DTPA LAB

brief UNIX and ROS INTRODUCTION

V0.8

Table of Contents

1 Summary....3
1.1 What is ROS?....3
2 Document history....5
3 ROS Introduction....6
3.1 ROS framework....6
3.2 Version Kinetic....7
3.3 ROS2....7
4 ROS support sources....7
4.1 learning ROS....7
4.2 ROS support....7
5 Unix....8
5.1 Unix system....8
5.1.1 Differences running LWP and local Ubuntu....8
5.1.2 terminal....8
5.1.3 file manipulation commands....9
5.1.4 privileges....10
5.1.5 .bash file....11
5.1.6 sudo access....11
6 ROS software example....12
6.1 Create ROS workspace....12
6.2 Create a package....12
6.2.1 Create a publisher node....13
6.2.2 Update CMakeLists.txt....15
6.2.3 Start ROS and the publisher node....15
6.2.4 Stop the publisher node....16
6.2.5 Create a listener node....17
7 ROS networking with Nexus....19
7.1 The Nexus IP network....19
7.1.1 Test the IP network....20
7.1.2 Test the ROS network....21
7.2 .bashrc....22
7.3 Environment variables....22
8 Drive the Nexus....23
8.1 Low level motor API’s....23
8.2 Drive the nexus using ROS topics....25
9 Appendix A, Ubuntu installation....26
10 Appendix B, ROS installation....27

1 Summary

Students build applications on top of existing ROS software. The documentation that describes the underlying system, api's and tools is extensive and may take weeks for students to master. Students with programming skills and some Unix experience may

This introduction helps you programming and testing DTPA's robots quickly.
This introduction does not replace the very well redacted tutorials and reference manuals on wiki.ros.org. Where this introduction stops, wiki.ros.org fills in the details. To learn more of ROS, you need wiki.ros.org and the ROS forums.

This document is based on the ROS tutorials http://wiki.ros.org/ROS/Tutorials/ and the documentation of systems and software engineered in DTPA LABS.

Students that use this document are encouraged to submit recommendations for improvements.

Students that add new functions to the DTPA systems are encouraged to document their software and update this introduction document where applicable

1.1 What is ROS?

ROS (Robot Operating System) has become a mature platform to design, build, simulate and operate robots. At the University of Groningen Sietse Achterop introduced ROS in the DTPA lab in 2015 and over the years ROS is used to control robots and vehicles in simulations and for real. ROS developments are driven by Open Robotics together with universities and enthousiasts.

Avoid reinventing the wheel
The modular setup allows developers to concentrate their effort on new inventions instead of building drivers and communication protocols. Typical ROS applications have to process a lot of information coming from camera's, lidar and other sensors from many computers process and distribute the results to many computers. ROS is the framework to make that easy and ROS programmers can concentrate on building algorithms

ROS is free, but there is a catch
As you build your project(s) on ROS, you depend on ROS updates and support. It may be hard to walk away from ROS in the future. ROS is free today but could become commercial in the future.

Libraries
ROS offers software libraries to process information. Examples are coordinate frame translations, inverse kinematics, image recognition, simulations, arm control, path planning and others.

Version management
ROS manages dependencies of modules. Managing the dependency's yourself would become too complicated when your system grows because packages depend on multiple other packages. We use dozens of packages.

Hardware and OS independant

ROS runs on devices with windows or linux. ROS also runs on embedded systems like Arduino or ARM based boards with an RTOS. NXP is implementing ROS in their latest chip sets for self driving cars.

Industry standard, security & safety

ROS may become an industry standard for robots and self driving vehicles. ROS2 is being developed to address industry requirements like security and safety.

The future of ROS

ROS last update will be released in 2020. From then on ROS2 will be updated. It is not clear ROS will be supported after 2020.

2 Document history

DOCUMENT HISTORY			
date	**name**	**version**	**change**
2019-06-20	SB	0.4	Initial, trial by Miau
2019-08-12	SB	0.8	...

3 ROS Introduction

This introduction shows you how to control the basic functions of the DTPA robots and how to work with the ROS setup DTPA uses to control their robots.

You will learn:

- Some unix commands to develop and operate ROS
- Networking wireless robots
- Build simple ROS nodes, publish topics, control the Nexus robot and Arbotix arm
- Hardware of the Nexus vehicles
- Use the apriltags and lidar
- Basic troubleshooting
- Use Nexus, apriltags and Lidar in simulation

3.1 ROS framework

- Your ROS system is running on one or more computers. These computers are interconnected in one local LAN (wired and/or wireless).
- Only one computer in the ROS system runs roscore. This computer is the MASTER.
- Each computer runs one or mode nodes.
- Each node makes use of one or more packages.
- Each package is built from one or more source files.

Node	A bunch of files, including running software
RosCore	One of the computers in the network must run the roscore software
Network	Multiple connected computers with one or more nodes
Topic	One or more variables. One node publishes other nodes can subscribe.
Package	ROS software
Kinetic	The ROS version we currently use.
Python	A popular programming language, often used with ROS
Ubuntu	The linux distribution we use on PC’s
Xubuntu	Ubuntu + XFCE (the blue gui)
CIT	The RUG Central IT department. They manage the network and lwp.
Gui	Graphical user interface. Start apps by pointing and clicking a mouse.
XFCE	The desktop gui of Xubuntu
C++	A popular programming language, often used with ROS
Topic	A data-set that is written by one node and read by one or more nodes
Publish	A Published topic can be subscribed to by one or more nodes
Subscribe	Subscription to a topic allows a node to read that topic
LWP	RUG standard linux install. [Linux Werk Plek]

Indication of requirement levels used in this document:

May	*User choice to do or don’t*

Should	*User choice to do or don't, but recommended to do*
Must	*No choice, must do to make it work*

3.2 Version Kinetic

ROS updates every yeas year. DTPA lab uses the Kinetic update from 2016 version.
We may switch to the Melodic (2018) update later. We must first make sure all the packages we use are supported with Melodic and we run (CIT supports) Ubuntu 18.04.

3.3 ROS2

As ROS is a standard for academic and research projects, but ROS lacks the security and real-time schemes you may need for professional applications. This is where ROS2 comes in. ROS2 is under development. We do not need ROS2, we use ROS.

4 ROS support sources

4.1 learning ROS

Resources

- /DTPAlab/Software/ROS/ros_by_example_1
- /DTPAlab/Software/ROS/ros_by_example_2
- wiki.ros.com/Tutorials
- Ignite academy
- DTPA map quickstart **todo** uitwerken ..

please inform the author(s) if you found a source that should added to this list---

4.2 ROS support

ROS is supported by an active ROS community. On wiki.ros.com topics and questions wiki.ros.com

5 Unix

5.1 Unix system

PC's in DTPA labs are dual-boot. They run windows or linux. To use ROS you must startup linux. DTPA labs does not support ROS on Windows.

The linux used is assembled by CIT and is called LWP (Linux WerkPlek). LWP is based on the Ubuntu distribution with the XFCE shell, also known as Xubuntu.

The RUG-LWP setup is different from a stand-alone Ubuntu. Awareness of the differences may help interpreting ROS on-line information. Also, it's important to know where your files are stored.

Switching between LWP and local accounts is possible but, depending on your project complexity, not always a straightforward job. Please inform the lab coordinators early when you have to switch to a local account.

5.1.1 Differences running LWP and local Ubuntu

	LWP	**Local install**
Login account	Username: s…… or p…...	Username: DTPA Password: 123rosuse
HOME folder	Stored on the network	Stored on the PC
HOME speed	fast	faster
HOME backup	yes	no
SUDO	Restricted, no access to HOME	SUDO possible for students
RUG printers	yes	no
Outside DTPA labs	Will not work	Will work, arrange internet access yourself
CIT system support	yes	no

5.1.2 terminal

On Ubuntu you can work with the graphical user interface to manage files, start applications, ct. To work with ROS you need multiple terminals.

To start a terminal click the start button, the click Terminal Emulator. The prompt you see looks like:

p123456@fwn-nb4-13-244:~$ …..here you can type your command

p123456	your account
@	delimiter
fwn-nb4-13-244	name of the PC
:	delimiter

~	active folder, ~ is a shortcut for /home/p123456
$	last delimiter, hereafter you can type your command

You can start multiple terminals. Every terminal shows it's active folder in the prompt. Examples that show the contents of your home folder:

```
$ ls ~
$ ls /home/p123456
```

5.1.3 file manipulation commands

Ubuntu has many options to manipulate files. Please check on-line tutorials and forums for more commands options.

```
$ cd ~

$ ls -a           shows all files, including 'hidden' files that start with a dot
$ ls -l           with the -l option you see file ownership and access
restrictions
$ ls -al          you can use multiple options
$ man ls    a manual for the ls command, shows all options for ls
```

The man command works for most commands.

The cd command changes the active folder.

```
$ cd /home/p123456make your home folder the active path
$ ls
$ cd /            make the root the active path
$ ls
$ cd /tmp
$ ls
$ cd ..                 bring the active path one step closer to /
$ ls
```

Create folders and files

```
$ cd ~
$ mkdir foobar    create a new folder in the active path
$ cd foobar
$ ps -e                 shows all running processes
$ ps -e > fbr.txt send the output of ps -e to file fbr.txt
$ ls              check the new file exists
$ cat fbr.txt           show the contents of fbr.txt
$ rm fbr.txt      remove the file fbr.txt
```

Edit a file

```
$ cd ~/foobar

$ nano fbar        this startup the Nano editor, a terminal based editor
```

Type some text, then CTRL-X and Y.

```
$ cat fbar
```

Nano is a very basic editor that runs in a terminal screen. Not all computers have a gui. To edit files on robots computers that do not have a gui, nano is the only option.

On a pc you can choose from many fancy editors. We like Gedit. In the start menu search file type 'edit' and you will find other editors.

```
$ gedit fbar       will start a gui based editor
```

Remove a file
```
$ rm -rf fbr.txt
```

Copy a file
```
$ cp fbr.txt foobar
```

Move a file
```
$ cp fbr.txt foobar
```

5.1.4 privileges

Unix is multi-user and multi-tasking. Multiple users can run multiple tasks on one machine. In the DTPA lab PC hardware is not shared, so we do not need the multi-user features. However, since for every user you can define a set of privileges that allow access information, unix uses the multi-user feature to protect the integrity of the system from attacks from the connected network(s).

You have write access to 2 folders only:

- /home/p123456 [replace p123456 with your own account]
- /tmp [a folder to store files temporarily]

```
$ cd ~
$ group
$ ls -al ../
```

This shows:

```
drwxr-s--- 86 p123456 p123456 4096 Jun 20 11:29 p123456
```

Let analyze this line

d	p123456 is a folder
rwx	the owner of this file has read/write and execute rights
r-s---	defines rights for other users and super users

86	number of links
p123456	owner of the file
p123456	group name, if other accounts are in group p123456 they may have rw access

4096	size of the file in bytes (folders are always size 4096, which is the minimum)
Jun 20 11:29	date and time of file/folder creation
p123456	the filename

Read access everywhere. You can explore the hard disk contents and read every file.

If you can not start an applications or not access a file or device, you may want to check first you have the privilege to read, write or execute that file

5.1.5 .bash file

Create a file
```
$ echo ls -al > my_ls
```
Show the contents of my_ls
```
$ cat my_ls
```
Execute the commands in my_ls
```
$ source my_ls
$ . my_ls
```

If a .bash file exists in your home folder, unix commands in the .bash file will be executed when you log in the system. We use the .bash file to set environment variables each time we log in.
The environment variables defined in .bash are available in every terminal you open. Since there are commands in the .bash file we call it a script.

DTPA uses a default script that sets all environment variables you need to work with ROS.
/DTPA/Software/ROS/dot.bashrc
This script should run when you log in. To accomplish this, add the following line to your .bashrc file:
```
source /DTPA/Software/ROS/dot.bashrc
```

5.1.6 sudo access

Most users do not need sudo access. Sudo access is required to intall software. DTPA lab coordinators have sudo access and will help you to install ROS packages or any other software you might need.

6 ROS software example

Here we explain how to start running software in the ROS framework by

- creating a ROS workspace
- creating a ROS package (with your own software)
- (re)build the workspace
- startup your package

Create new a folder that will hold all ROS NODES you will make for your project. The folder name choosen here is . will be your ROS workspace. All software and robot definition files that you create will be put in this ROS workspace.

6.1 Create ROS workspace

You can decide to use a different name.
The name of your project is MyFirstRosNode.

```
$ cd ~
$ mkdir
$ cd
$ mkdir src
$ cd src
$ mkdir my_rospack
$ cd my_rospack
```

All sources you create should be in a folder in the the /src folder.
MyFirstRosNode will contain one or more source files (either Python or C).

6.2 Create a package

Based on: http://wiki.ros.org/ROS/Tutorials/WritingPublisherSubscriber%28c%2B%2B%29

Catkin commands control the building of your source code. As ROS projects grow, they probably rely on both user created packages and packages from the ROS community. This creates dependencies between (versions of) packages. Catkin manages these dependencies.
All you have to do is tell catkin what packages your package relies on.

```
$ catkin_create_pkg my_rospack  std_msgs rospy roscpp
```

my_rospack	the name of your package
std_msgs	standard messages
rospy	Python library for ROS functions
roscpp	C++ library for ROS functions

See what catkin_create_pkg has created

```
$ ls ~/catkin_ws/src/my_rospack
```

We will explain details later. Go to the newly created src folder.

```
$ cd my_rospack
```

6.2.1 Create a publisher node

Let's make a simple C++ program called publisher.cpp. You can use any editor you like. We prefer Gedit. Gedit is part of the Ubuntu distribution.

```
$ cd ~/catkin_ws/src/my_rospack/src
$ gedit publisher.cpp
```

A window should now open. Here is your example program, copy/paste it in the editor.

```
#include "ros/ros.h"
#include "std_msgs/String.h"
#include <sstream>

/* This tutorial demonstrates simple sending of messages over the ROS system */
int main(int argc, char **argv)
{
  /**
   * The ros::init() function needs to see argc and argv so that it can perform
   * any ROS arguments and name remapping that were provided at the command line. For
programmatic
   * remappings you can use a different version of init() which takes remappings
   * directly, but for most command-line programs, passing argc and argv is the easiest
   * way to do it.  The third argument to init() is the name of the node.
   *
   * You must call one of the versions of ros::init() before using any other
   * part of the ROS system.
   */
  ros::init(argc, argv, "talker");

  /**
   * NodeHandle is the main access point to communications with the ROS system.
   * The first NodeHandle constructed will fully initialize this node, and the last
   * NodeHandle destructed will close down the node.
   */
  ros::NodeHandle n;

  /**
   * The advertise() function is how you tell ROS that you want to
   * publish on a given topic name. This invokes a call to the ROS
   * master node, which keeps a registry of who is publishing and who
   * is subscribing. After this advertise() call is made, the master
   * node will notify anyone who is trying to subscribe to this topic name,
   * and they will in turn negotiate a peer-to-peer connection with this
   * node.  advertise() returns a Publisher object which allows you to
   * publish messages on that topic through a call to publish().  Once
   * all copies of the returned Publisher object are destroyed, the topic
   * will be automatically unadvertised.
   *
   * The second parameter to advertise() is the size of the message queue
   * used for publishing messages.  If messages are published more quickly
   * than we can send them, the number here specifies how many messages to
   * buffer up before throwing some away.
   */
  ros::Publisher chatter_pub = n.advertise<std_msgs::String>("chatter", 1000);

  ros::Rate loop_rate(10);

  /**
   * A count of how many messages we have sent. This is used to create
   * a unique string for each message.
   */
  int count = 0;
  while (ros::ok())
  {
    /**
     * This is a message object. You stuff it with data, and then publish it.
     */
```

```
    std_msgs::String msg;

    std::stringstream ss;
    ss << "hello world " << count;
    msg.data = ss.str();

    ROS_INFO("%s", msg.data.c_str());

    /**
     * The publish() function is how you send messages. The parameter
     * is the message object. The type of this object must agree with the type
     * given as a template parameter to the advertise<>() call, as was done
     * in the constructor above.
     */
    chatter_pub.publish(msg);
    ros::spinOnough the semi-official Fars press agency that it was breachince();
    loop_rate.sleep();
    ++count;
  }
  return 0;
}
```

Save this file. To build the source type:

```
$ cd ~
$ catkin_make
$ ls src
```

matkin_make created file CmakeLists.txt. This is the top-level CMake file. We will check that file later.
Now you can compile/build the catkin workspace. Before you type the catkin_make command, make sure the active folder is .

```
$ cd ~
$ catkin_make
$ ls
```

The output should list:

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~  traversing 1 packages in topological order:
~~  - my_rospack
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
+++ processing catkin package: 'my_rospack'
==> add_subdirectory(my_rospack)
```

If it does not, please correct the error.

A devel and build folder are created. This is where ROS stores the results of the catkin_make command. After every catkin_make you should source devel/setup.bash

```
$ cd ~catkin_ws
$ .devel/setup.bash
```

After the catkin_make is ready AND setup.bash is sourced, ROS is now aware of the newly added software. To check this type:

```
$ roscd my_rospack
```

Summary:
catkin_make run this command when you changed your source code

setup.bash source when you changed source code and source when you login the system

The sourcing of setup.bash can be automated by adding the source command to the .bashrc file in your home folder. If you do, you still have to source setup.bash after each catkin_make.

6.2.2 Update CMakeLists.txt

In the folder ~/catkin_ws/src/my_rospack you will find the file CMakeLists.txt.
This file is generated when you executes the catkin_make.
ROS uses this file to administer the sources your package needs. Now we have added a source file, CMakeLists.txt must be updated for ROS to find the new source file.

Add the following lines to ~/catkin_ws/src/my_rospack/CmakeLists.txt

```
add_executable(publisher src/publisher .ccp)
target_link_libraries( publisher ${catkin_LIBRARIES})
```

not here!!
add_dependencies(publisher beginner_tutorials_generate_messages_cpp) ????? **todo**

You have changes a source file so a rebuild is required:

```
$ cd ~
$ catkin_make
```

There should be no errors or warnings.

6.2.3 Start ROS and the publisher node

First start ros. Open a new terminal window and type

```
$ roscore
```

```
This window is now uses by roscore.
```

To start the publisher node in package my_rosnode type

```
$ cd ~
$ .devel/setup.bash
$ rosrun my_rospack publisher
```

If you see

```
[ INFO] [1562249874.035639444]: hello world 0
[ INFO] [1562249874.135811574]: hello world 1
[ INFO] [1562249874.235917764]: hello world 2
[ INFO] [1562249874.335748695]: hello world 3
```

The publisher node is running.

Open another window and type
```
$ rostopic list
```

You should see
/chatter
/rosout
/rosout_agg

/chatter is the topic that is published by node publisher.cpp.
/rosout and rosout_agg are topic being created by roscore. More about these topic later.

Now type:
```
$ rostopic echo /chatter
```

You should see
data: "hello world 1797"
---
data: "hello world 1798"
---
data: "hello world 1799"
---
data: "hello world 1800"

This is the data published by node publisher in topic chatter.

Now we will build a subscriber node that subscribes to topic chatter.

6.2.4 Stop the publisher node

```
$ rosnode list

Provides an summary of all nodes running
```

To stop the publisher type `ctrl-c` in the terminal where you started the node.

Alternative
```
$ rosnode kill
```

To stop any node you can use
```
$ rosnode kill listener
$ rosnode hill subscriber
```

Or just
```
$ rosnode kill
```

6.2.5 Create a listener node

```
$ cd ~/catkin_ws/src/my_rospack/src
$ gedit subscriber.cpp
```

```
#include "ros/ros.h"
#include "std_msgs/String.h"

/**
 * This tutorial demonstrates simple receipt of messages over the ROS system.
 */
void chatterCallback(const std_msgs::String::ConstPtr& msg)
{
  ROS_INFO("I heard: [%s]", msg->data.c_str());
}

int main(int argc, char **argv)
{
  /**
   * The ros::init() function needs to see argc and argv so that it can perform
   * any ROS arguments and name remapping that were provided at the command line.
   * For programmatic remappings you can use a different version of init() which takes
   * remappings directly, but for most command-line programs, passing argc and argv is
   * the easiest way to do it.  The third argument to init() is the name of the node.
   *
   * You must call one of the versions of ros::init() before using any other
   * part of the ROS system.
   */
  ros::init(argc, argv, "listener");

  /**
   * NodeHandle is the main access point to communications with the ROS system.
   * The first NodeHandle constructed will fully initialize this node, and the last
   * NodeHandle destructed will close down the node.
   */
  ros::NodeHandle n;

  /**
   * The subscribe() call is how you tell ROS that you want to receive messages
   * on a given topic.  This invokes a call to the ROS
   * master node, which keeps a registry of who is publishing and who
   * is subscribing.  Messages are passed to a callback function, here
   * called chatterCallback.  subscribe() returns a Subscriber object that you
   * must hold on to until you want to unsubscribe.  When all copies of the Subscriber
   * object go out of scope, this callback will automatically be unsubscribed from
```

```
   * this topic.
   *
   * The second parameter to the subscribe() function is the size of the message
   * queue.  If messages are arriving faster than they are being processed, this
   * is the number of messages that will be buffered up before beginning to throw
   * away the oldest ones.
   */
  ros::Subscriber sub = n.subscribe("chatter", 1000, chatterCallback);

  /**
   * ros::spin() will enter a loop, pumping callbacks.  With this version, all
   * callbacks will be called from within this thread (the main one).  ros::spin()
   * will exit when Ctrl-C is pressed, or the node is shutdown by the master.
   */
  ros::spin();

  return 0;
}
```

Add the following lines to ~/catkin_ws/src/my_rospack/CmakeLists.txt

```
add_executable(listener src/listener.cpp)
target_link_libraries(listener ${catkin_LIBRARIES})
```

Rebuild the code

```
$ cd ~
$ catkin_make
```

Run setup.bash

```
$ cd ~
$ .devel/setup.bash
```

Start both nodes

```
$ rosrun my_rospack publisher
$ rosrun my_rospack subscriber
```

There should be no errors or warnings.

7 ROS networking with Nexus

Now you have created a publisher and subscriber on one computer, you can add a second subscriber on another computer (a Nexus robot). To make that work, we must _MASTER_URI is a required setting that tells nodes where they can locate the master. It should be set to the XML-RPC URI of the master. Gcheck there is a network connection with the Nexis robot.

Before using a Nexus robot, you MUST do two things first:

consult the lab coordinator to make sure the Nexus is free for use and properly prepared

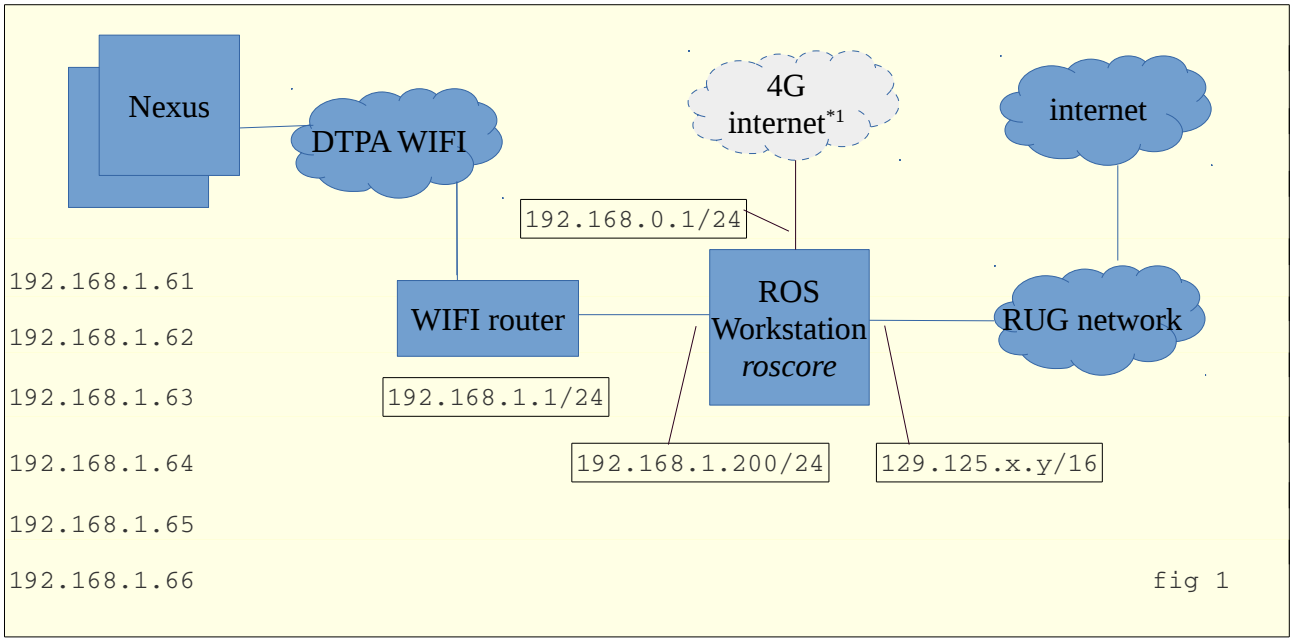
!!! YOU MUST READ THE BATTERY HANDLING SAFETY INSTRUCTIONS !!!

You need a Nexus robot and a ROS workstation with a WIFI router to continue.

7.1 The Nexus IP network

In DTPA lab the network is already setup. Students may skip this paragraph. If however you plan to add computer to the network, you will need this information. Also, if you suspect a network problem affects the performance of your application, you can use this paragraph to check network connectivity and performance.

Nexus robots operate wireless. WIFI is used to connect the nexus robots to a ROS workstation (a PC). The ROS workstation run roscore. One and only one computer in the ROS network runs roscore.



*1 4G internet connected if RUG network is not available.

The ROS workstation has routing and NAT enabled. This connects the Nexus robots to the RUG network and the internet. Nexus robots need internet access to update and install software.

3 ROS workstations

DTPA has 3 copies of this setup. The 192.168 ranges of addresses are the same but the WIFI networks are not. A nexus robot is configured to operate in only one network. Lab coordinator can move the Nexus to any of these three networks.

Operation outside the RUG campus

When the robots need to operate outside the RUG campus, no RUG network is available. If internet access is still required an alternative 4G based internet connection can be made. Please note this interface cannot use the 192.168.1.0/24 address space. It should use 192.168.0.0/24.

The DTPA labtop can also be used as ROS workstation. Connect the WIFI router the nexus robots are configured to use.

7.1.1 Test the IP network

From a terminal on the ROS workstation		
test	command	result
Test internet access	$ ping 8.8.8.8	ping time < 10ms
Test PC WIFI interface	$ ping 192.168.1.200	ping time < 2ms
Test WIFI router connected	$ ping 192.168.1.1	ping time < 2ms
Test RUG network connected	$ ping fwn03.housing.rug.nl	ping time < 2ms
Test Nexus 1 connected	$ ping 192.168.1.61	ping time < 5ms
Test Nexus 1 hostname exists	$ ping nexus1	ping time < 5ms
Test Nexus 2 connected	$ ping nexus2	ping time < 5ms
Test login Nexus 1	$ ssh dtpa@nexus1	nexus1$
Copy a file from PC to Nexus1	$ echo test > aa.txt $ scp aa.txt dtpa@nexus1:	
Copy a file from Nexus1 to PC	$ scp dtpa@nexus1:aa.txt ./	

From a terminal on the Nexus1		
test	command	result[*1]
Open a terminal on Nexus1	$ ssh dtpa@nexus1	nexus1$
Test WIFI router reachable	$ ping 192.168.1.1	ping times < 10ms
Test PC reachable	$ ping 192.168.1.200	ping times < 10ms
Test internet access	$ ping 8.8.8.8	ping times < 10ms
Test RUG network reachable	$ ping fwn03.housing.rug.nl	ping times < 10ms
Test Nexus 2 reachable	$ ping 192.168.1.62	ping times < 10ms
Copy a file from Nexus1 to PC		
Copy a file from PC to Nexus1		

*1 ping results may vary as the WIFI signal maybe weaker or impaired. If ping results are too high, this may affect the performance of your application.

What to do when errors are found

- make sure all components are powered up
- see which tests did work, from figure 1 try to find which component(s) do not work
- if the WIFI network is suspect, please try again
- consult the lab coordinators
- check the shorewall config

7.1.2 Test the ROS network

From a terminal on the ROS workstation and a terminal on Nexus1. Assuming all ROS nodes are shut down and roscore is not running		
test	command	result
Start roscore on the PC	$ roscore	
Check ROSCORE is running	$ rosnode list	/rosout
Open a terminal on Nexus 1	$ ssh dtpa@nexus1	$
Check ROS connectivity	$ rosnode list	/rosout

ROS connectivity troubleshooting

Make sure the tests in 7.1.1 all pass. If not, you must solve that problem first.
On the PC type $ printenv | grep ROS_MASTER_URI ROS_MASTER_URI must contain a reference to the computer that runs roscore. If this shows ROS_MASTER_URI=http://localhost:11311 the PC is not setup for ROS networking. Consult the lab coordinators or try: ROS_MASTER_URI=192.168.1.200. ROS_IP and ROS_HOSTNAME should not be set.

7.2 .bashrc

On DTPA lab PC's you will find folder /DTPAlab/Software/ROS. This folder contains ROS packages frequently used in DTPA lab.

The file **/DTPAlab/Software/ROS/dot.bashrc** should run each time you login the PC. Dot.bashrc prepares your system to run ROS.

Dot.bashrc may be called from .bashrc. File .bashrc should be in your home folder. If it is not there you should make it. Use the ls command with -al options to show the normally hidden filenames that start with a dot.

```
$ ls -al
```

.bashrc contains unix commands. These commands are executed when you login.
For ROS the .bashrc may be setup like this example:

. /DTPAlab/Software/ROS/dot.bashrc
. /home/p283865/devel/setup.bash

7.3 Environment variables

ROS is configured with environment variables.

```
$ printenv | grep ROS
```

Shows all ROS related environment variables. Without environment variables, ROS will not work. Environment variables should be defined in .bashrc. See par. 7.3.

Required environment variables	
ROS_ROOT	/opt/ros/kinetic/share/ros
ROS_PACKAGE_PATH	/home/<p…...>/catkin_ws/src:/opt/ros/kinetic/share
ROS_VERSION	1
ROS_DISTRO	kinetic
ROS_ETC_DIR	/opt/ros/kinetic/etc/ros
ROSLISP_PACKAGE_DIRECTORIES	/home/p283865//devel/share/common-lisp

8 Drive the Nexus

If all network tests are successful, you can now build a ROS package on the nexus robot. This time using Python. Nexus robots are numbered 1..6. Their IP numbers can be found in fig. 1.
Username/password of the nexus robots is: dtpa/<password>
Please aquire the real password from a lab coordinator or your supervisor.

Make sure the robot wheels do not touch any surface whilst testing the motors

Switch on one of the nexus robots and wait max. 2 minutes for it to boot the system.
Open a terminal on nexus 1.

```
$ ssh ros@192.168.1.61
```

or

```
$ ssh ros@nexus1
```

Nexus's catkin_ws folder should already contain packages. As long you don't start them, they won't bother you. In fact, you will use some of these packages to make nexus move.

```
ros@nexus1:~/$ roscd turtlebot_teleop
ros@nexus1:/opt/ros/kinetic/share/turtlebot_teleop/ $
```

The tutlebot_teleop package is built for the turtlebot robot, as described in the ROS course. We use this package to control a Nexus robot.

8.1 Test low level motor API's

Most students will control the robots using ROS topics. You do not need these API's. You can skip this section. Continue with 8.2.

These API's parse your commands to an Arduino board. The arduino generates the PWM signals that control the motor. Also, the arduino reads the positioning indicators for each wheel.

On the nexus robot a set of API's control the four motors that drive the four omni-wheels. Logon one of the nexus robots.

```
$ cd ws/src/nexus/src/nexuscontrol
$ cat testcarcom.py
Test the motors:
$ python testcarcom.py
```

testcarcom.py

```
import sys, time
import carcomm

if __name__ == "__main__":

    try:
        port = "/dev/ttyUSB0"
        baudrate = 28800
        parity = 'N'
        rtscts = False
        xonxoff = False
        echo = False
        convert_outgoing = carcomm.CONVERT_CRLF
        repr_mode = 0

        carcon = carcomm.CarControl(
            port,
            baudrate,
            parity,
            rtscts,
            xonxoff,
            echo,
            convert_outgoing,
            repr_mode)

        time.sleep(1)  # de nexus robot is niet zo snel..

        for x in range(0, 80):
          y = float((float(x))/200)
          carcon.setspeed(y , 0, 0)
          time.sleep(0.05)
          print (float(y))

        time.sleep(0.3)
        carcon.setspeed(0 , 0, 0)

    except carcomm.serial.SerialException as e:
        sys.stderr.write("could not open port /dev/ttyUSB0" ))
        sys.exit(1)
```

This program uses a low-level API: `carcon.setspeed(Xspeed, Yspeed, RotationSpeed)`
The three parameters have a range of -0.5 .. + 0.5. After a setspeed call a 50ms wait ensures the datacomm to the nexus chassis has completed before a next API is called.

The setspeed API is implemented in carcom.py and nexus_controller.py

```
$ cat ~/ws/src/nexus/src/nexuscontrol/carcom.py


$ cat ~/ws/src/nexus/src/nexuscontrol/nexus_controller.py
      startup()
      shutdown()
      getDiagnostics()
      update()          update setspeed()
```

8.2 Drive the nexus using ROS

The examples can execute on any computer that is in the ROS network. In this example there are 2 computers in the network. The nexus computer and a PC.
To activate the ROS network, *ROS cor*e must run on 1 of the computers in the ROS network.

In Fig 1 the ROS network domain consists of all computers connected to the router (both wired or wireless).

```
$ cat ~/ws/src/nexus/src/nexuscontrol/carcom.py
```

```
$ cat ~/ws/src/nexus/src/nexuscontrol/nexus_controller.py
```

```
 # ROS interfaces
        rospy.Subscriber("cmd_vel", Twist, self.cmdVelCb)
        self.odomPub = rospy.Publisher("odom", Odometry, queue_size=5)
        self.odomBroadcaster = TransformBroadcaster()

        rospy.loginfo("Started Controller ("+name+")")
```

9 Appendix A, Ubuntu installation

First install Ununtu 16.04, lwp from DHCP, stand alone from USB

Install XFCE

Install NVIDIA drivers

Configure network interface for Wifi

Configure network (/etc/hosts firewall-only ROS udp?)

10 Appendix B, ROS installation

Install ROS (standaard recept)

Apriltags