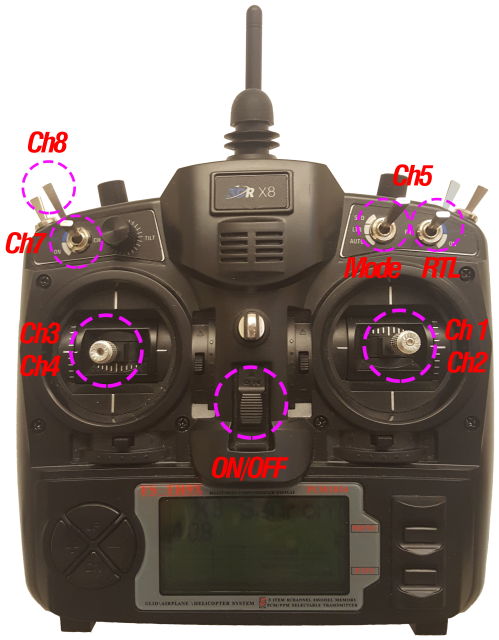

Figure 12: The remote controller and the sticks and switches that are used.

”STD” mode is set to be ”Altitude Hold” in the software. If the throttle stick is 1500 ± 99 in PPM signal from middle position (1401 - 1599), Sauron is holding a constant height using the on-board barometer. However, Sauron will drift a couple of meters in the worst case.

While flying, operational mode ”STD” and ”RTL” = OFF should be used. In order to complete the arming process during initializing, channel 7 must be set on OFF, disabling landing mode. Channel 4, the throttle stick, must be below middle and above its lower setting. Channel 8 should be set to its upper position, manual mode. In order to complete the arming process, the remote controller must be turned on.

4.1.2 Drone

Make sure that the battery is fully charged before flight. About 15-20 minutes of flight-time is expected from a fully charged battery. When installing the battery on Sauron, make sure that the battery is placed as close to the WiFi dongle as possible. This will ensure that the center of gravity is as close to centered as possible. When connecting the battery cable, make sure that the cable is wrapped around something so that it is unable to reach any propeller during flight.

The Raspberry Pi is powered using an UBEC 5 V circuit directly connected to the power rail on the Raspberry. If the RPI is powered from a wall charger, the UBEC connector must be removed. If not, current may flow backwards into the UBEC, ruining it. This goes both if Sauron is powered through the battery and when it is not. See Figure 13.

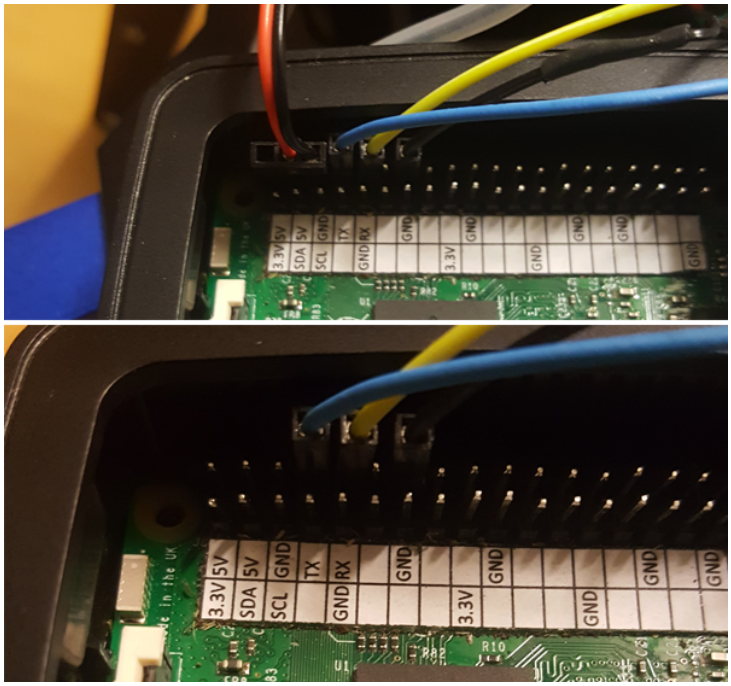

Figure 13: Connector for 5 V UBEC on Raspberry Pi.

On top of Sauron, there is a button with a red light. This button indicates if the ESCs are ready to start spinning the motors. When the red light is flashing, the motors will **never** start spinning. When the red light is continuously red, the motors could start spinning at any time. Exercise extreme caution when approaching Sauron when the red light is continuously lit. To arm and disarm Sauron, press and hold the red button. See Figure 14. There are two levels of arming that both need to be completed before the propellers start to spin. The hardware arming process is described above. The software arming is done in the *pilot_comm.py* script. During this arming check, ArduPilot is checking if all sensors are working as intended and that the settings on the remote controller are set for flight as described.

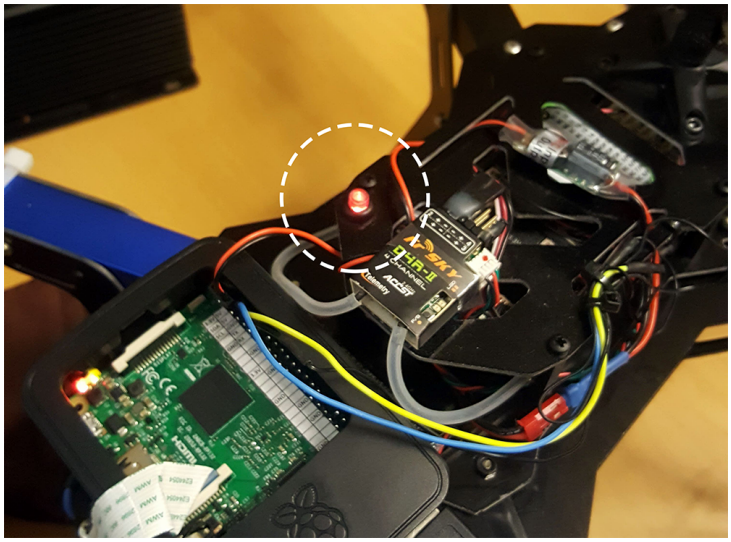

Figure 14: Indicator for arming status of Sauron. Continuously lit indicates armed, flashing light indicates not armed.

During the first flight, it is highly recommended to remove all propellers before start-up. This way, you can test that the video feed is transmitted to BaseStation and that all remote controller commands are working as intended. See the instruction manual of Sauron [2] when remounting the propellers to see which propeller that goes where.

When Sauron is powered there are a number of ”beeps”, all with different purposes. When the remote controller is powered on and off, a signature beep is transmitted. When Sauron is armed and the propellers are about to start, a continuous beep will be transmitted for about 3 seconds. When changing operational modes (should not be done, keep ”STD” mode during flight), a single beep will be transmitted. If the battery voltage indicates that the battery state is low during flight, an intense beeping noise will be transmitted and Sauron will land after about 3 seconds. When this happens, switch to manual mode (channel 8) so that you can manually adjust pitch, roll and yaw so that Sauron can land at a flat spot safely.

If channel 7 is switched to ON, the battery voltage is below failsafe or the remote controller is switched off during flight, Sauron will be put in ”LAND” mode. During this mode, Sauron will drop about 40 cm per second until touching ground. At that point, the propellers may continue to rotate at a slow pace. To stop the propellers, hold the throttle/yaw stick in its lower left position for a couple of seconds. The arming and disarming process is further described in the instruction manual for Sauron [2]. If for some reason holding the throttle/yaw stick in its lower left position does not stop the propellers from rotating then either switch channel 7 to "ON" (enabling "LAND" mode) or turn off the RC controller. Note that turning the RC controller off should of course only be done when Sauron is on the ground.

4.1.3 BaseStation

Make sure to plug in the power adapter during flight. Doing so will ensure that the processor is using its full potential, minimizing lag. BaseStation should be connected with Ethernet cable to the router as WiFi will greatly reduce the performance. If a cable is not available make sure to connect via a 5 GHz WiFi (N or AC standards should both be fine).

4.1.4 Router

Place the router close to where you will fly with Sauron. Sauron is connected to the router on the 5 GHz band and BaseStation is connected to the router using cable. Make sure that there are no obstacles in-between Sauron and the router.

4.2 Software

The communication between Sauron and BaseStation is established over WiFi using ROS. In the ROS network, different nodes communicate with each other by subscribing and publishing information to different topics.

4.2.1 Sauron

Note that everything done in this section is done on Sauron (on the Raspberry Pi mounted on the drone).

The commands sent to PixHawk from the Raspberry Pi onboard Sauron are transmitted using the *DroneKit-Python API* that uses *MAVLINK* as the communication protocol. A whole list of parameters can be found at [3], and for examples of DroneKit implementations in Python see [4]. Both of these links will make it easier to understand the current Python code as well as to further develop Sauron.

The software needed on the Raspberry Pi to be able to get the system up and running is ROS Kinetic on an Ubuntu Mate clean install, catkin, Python, DroneKit and raspicam_node. These are already installed on the Raspberry Pi. If any complications with the software arise, the step-by-step guides below explain how to install the software needed. Start off with a clean install of Ubuntu Mate. Note that there should be a backup image of the operating system available.

Installing ROS kinetic (http://wiki.ros.org/kinetic/Installation/Ubuntu)

```
$ 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:80 --recv-key
  421C365BD9FF1F717815A3895523BAEEB01FA116

$ sudo apt-get update

$ sudo apt-get install ros-kinetic-desktop

$ sudo rosdep init

$ rosdep update

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

$ source ~/.bashrc
```

Installing Catkin (http://wiki.ros.org/catkin)

```
$ sudo apt-get install ros-kinetic-catkin
```

Create a catkin workspace:

```
$ mkdir catkin_ws
```

```
$ cd catkin_ws

$ mkdir src

$ catkin_make

$ sudo nano ~/.bashrc

$ sudo echo "source ~/catkin_ws/devel/setup.bash" >> ~/.bashrc
```

Installing Python

```
Install some dependencies:

    $ sudo apt-get install build-essential checkinstall

    $ sudo apt-get install libreadline-gplv2-dev libncursesw5-dev libssl-dev
   libsqlite3-dev tk-dev libgdbm-dev libc6-dev libbz2-dev

Download python:

    $ cd ~/Downloads/

    $ wget https://www.python.org/ftp/python/$version/Python-$2.7.12.tgz

Extract python:

    $ tar -xvf Python-$2.7.12.tgz

    $ cd Python-$2.7.12

Install python:

    $ ./configure

    $ make

    $ sudo checkinstall
```

Installing DroneKit-Python

```
    $sudo apt-get install python-pip python-dev

    $sudo pip install dronekit
```

When all the necessary software has been installed follow the steps below to get Sauron up and running.

Adding Code from Git

```
Begin with cloning the entire git repository to a folder of your choice:

$ cd directory_you_wish_to_clone_to

$ git clone https://gitlab.ida.liu.se/tsrt10_2017/minesweeper.git

You now have a folder containing all the code from git
```

Setting up catkin_ws on Sauron RPi

```
$ cd
$ cd catkin_ws/src
Create the pilot package:
    $ catkin_create_pkg pilot std_msgs roscpp rospy
    $ cd ~/catkin_ws
    $ catkin_make
    $ cd src/pilot
    $ mkdir scripts
    Navigate to the Onboard RPi catkin_ws/src/pilot/scripts from the git:
    /minesweeper/Sauron_2017/Sauron_OnBoardRPi/catkin_ws/src/pilot/scripts
    Copy everything in this folder (ctrl + a & ctrl + c)
    Place these files in the your own catkin_ws/src/pilot/scripts
        Check if the files are executable
        $ cd scripts
        $ ls
        If all the python files are green you are good to go
        If not type "chmod a+x file_name.py" for each non-green file

    Navigate to the Onboard RPi catkin_ws/src/pilot from the git:
    /minesweeper/Sauron_2017/Sauron_OnBoardRPi/catkin_ws/src/pilot
    Copy the folder "launch" and place it in your own catkin_ws/src/pilot
    Copy the file CMakeLists.txt and replace the CMakeLists file in
    your own catkin_ws/src/pilot

    $ cd ~/catkin_ws
    $ catkin_make
$ cd ~/catkin_ws/src
Create the raspicam_node package:
    From: https://github.com/UbiquityRobotics/raspicam_node
    Also see the link above for camera calibration!
    $ git clone https://github.com/UbiquityRobotics/raspicam_node.git
    $ /etc/ros/rosdep/sources.list.d/30-ubiquity.list
$ yaml https://raw.githubusercontent.com/UbiquityRobotics/rosdep/master/raspberry-pi.yaml
    $ rosdep update
    $ cd ~/catkin_ws
    $ rosdep install --from-paths src --ignore-src --rosdistro=kinetic -y
    $ catkin_make
    Now navigate to the Onboard RPi catkin_ws/src/raspicam_node from git:
    Copy the folders "launch" and "camera_info"
    then replace them in your own catkin_ws/src/raspicam_node
    $ catkin_make
```

4.2.2 BaseStation

Note that everything in this section is done on BaseStation (laptop).

Running the command: *"$ roslaunch sauron sauron.launch"* starts four ROS nodes. If no *roscore* has been started before, *roscore* will be started as well. The nodes started are:

- *Republish* - Decompresses JPEG images from topic /raspicam_node/image to RAW images on topic /usb_cam/image_raw .
- *apriltags* - Detects AprilTags from RAW images on topic /usb_cam/image_raw.
- *sauron* - Starts the *"sauron.py"* script. Reads from /apriltags/marker_array which includes all data for the detected AprilTags. Manipulates data and publishes on topic /sauron/output.
- *pid* - Starts the *"pid.py"* script which subscribes to /sauron/output. Calculates steering commands for autonomous drive for Sauron. Publishes to /sauron/pid.

Installing ROS kinetic (http://wiki.ros.org/kinetic/Installation/Ubuntu)

```
Follow the steps from above (Section 4.2.1 Sauron)
```

Installing Catkin (http://wiki.ros.org/catkin)

```
Follow the steps from above (Section 4.2.1 Sauron)
```

Installing Python (http://wiki.ros.org/catkin)

```
Follow the steps from above (Section 4.2.1 Sauron)
```

Installing Xenobot AprilTag library and wrapper for ROS including usb_cam for testing purposes (https://github.com/xenobot-dev/apriltags_ros)

```
Install libcgal:

    $ sudo apt-get --yes install libcgal-qt5-dev libcgal-dev

Install apriltags-cpp and building:

    $ cd ~

    $ git clone https://github.com/xenobot-dev/apriltags_ros.git

    $ cd ~/apriltags_ros/software/apriltags-cpp

    $ mkdir build

    $ cd build

    $ cmake .. -DCMAKE_BUILD_TYPE=Release

    $ make

    $ sudo make install

Install apriltags and usb_cam in ROS:

    $ cp -r ~/apriltags_ros/apriltags_ws/src ~/catkin_ws/src
```

```
$ cd ~/catkin_ws
$ catkin_make
$ rm -r ~/apriltags_ros
```

Installing OpenCV (https://www.learnopencv.com/install-opencv3-on-ubuntu/). OpenCV is used by the Apriltag package.

```
Install OpenCV according to the guide linked.
```

Adding Code from Git

```
Follow the steps from above (Section 4.2.1 Sauron)
```

Setting up catkin_ws on the Base station

```
$ cd ~/catkin_ws/src
$ catkin_create_pkg sauron std_msgs roscpp rospy
$ cd ~/catkin_ws
$ catkin_make
Create a python_scripts directory in the package
$ cd ~catkin_ws/src/sauron
$ mkdir python_scripts
Copy the python files from the git clone:
minesweeper/Sauron_17/Sauron_BaseStation/catkin_ws/src/python_scripts
and paste them in your own catkin_ws/src/sauron/python_scripts
    Check if the files are executable
    $ cd python_scripts
    $ ls
    If all the python files are green you are good to go
    If not type "chmod a+x file_name.py" for each non-green file
Copy the directory "launch" from the git clone and place it in
your own catkin_ws/src/sauron
Copy the file CMakeLists.txt and replace the CMakeLists file in
your own catkin_ws/src/sauron
$ cd ~/catkin_ws
$ catkin_make
```

Practical information regarding the apriltags The tag used is tag number 12. If you want to print a new tag in a different size you need to update the apriltags.launch file to reflect this change. The parameter

```
"~default_tag_size" value="0.XXX"
```

sets the tagsize in meters. Currently two different tagsizes have been used, 0.235 for stationary target and 0.165 for tracking Balrog.

4.2.3 GUI

Running the command *"$ roslaunch sauron sauron.launch"* will bring up a window displaying the video feed and highlight detected apriltags. In the command window, relevant information will be printed.

4.3 Usage

To fly Sauron first you need to connect the hardware as described in the Hardware section, but do not arm it yet. Setup BaseStation and the router (ASUS RT-AC51U) as close to where you intend to fly as possible whilst being safe from harm if anything would go wrong. Make sure that BaseStation's power is plugged in.

- Connect the Ethernet cable from BaseStation to the router. A solid blue light should be seen on the router's Ethernet port.
- Start the hand controller and set all switches in the correct positions (see section Hardware).
- Place Sauron in the middle of the intended flight area and then arm it by pressing down the red flashing button until it is solid red.
- Do the following steps in a terminal on BaseStation to setup for flight:

```
Start roscore:
[terminal 1] $ roscore
Open a new terminal and ssh to the RPi on sauron to start raspicam_node via the
launch file of your choice, we recommend using the 600x600 pixels at 15 fps.
[terminal 2] $ ssh sauron@192.168.1.188
[terminal 2] $ roslaunch raspicam_node 600x600_15fps.launch
Open a new terminal window and start the nodes:
apriltags, Republish, sauron and pid
[terminal 3] $ roslaunch sauron sauron.launch
This will open a new window 600x600 in size displaying the apriltag
recognition.
```

- For the last step before everything is ready for flight you need to open another terminal window and ssh to Sauron. Make sure that the throttle is set low (below 50%)! When running the command the motors will start spinning after 15-30 seconds **Do not approach Sauron!**

```
[terminal 4] $ ssh sauron@192.168.1.188
[terminal 4] $ rosrun pilot pilot_comm.py
```

Lift off is always done manually! Take off and make sure to get a good feeling of the controls as you want to quickly take back control if the controller in Auto mode should act strange or if Sauron for some other reason loses sight of the AprilTag.

If you can't connect to Sauron via ssh, Sauron and BaseStation most likely are not on the same network. First make sure that BaseStation's Ethernet cable was correctly plugged in and make sure that the router is turned on. If BaseStation seems to be connected properly you will need to disarm Sauron and plug it into a screen, mouse and keyboard. Then on the Sauron RPI make sure that the WiFi it automatically connects to when turned on is the provided router (MINESWEEPER_5G)

4.4 Getting Started Quick Guide

Please read the full Users manual before using the quick guide. **Please use caution when operating Sauron! Avoid flying in populated areas. Make sure you are flying in accordance with local regulations and laws.**

Router information:

Network name:	MINESWEEPER_5G
Default password:	cdio2017
192.168.1.1 username:	admin
192.168.1.1 password:	cdio2017

Step-by-step guide to get in the air:

1. Make sure that the provided router is plugged in.
2. Start BaseStation (laptop) and connect to the network with an Ethernet cable.
3. Start the Raspberry Pi on Sauron by connecting the battery to Sauron. The RPI is set to automatically connect to the correct network on a static IP address.
4. Start a roscore on BaseStation by opening the terminal and prompt the command: **roscore**
5. Open a new tab or a new terminal and start the AprilTag detection and PID controller with the command: **roslaunch sauron sauron.launch**
   (a) If you want to read the information in the system (distance from the AprilTag and regulator output), open a new terminal and prompt **rosrun sauron pid.py**
6. Turn on the remote controller and make sure manual mode is set (channel 8), ”STD” mode is active, ”RTL” and channel 7 is set to OFF. Also set the throttle stick somewhere between its middle and its lower position.
7. Access the Raspberry Pi on Sauron via BaseStation using *ssh* in a new terminal tab or window. command: **ssh sauron@192.168.1.188**
   Enter the password for Sauron: **cdio2017**
8. Lastly, arm Sauron by pressing down the red flashing button, figure 14. Move away to a safe position.
9. Start Sauron from the terminal you just opened. It takes a couple of seconds for the arming process to complete so please be patient and do not approach Sauron. If a window displaying the video feed does not open, something is off. The propellers may start to spin anyway. command: **roslaunch pilot pilot.launch**
10. Lift off by moving the throttle stick a bit above the middle position.
11. Only put Sauron in automatic mode when Sauron has a clear view of an apriltag with ID 12.

References

[1] CDIO 2016 TIGER. User manual, 2016.

[2] 3D Robotics. 3dr x8+ operation manual, 2017. URL https://3dr.com/wp-content/uploads/2014/11/X8+-Operation-Manual-vA.pdf.

[3] Ardupilot development team. Ardupilot - complete parameter list, 2016. URL http://ardupilot.org/copter/docs/parameters.html#arming-check-arm-checks-to-peform-bitmask.

[4] 3D Robotics. Dronekit-python api, 2015-2016. URL http://python.dronekit.io/guide/index.html.