AeroWorks Documentation

Release 0.1 Alice

Alex Kamphuis, Cees Trouwborst, Robin Hoogervorst

November 21, 2016

Contents

VIKI is software aiming to enable the quick and easy start up of experiments involving ROS. Development has started in 2014 at the University of Twente under funding of project Aeroworks. VIKI can help a student that is new to a experimental setup to quickly start where the previous contributor stopped. For example: if you are designing a position-controlled quadrotor, based on on-board camera and IMU info, you can immediately start a position-controlled drone based on external cameras, redo/validate the experiments of the researcher who created that setup, and focus on your research instead of on peripherals.

The repository can be found on github: http://www.github.com/UT-RAM/viki. If you have any issues, please contact *The developers* or place on issue on github.

This documentation is split up into three parts, each having its own goal:

1. *User documentation* aims at the regular VIKI users which are not developing their own modules. This starts with *Installation* and Quick start guide to get up and running with VIKI.
2. *Custom module development* aids people at writing their own custom modules for VIKI
3. *Developer documentation* is for internal documentation on VIKI or for people wanting to know more about the internals of VIKI.

CHAPTER 1

Contents

1.1 User documentation

This part of the documentation is focused on the using VIKI itself, not on developing modules or contributing to VIKI itself. Are you looking to write a module? Please have a look at *Custom module development*. Are you looking for internal documentation on VIKI? Look at *Developer documentation*.

This documentation will guide you through the main process of using the GUI and launching your first applications.

1.1.1 Installation

This installation guide assumes you have installed ROS already. If you have not done this, follow this guide to install it.

If you have already a catkin workspace, VIKI can be installed in that workspace. If not, you have to create one using this guide. It is also possible to create a new workspace, just for the VIKI modules or even extend your old workspace.

If you want to have the most clean configuration, we suggest creating a new workspace for VIKI. ROS packages that are not defined as a module can not be used anyway in VIKI and setting up a new workspace allows for a clean installation with up-to-date package versions. If you just want some functionality from the modules of VIKI, while not using the GUI, you can also just get a module repository and use ROS packages directly in your base workspace.

VIKI consists of a core and at least one module repository. Right now, there is only one main repository, but this design makes it easy to create your own repository of modules for use easy sharing. This installation will setup the core and the main module repository for you. Adding extra module respositories is quite easy and discussed in *Module repositories*.

Requirements

VIKI uses ROS for all its main functionality and therefore builds on top of those dependencies. In line with ROS, VIKI only supports Ubuntu as its platform and is tested on LTS versions 12.04 and 14.04. Other Ubuntu (and even Linux) distributions may work just as well, but are not supported.

Installation

Installation is as easy as pulling the repository into your (or a new) catkin_workspace. The repository can be found on https://github.com/UT-RAM/viki/. VIKI is a standalone tool and therefore, by default, does not live in your catkin workspace. We recommend installing it in your home folder, but you're free to install it where you like.

```
cd ~
git clone https://github.com/UT-RAM/viki
```

This will create a new folder /home/<user>/viki, where all the VIKI magic lives.

Configuring VIKI

VIKI has a self-configuring tool, which setups the folder directories for you. This is done by launching the configure application from the terminal.

```
cd <viki_dir>
./viki configure
```

Running this will install some extra dependencies, setup a config.json file for the right directories and setup a .desktop file such that you can launch VIKI from the Unity Dash.

Module repositories

To be able to really use VIKI, a module repository needs to be added. VIKI works with module repositories, such that you can get started quickly by automatically adding a set of packages. Using the command line, you can add the default repository as follows

```
./viki add-module-repository
```

This will pull the repository and automatically install all dependencies that might be needed for the modules so they can be launched. Furthermore, VIKI should be able to run catkin_make instantly, but this does not work as well,

ROS package dependencies

If you followed the steps above, you should be all set. VIKI tries to handle all its ROS package dependencies by using the rosdep tool. However, since this is not optimized for different ROS versions, this may sometimes not work that well. To get an overview of what your missing packages are, run

```
./viki check-packages
```

based on this, you might find out you're missing the *libuvc_camera* package when using ROS jade. Since this package is not available for jade, it may not automatically install. To install this by hand, you can run

```
sudo apt-get install ros-indigo-libuvc-camera
```

as the package is available for ROS indigo. It should not be that hard to figure this out for your packages and we are working to find a way to neatly resolve this. In the meantime, if you're still experiencing any problems, please do not hesitate to contact *The developers*

1.1.2 Getting started

This guide assumes you have installed VIKI succesfully. If you have not, check out *Installation*. Using this guide you can run your first setup: controlling a turtle-simulator with your keyboard.

Launching VIKI

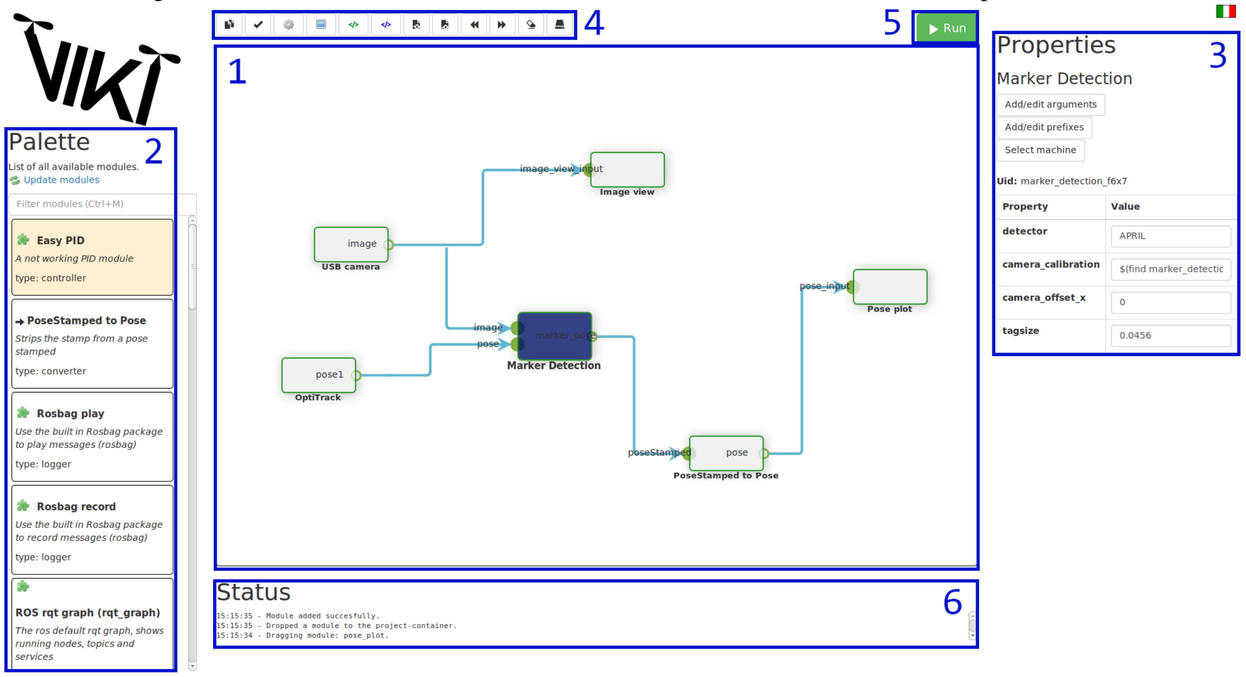
First off, you will have to start VIKI. Open a terminal, navigate to the installation folder of viki (usually *~/catkin_ws/src/viki*) and execute

```
./viki run
```

You should see a screen pop up with the interface of VIKI.

A quick look at the interface

After launching VIKI, we can take a look at the interface and the elements that the interface provides:



Palette

List of all available modules.

Update modules

Filter modules (Ctrl+M)

Easy PID
A not working PID module
type: controller

PoseStamped to Pose
Strips the stamp from a pose stamped
type: converter

Rosbag play
Use the built in Rosbag package to play messages (rosbag)
type: logger

Rosbag record
Use the built in Rosbag package to record messages (rosbag)
type: logger

ROS rqt graph (rqt_graph)
The ros default rqt graph, shows running nodes, topics and services

Properties

Marker Detection

Add/edit arguments

Add/edit prefixes

Select machine

Uid: marker_detection_f6x7

Property	Value
detector	APRIL
camera_calibration	$(find marker_detectic
camera_offset_x	0
tagsize	0.0456

Status

```
15:15:35 - Module added succesfully.
15:15:35 - Dropped a module to the project-container.
15:15:34 - Dragging module: pose_plot.
```

1. **Canvas**: This is where you will build your project.
2. **Module Palette**: This shows all the modules that are available.
3. **Module specifics**: Once a module is selected, this pane shows all the parameters that are available for this module. Furthermore, there are three buttons to edit arguments, prefixes and select the machine that this module will run on.
4. **Toolbar**: Here are the basic actions that you will need during use. Hover over each button to see its purpose
5. **Run button**: Press this button to run the setup you created.
6. **Status pane**: This shows some status information, which can basically be used for debugging.

People that are familiar with for example Simulink probably recognize this layout. You can search for modules that you need in your project in the palette, when you have found a module that you need, you can drop it to the canvas, connect it with other modules and run your project to see all the magic happen. During the next few steps, we will guide you through your first launch.

Launching your first project

In this quick guide, I will guide you through launching your first project with VIKI using the turtlesim. We need two modules for this setup: the turtle simulator and a module that interprets input from your keyboard and sends it to the simulator. Below the VIKI-logo you can search for modules by clicking in the textbox and entering *turtle*. You will see two modules called:

- turtle_teleop_key
- turtlesim node

You need both of these. Click and drag each of these modules to the canvas. You should now see two loose modules visible on your canvas.

Now that we have the two modules, we need to connect them. We will need to connect the output of *turtle_teleop_key* to the input of *turtlesim node*. You can do this by dragging the teleop's output node to the turtle's input node. An arrow will appear that indicates the direction of information. Notice that you can not start dragging at the turtle's input node: you need to follow the direction of information.

Your setup is now ready. Hit the green *run* button on top of the screen. A new terminal should open that gives some text feedback. This terminal is used to open your complete setup. More importantly, you should also see a window with a turtle in it. If you focus your input on the text window (click somewhere on it) then you can use your arrow keys to control the turtle in the other window.

When you're done playing around with the turtlebot, you can select the terminal window and press *Ctrl + C*, which will kill its processes. After gracefully shutting down, the terminal window will disappear, and you're free to again run your canvas setup.

Where to move from here?

Know that you know the basic functionality of VIKI, you're basically ready to go! Probably, you'll want to write some own specific functionality by adding your own modules. If this is the case, you can go to the Contributor page to see more information about writing your own module. For more functionality, you can check out the *Features* page or take a look at *Distributed systems (SSH documentation)* functionality. If you want to learn more about the vision, background and core of VIKI, or want to contribute to the core, you can visit the *Developer documentation* page.

1.1.3 Commandline Interface

VIKI comes with a command line interface. During installation, you have already used this. This command line interface is mainly used for dependency management and installing ROS packages that are needed for VIKI modules.

The following commands are available

run Runs VIKI

configure Sets VIKI up for the first time. This installs dependencies, places VIKI in your PATH and creates a desktop entry, so VIKI can be launched easily

check-packages This will check if ROS dependencies are met. First it will look for ROS packages that are not installed, but that are needed for the module repositories that are installed. Furthermore, it will use the rosdep tool to install second level dependencies of ROS packages.

install-packages This will automatically install all packages that are missing, according to check-packages. Normally, this tool will install the packages using apt-get, but if that is not available, it will pull the dependencies from source

add-module-repository This will add another module repository. Right now, it will always pull the core repository, but this is te be adapted soon.

To see this information, you can also run the viki help command.

```
./viki --help
```

1.1.4 Features

VIKI aims at making software for robotics easier to run and exchange between the community.

Currently supported

VIKI supports a wide set of features

- Build a graphical model of your robot software with modules based on ROS
- Save/Open build canvas models
- Add extra commandline options to modules on the canvas
- Extend functionality by easily creating your own modules
- Export your build model to a launch file
- Automatically launch nodes on a different machine with SSH

Of course, VIKI supports (at least the basic) functionality of ROS features.

- Supply parameters for launch, which can be edited directly in the GUI as well
- Dependency management on the modules, based on rosdep, rospack and git repositories
- Many2Many communication between modules, also to be edited easily in the GUI

Future goals

VIKI is continuing to be improved and different features get added constantly. Functionality that has not made it yet to its release includes

- Communication via services
- Communication via topics that are not created by nodes in VIKI
- ROS multimaster support

We're working on bringing these features into VIKI as well! If there's any functionality you would like to see, contact *The developers* and we will (most likely) help you out quickly!

1.1.5 Distributed systems (SSH documentation)

VIKI allows you to run nodes on several computers, it does not even be installed on the other systems. Since VIKI builds on top of ROS, the distributed functionality of ROS can be used. For this to work, there are some network requirements that can be tedious to setup. This guide helps you in setting up these requirements in order for the network to work. Below there is one way to set up your distributed system, there are however more possibilities (if you are an advanced user).

Setting up the network

ROS works by running a ros master, or roscore, on one system. All other systems connect to it via it's location that is saved in the *ROS_MASTER_URI* environment variable. Other computers on the network are found by using their name. Furthermore all ports need to be open as ROS opens a new connection for every topic on a random port.

First we make sure we can communicate bidirectionally between systems. As an example take a computer named Ash and a computer named Pikachu. Ash will be the main computer where we run VIKI. Pikachu will be the remote computer (with ROS, but possibly without VIKI). There can be more computers like Pikachu (with different names!) but in this guide we will describe just one.

Network connection

Make sure your computers are connected to the same network. This can be done by connecting to the same wireless or wired network. If you're running ROS in a VM, like VirtualBox, this can be achieved by adding an extra 'internal' network adapter.

Finding computer names

Find the name of your computers. In this case we know that they are Ash and Pikachu. If you are unsure of your computer name you can find it by typing

```
hostname
```

in a terminal. It will return the name of your computer.

Finding IP adresses

Now we need to find the IP adresses of both computers. Open a terminal and run

```
ifconfig
```

You will get some feedback on the network connections that you have open. Find the adapter that is connected to the network you are going to use and look for *inet addr:*. Behind it will be a number that looks like *192.168.1.10*. Keep these values in mind, since we will need them for configuration later on.

Note: IP addressses of computers are usually dynamic, meaning they will change over time. To keep yourself from reconfiguring this all the time, you might also want to make both systems have a static IP address. For ubuntu, some more information can be found here.

Setting hostnames

To make sure the computers can find each other, they need to be able to resolve the hostname to an IP address. When there is a DNS running on your network, you might use this to resolve the hostnames. If not, you will have to add records to the hosts file of each computer. To do this:

Open a new terminal and start editing the hosts file, e.g. run

```
sudo gedit /etc/hosts
```

to edit the file with gedit.

The top line of this file says something like

```
127.0.0.1       localhost
```

This line resolves the 'localhost' to 127.0.0.1. For every computer, we will need such an entry as well. For each computer in the network that you want to use, add a line that has the IP address of the computer, some space separation, and it's hostname. In our case the first three lines of the file will look like this

```
127.0.0.1               localhost
192.168.1.10    Ash
192.168.1.20    Pikachu
```

Testing

When these steps are defined, the computers should be able to find each other. You can test if you found the right IP adress and if communication is possible by running

```
ping 192.168.1.20
```

with *192.168.1.20* replaced by the IP adress of the computer you want to reach. You can test if your changes to */etc/hosts* has worked by running

```
ping Pikachu
```

from Ash, or the other way around. Both should give you information about succesfully sent packages. If not, recheck your network and make sure you have the right IP's.

Setting up SSH

ROS uses the SSH protocol to run things on other systems. For it we need a username and password on the machine on which we want to run things. These are the username and password you would normally use to log in on the computer.

If it is not yet previously installed you need to install *openssh-client* on the main PC (Ash) and *openssh-server* on the remote. You can do this by running one of the following two lines and accepting the questions

```
sudo apt-get install openssh-client
sudo apt-get install openssh-server
```

Testing SSH

Now test if you can make an SSH connection to the other machine. Open a terminal on Ash and run

```
ssh username@Pikachu
```

where you replace *username* by the username you would normally use to log in on Pikachu. The terminal asks you if you want to connect, reply yes and fill in your password when asked. You should now see username@Pickachu in front of you terminal entry-area in stead of anotherusername@Ash (which is the normal situation). This means you are now 'in' Pikachu. You can exit by typing

```
exit
```

and you will return to the normal terminal (on Ash).

Adding to known_hosts

Your computer keeps a list of computers you can connect to called *known_hosts*. To make ROS able to connect to another computer you need to add it to the list. Open a terminal and run

```
ssh-keygen -R [hostname]
ssh-keygen -R [ip_address]
ssh-keygen -R [hostname],[ip_address]
ssh-keyscan -H [hostname],[ip_address] >> ~/.ssh/known_hosts
ssh-keyscan -H [ip_address] >> ~/.ssh/known_hosts
ssh-keyscan -H [hostname] >> ~/.ssh/known_hosts
```

This will add both the username and the ip adress and the combination ot the known_hosts file, so that you can acces it either way.

At this point you should be able to run anything you want on the remote computer by using VIKI. However by default SSH session are not 'visible' on the remote PC. They just run in the background. To for instance open a screen on the remote machine you need to run nodes in it's X server. VIKI uses an environment loader for this.

Creating the environment file

On Pikachu create a file called *.viki_env* in the home folder for instance via

```
gedit ~/.viki_env
```

and add to it the following:

```
#!/bin/bash
. /opt/ros/jade/setup.sh
DISPLAY=:0; export DISPLAY
exec "$@"
```

replace *jade* by the ros distribution on Pikachu, then save and exit.

Testing the setup (tutorial)

Everything should now be set up. We can test the setup with a simple configuration of VIKI. For instance:

1. Start VIKI
2. Add a turtle sim node
3. Add a turtle teleop node
4. Connect them
5. Click the 'machin list' icon (top of screen in VIKI)
6. change the localhost to the name of the computer running VIKI (Ash in above example)
7. add a machine, give it a name and fill in the adresses/username and password. In our case: name Pikachu, adress Pikachu
8. Save and close
9. Click the turtlesim node, and in the properties screen (right side) use select screen to make it run on Pikachu rather then default (local)
10. Select the teleop node and add a prefix *xterm -e* to make it open in a new terminal

11. Click run

This setup will open a roscore on the current machine, a turtlesim node on Pikachu, and a teleop node on Ash. Run RQT to see everything works (remember to refresh after the you have sent a few messages) or just run around with your turtle trying to draw a Pikachu.

1.2 Custom module development

This part of the documentation is aimed at people writing their own modules. If you just want to use viki, take a look at *User documentation*, or if you're interested in internal documentation, have a look at *Developer documentation*.

As you probably already know, VIKI modules provide the core functionality of VIKI. They are an abstraction layer on top of ROS packages, allowing for easy use of them using default settings, easy linking through the gui and, perhaps most importantly, easy composition of different ROS packages. The *Tutorial on module XML writing* will introduce you into the basic concept. After that, you can get more in-depth on the other topics.

1.2.1 Tutorial on module XML writing

In this tutorial we will create a module. The module in this tutorial will start the ros_rqt_graph (which is already installed in ROS), which is a nice graph of ROS nodes and topics active on your PC.

To keep your module nicely separated from the rest it is advisable to create your modules in a new folder. Let us assume that your name is Doutzen Kroes and you name your folder *DoutzenMSC*. If you have installed the framework in the default location then running this should do the job for you:

```
cd ~/catkin_ws/src/viki_modules/ && mkdir DoutzenMSC
```

Inside there create your very first module definition file. You have no choice for name here, it must be named *viki.xml*. Open it to start defining your module. This should help you:

```
gedit viki.xml
```

Write, on the first line of the file

```
<!-- VIKI_MODULE -->
```

and save the file. Congratulations, you have just defined your first module! Nevertheless you did not do a good job, yet. Try running VIKI (you should know how by now). You should see that there is something wrong with the file you have just created because the terminal tells you something like:

```
Skipped adding '../aeroworks/contrib/DoutzenMSC/viki.xml' because it is a broken file. Error thrown w
Traceback (most recent call last):
  File "core/aero/scan.py", line 33, in getAvailableModules
    dom = xml.dom.minidom.parse(fPath)
  File "/usr/lib/python2.7/xml/dom/minidom.py", line 1918, in parse
    return expatbuilder.parse(file)
  File "/usr/lib/python2.7/xml/dom/expatbuilder.py", line 924, in parse
    result = builder.parseFile(fp)
  File "/usr/lib/python2.7/xml/dom/expatbuilder.py", line 211, in parseFile
    parser.Parse("", True)
ExpatError: no element found: line 2, column 0
```

As you can see that is quite a lot of text. The framework does not crash, and other modules are still available, but you can imagine that this is undesireable.

Now open your gedit with viki.xml again, and we will create something more useful.

Add this to your module file and save:

```
<module type="feedback_to_user" id="rqtmodule">

</module>
```

To help you and others increase the re-useability of the module please add metadata to all your module files, between the module tags, in this format:

```
<meta>
    <name>rqt module</name>
    <description>Created using a tutorial, this modules starts the ros_rqt_graph</description>
    <author>Doutzen Kroes</author>
</meta>
```

Your module is now no longer broken (confirm by running the core if you like) but it has not got any functionality either. You will need to add a reference to an *executable* which is a rosnode for us. Add this below your meta data, between the module tags, and save:

```
<executable id="graphnode" pkg="rqt_graph" exec="rqt_graph">
</executable>
```

For your convenience, this is how your viki.xml file should look now:

```
<!-- VIKI_MODULE -->
<module type="feedback_to_user" id="rqtmodule">
    <meta>
        <name>rqt module</name>
        <description>Created using a tutorial, this modules starts the ros_rqt_graph</description>
        <author>Doutzen Kroes</author>
    </meta>

    <executable id="graphnode" pkg="rqt_graph" exec="rqt_graph">
    </executable>
</module>
```

Your module is now finished and runnable. Of course, these are the real basics, look at *description of module attributes and tags* to go more in-depth regarding writing a module file.

1.2.2 More advanced modules

VIKI comes with (a currently not so) wide range of modules. As an engineer, you probably want to use your own functionality within VIKI. To do this, you will have to write a ROS package and enable it for VIKI by creating a *viki.xml* file. This file is a module description for your package which describes the inputs, outputs, executables and meta data. This guide assumes you already have a ROS package (or multiple of them) that you want to use within VIKI. For creating new ROS packages for use in VIKI, it is useful to follow some guidelines, which can be found at *Package and module writing guidelines*.

A module file is standard XML and generally looks like this:

```
<!-- VIKI_MODULE -->
<module type="sensor" id="camera_source">

    <meta>
        <name>USB camera</name>
        <author>Robin Hoogervorst</author>
        <icon>icon.png</icon>
        <description>
            Wrapper for the USB cam package within ros
        </description>
    </meta>

    <dependencies>
        <depends>libuvc_camera</depends>
    </dependencies>

    <!-- The in- and outputs of the module as a whole. They are linked to specific executables withi
    <outputs>
        <output type="ros_topic" name="image" link="usb_cam/image_raw" message_type="sensor_msgs/Imag
    </outputs>

    <executable id="usb_cam" pkg="libuvc_camera" exec="camera_node">
        <params>
            <param name="vendor" type="str" default="null" />
            <param name="product" type="str" default="null" />
            <param name="width" type="str" default="640" />
            <param name="height" type="str" default="480" />
            <param name="frame_rate" type="str" default="15.0" />
        </params>

        <outputs>
            <output type="ros_topic" name="image" message_type="sensor_msgs/Image" required="false" /
        </outputs>
    </executable>
</module>
```

General outline

Every module file starts out with:

```
<!-- VIKI_MODULE -->
```

This line makes sure the module is found by VIKI during the scan of files. Not implementing this line makes your module invisible to VIKI. The second line starts the module description:

```
<module type="sensor" id="camera_source">
```

This line specifies the type and id of the module. The type is a string on which VIKI groups its modules. You are free to choose anything you like here, but to be able to ingerate with VIKI better it is recommended to choose one of the following:

- Controller
- Middleware
- Rosdefaults
- Sensor
- Userinput
- Vehicle
- Views

The id is a string that is unique for every module. **Make sure it is not too generic!** If you make a module called 'camera', the chance is big that such a module already exists and causes conflicts with other modules. Preferably, use a prefix for your specific project modules.

After the opening lines, the important stuff happens. There are 5 types of XML nodes you can add here:

- meta
- inputs
- outputs
- executable
- configuration

Meta

As the name suggests, this is the place to add the meta information of your package. This looks like

```
<meta>
    <name>USB camera</name>
    <author>Robin Hoogervorst</author>
    <icon>icon.png</icon>
    <description>
        Wrapper for the USB cam package within ros
    </description>
</meta>
```

These tags basically speak for themselves, but for completeness sake:

- *name:* This is the name of the module as VIKI shows it in the list
- *author:* This is the name of the Author of the module. If you're writing it, it would be you.
- *icon:* The icon that VIKI uses to show it in the list. For more options, see below
- *description:* The description shown in the interface of VIKI

Icon

For specifying the icon, you have three options:

- Specify nothing: The default VIKI icon will be used.
- Specify a filename: VIKI will look for a image with this filename in the same folder as the viki.xml file. If you add icon.png here, make sure you add a icon.png file as well.
- Specify a bootstrap icon: If you provide an icon name which starts with 'glyphicon-' (e.g. glyphicon-star), VIKI will look for a bootstrap icon. This is an easy way to quickly add fancy icons. An overview of icons can be found at bootstrap icons

Dependencies

To be able to automatically let VIKI install the ROS packages, you need to specify which you are using. This is simply done by specifying the name of the ROS package inside the dependency tags.

```
<dependencies>
    <depends>libuvc_camera</depends>
</dependencies>
```

This can now be automatically installed using the VIKI command line tool. If there is no apt-get package available for your package, you can specify the repository of the package. Set a 'type' and 'src' attribute in the depends tag like such:

```
<dependencies>
            <depends type="git" src="https://github.com/ros-drivers/mocap_optitrack">mocap_optitrack<
    </dependencies>
```

Inputs and Outputs

```
<inputs>
    <input type="ros_topic" name="image_view_input" link="image_view/image" message_type="sensor_msgs
</inputs>

<outputs>
    <output type="ros_topic" name="image" link="usb_cam/image_raw" message_type="sensor_msgs/Image" r
    <output type="ros_topic" name="image2" link="usb_cam_2/image_raw" message_type="sensor_msgs/Image
    <output type="ros_topic" name="<name>" link="<executable_id>/<topic_name>" message_type="<ros_typ
</outputs>
```

The inputs and outputs come after the meta information. These specify the *module* in- and outputs, not executable specific ones. As can be seen, these blocks consist of a group XML node and (a set of) XML node(s) for each in/output. Attributes available for each specific in/output:

- *type*: This is the type of input for the module. Currently, only 'ros_topic' is supported.
- *name*: This is the name of the output, which will be shown in VIKI.
- *link*: This specifies to which ROS executable topic this links. It is of the format '<executable_id>/<topic_name>'.
- *message_type*: A ROS topic type: (e.g. sensor_msgs/Image, geometry_msgs/PoseStamped, std_msgs/Empty). VIKI makes sure you only can connect topics of the same type. So it's important to specify!
- *required*: Indicates whether the topic is required to be connected. Is currently not used in the interface, but will probably be implemented in the future.

Executables

An executable in a ROS node specifies a ROS node that is to be executed.

```
<executable id="usb_cam" pkg="libuvc_camera" exec="camera_node">
    <params>
        <param name="vendor" type="str" default="null" />
        <param name="product" type="str" default="null" />
        <param name="width" type="str" default="640" />
        <param name="height" type="str" default="480" />
        <param name="frame_rate" type="str" default="15.0" />
    </params>

    <outputs>
        <output type="ros_topic" name="image" message_type="sensor_msgs/Image" required="false" />
    </outputs>
</executable>
```

The first line has three attributes:

- *id*: This is the id used in the configuration to specify this executable. The module inputs and outputs are linked to executable inputs and outputs using this id.
- *pkg*: The package from which to run the node

- *exec*: The node that is to be run

The pkg and exec parameters correspond to running the node with

```
rosrun <package> <executable>
```

The params block corresponds to the parameters that can be set for each executable. The type corresponds to the types of ros parameters. Since this module file is basically an template for what will be run, only a default option can be set and no definite value. These default options can be changed by the user using VIKI before launch.

1.2.3 Module repositories

It is important to grasp the concept of module repositories. VIKI recursively scans its module folder, allowing us to put multiple repositories in that folder. By default, the core VIKI modules are located in *viki_modules/core*. If you would easily like to add your own specific modules, you can create a subfolder next to the core folder, such as *viki_modules/drone_modules*, in which you place your own modules.

1.2.4 description of module attributes and tags

VIKI_MODULE (comment, not a tag) comment at the beginning of viki.xml file telling VIKI that this is a file she can work with

MODULE Top level tag for the module attributes: - type: the kind of module this is, for instance userinput or sensor - id: unique identifier for this module (no modules can have the same name)

META Container for metadata

NAME Short name for the module

AUTHOR Module author/creator

DESCRIPTION Explains what the module does

ICON Icon used in VIKI's gui when displaying this module

INPUTS Container for module-wide inputs (when not inside **executable**-tag)

INPUT Description of module-wide input attributes: - type: (at this stage always ros_topic) - name: name for this input - link: connect to *thisexecutable/withthisinput/ - message_type: ros message type - required: tag to identify importance, for now unused

OUTPUTS Container for module-wide outputs (when not inside **executable**-tag)

OUTPUT Description of module-wide output attributes: - type: (at this stage always ros_topic) - name: name for this output - link: connect to *thisexecutable/withthisoutput/ - message_type: ros message type - required: tag to identify importance, for now unused

EXECUTABLE rosnode to be added attributes: - id: unique identifier inside this module - pkg: package this node can be found in - exec: rosnode to be run

INPUTS container for topics this executable can subscribe to (when inside **executable** tag)

INPUT topic this executable can subscribe to attributes: - type: (at this stage always ros_topic) - name: name for this input - message_type: ros message type - required: tag to identify importance, for now unused

OUTPUTS container for topics this executable can publish to (when inside **executable** tag)

OUTPUT topic this executable can publish to attributes: - type: (at this stage always ros_topic) - name: name for this output - message_type: ros message type - required: tag to identify importance, for now unused

PARAMS container for parameters

PARAM Parameter for this executable attributes: - name: name for this parameter (must be unique inside this module) - type: datatype ofthis parameter (unused) - default: default value of this parameter

1.2.5 Package and module writing guidelines

If you want to extend functionality in VIKI, you can make your own module. The module basically describes the ROS package, which you have written. When developing a ROS package for use in VIKI, it is good to follow some general guidelines to optimize the use and re-use of your package.

Modularity

VIKI aims at creating and maintaining a highly modular environment. This means we highly encourage you to write your modules as modular as possible. It is good to really think beforehand what you want the module to do and keep it as general as possible.

Location

If you want to add your own module, you can place this in your own folder in the viki_modules directory. If you have installed VIKI properly, you should have a viki_modules directory inside the src directory in your workspace. In there, there is the core directory that supplies the basic modules for VIKI. You can place a directory next to that, where your own modules and packages will live. You can place the viki.xml for VIKI in the same folder as you create your ROS package with.

```
+--- catkin_ws/src
|------ viki/
|------ viki_modules/
|       |--- core/
|       |--- <user>
|       |    |--- <your_module>
|       |    |    |--- src/
|       |    |    |--- package.xml
|       |    |    |--- viki.xml
|       |    |--- <your_second_module>
|       |         |--- src/
|       |         |--- package.xml
|       |         |--- viki.xml
```

Since VIKI does have good dependency management, you can also just put your ROS package in its own seperate repository and define the dependency in the module file.

Checklist

When writing a (ROS package for a) VIKI module you **must**:

- Use catkin to build your package
- Specify the right dependencies in the package.xml file (from ROS)
- Specify your name and contact information within the viki.xml file
- Specify your own package name in the dependency in the viki.xml file

Your module **must**:

- Publish its topics in its base namespace (so no '/' at the start of your topic name). (For more information about namespaces, see http://wiki.ros.org/Names) If you use python, this is done automatically, but if you're using C++, make sure you have the right nodehandle.

Your module **should**

- Have a clear (and preferably short) answer to the question: 'What is the task of this module?'

Your ROS package/node **should**:

- Live in the same folder as the viki.xml file
- Have clear documentation within the code

1.3 Developer documentation

This part of documentation provides information about the core of VIKI. This is useful if you either want to learn more about the internals of VIKI, or are interested in developing features and submitting a pull request on GitHub. Are you looking to contribute to VIKI? Great! Please follow the guidelines stated in *Contributing*

1.3.1 Contributing

So, you're probably here because you want to contribute to VIKI. That's great! We love the open-source community and are looking for people that want to make VIKI great, just as we do.

We follow the git branching model as described here http://nvie.com/posts/a-successful-git-branching-model/. If you already know how to work with it and want to contribute quickly. There is a quick contributing.md available in the GitHub repository.

Workflow

1.3.2 Releasing

Workflow for releasing a new version:

1. Checkout to a new release branch based on the commit that the release needs to be based on
2. Change the VERSION file by removing -dev from the version and possibly changing the name
3. Run the bumpversion.py script, which adds the right licensing to the files, based on the version provided in the VERSION file. **Make this a seperate commit**. If anything goes wrong, it is easy to revert this.
4. Commit and push release branch to github
5. Submit a pull request on GitHub and discuss
6. Create a new release using GitHub (marks the commit as release)
7. On the dev branch, make sure to change the version to <target version>-dev

Possibly, a lot of this can be automated. Since this is not that much work, this is not done at the moment, but would be not that difficult to do.

Hotfixing

When a hotfix needs to be made, make sure to checkout to the commit **before** the bumpversion has run. Based on that commit

1. Create a new hotfix branch
2. Make your changes and commit (only the neccessary changes)
3. Follow the release step 2-6 to publish this hotfix immediately

1.3.3 Framework description

There are a few parts to the framework:

Core

The core of the framework is the actual part doing the work. The core provides functionality and a GUI but is rather stupid: it knows of no ROS nodes, executables or anything else you actually want to run. To create a useful experience the core needs the following additions.

Documentation

Documentation consists of technical information and user guides. This page is also part of the documentation and you have probably already seen home.

Updating documentation

Once you have made changes please update the documentation by editing the *.rst* files in *VIKI/doc/* and re-build the documentation by running the following the the VIKI root directory

```
sphinx-apidoc -f -o docs core; cd docs; make html; cd ..
```

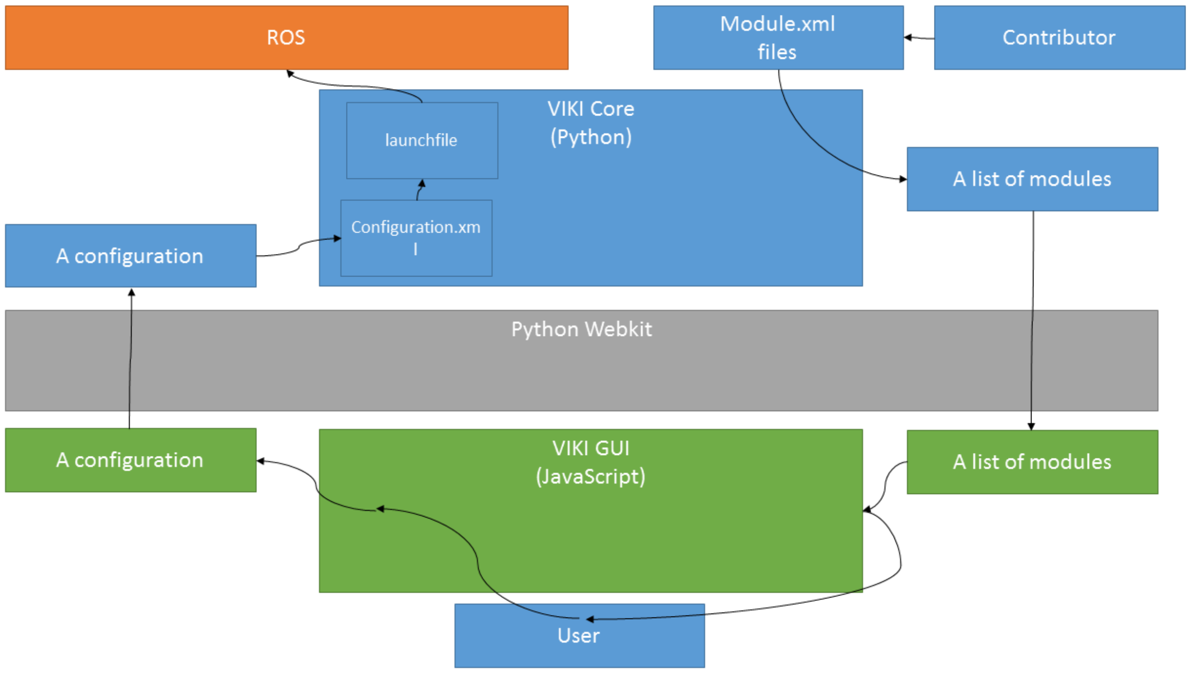
Modules

A module is a collection of ROS packages or nodes, with some configuration added so that it provides one simple function. A module is the smallest building block of the framework. You can write your own modules and put it in a dedicated folder so that you and others can use it. If you are a RAM member the developers provide a list of useful modules in an actively maintained repository at http://www.github.com/UT-RAM/viki-modules .

A more elaborate description (much more technical) can be found in *description of module attributes and tags*

1.3.4 System Overview

The picture below is an overview of the system.



Start at the right top and follow the arrows to match the description here:

- **contributors** People that have contributed modules to the framework they make viki.xml files
- **viki.xml** Several files that describe what nodes go together in what way, building blocks of an experimental setup
- **VIKI CORE*** The core, written in Python, interprets the viki.xml files and generates a list of modules.
- **list of modules** A list of modules that are now readily available in the system
- **Python Webkit** A webbrowser that opens the GUI frontend page. It has been wired via a dirty trick to make communication between javascript and Python possible: It simply changes the name of the page to send a message.
- **list of modules** Again a list of modules, now parsed in JavaScript
- **VIKI GUI** Graphical user interface, based on JavaScript. Uses the module list to build a pallette and uses the JsPlumb library for a drag-and-drop interface.
- **user** drags and drops in the GUI to create something he likes, tells the user to build a configuration
- **configuration** a set of modules connected together, with parameters, topic attached to each other etc. This is a full description of the experimental setup
- **Python webkit** used for communication to the Python backend again
- **Configuration** An abstraction of the complete experimental setup, in Python
- **configuration.xml** Human readable, temporary file that reflects the configuration. Basically just the DOM.
- **Launchfile** The core matches info in the configuration.xml with information from the viki.xml files to generate a ROS runnable launch file. Calls ROS to launch if required.
- **ROS** Robot Operating System

1.3.5 API documentation

For JavaScript (front end): Automatic API generation is not possible. The JavaScript is written self-explanatory as much as possible, commented where needed. In the future further documentation will be provided.

For Python (back end). This is automatically generated from the docstrings:

- genindex
- modindex
- search

1.4 The developers

1.4.1 Supervisor

Contact for questions concerning purpose, license and scientific work.

```
Matteo Fumagalli
m_fumagalli@m-tech.aau.dk
```

1.4.2 Core developers

Contact for any problems concerning the framework (try using one the the userguides first, they are listed on the home-page), technical issues and request for addition of modules.

```
Cees Trouwborst
ceestrouwborst@gmail.com
+316 4100 2540
```

```
Alex Kamphuis
a.kamphuis-1@student.utwente.nl
+316 1020 1776
```

```
Robin Hoogervorst
robin9975@gmail.com
```

CHAPTER 2

Automatically generated pages

- genindex
- modindex
- search