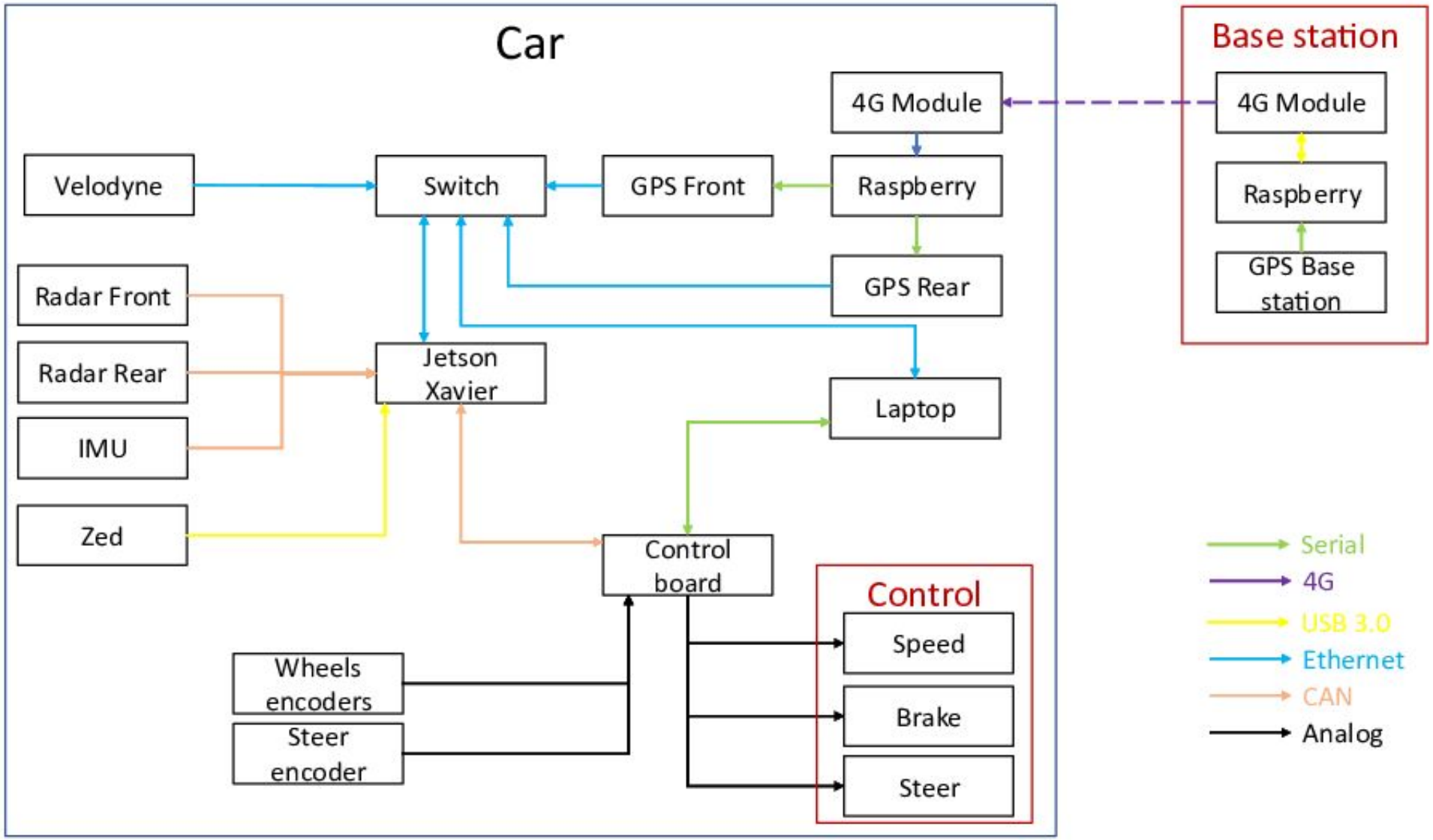
ROS ON MULTIPLE MACHINES

ROBOTICS

POLITECNICO
MILANO 1863

Example setup



ROS DISTRIBUTED

Ros can work as a distributed system on multiple devices connected to the same network

Big project always work as a distributed system

Remote monitoring of robots can easily be done with only one ROS network

To use ROS on multiple device you need to run the ROS master, command “roscore”, only on one of the device

For all the other nodes you need to specify the ip of the master

COMMON CONFIGURATION

Get your ip: “ifconfig command”-> “inet addr”

Export all the variables to properly configure the master, to set all those variables for every new terminal add them at the end of your ~/.bashrc

$ gedit ~/.bashrc

MASTER CONFIGURATION

First we set the master IP:

```
export ROS_MASTER_URI=http://master_ip:11311
```

First we set the master URI

your IP

Standard ROS port, you can also run on different ports

MASTER CONFIGURATION

Tell ROS master my IP:

```
export ROS_IP=master_ip
```

← your IP

Tell ros my ip

CLIENTS CONFIGURATION

First we set the master IP:

```
export ROS_MASTER_URI=http://master_ip:11311
```

First we set the master URI

master ip

Standard ROS port, you can also run on different ports

CLIENTS CONFIGURATION

Tell ROS master my IP:

```
export ROS_IP=master_ip
```

← your IP

↑ Tell ros my ip

ROS DISTRIBUTED

On the **master** pc use the command “roscore”

To test if everything is working on the **clients** open a new terminal and call “rostopic list” without previously running “roscore”. You should be able to see topics on the ROS network.

Now all client are on the same network and can communicate and start node on the distributed ROS netowrk

TIME SYNCHRONIZATION

Recording high-throughput bags often requires to split the recordings on different ROS devices, to use the bags all together they need to have coherent timestamp.

We then need to synchronize the clock of all the devices on the ROS network.

The standard procedure to synchronize multiple devices on a local network is tu use an ntp server on a master device and a chrony client for all the other devices.

In a ROS network the procedure is to install the NTP server on the master and chrony on all the other nodes.

MASTER CONFIGURATION (/etc/ntp.conf)

```
driftfile /var/lib/ntp/ntp.drift

statistics loopstats peerstats clockstats

filegen loopstats file loopstats type day enable

filegen peerstats file peerstats type day enable

filegen clockstats file clockstats type day enable

pool 0.ubuntu.pool.ntp.org iburst

pool 1.ubuntu.pool.ntp.org iburst

pool 2.ubuntu.pool.ntp.org iburst

pool 3.ubuntu.pool.ntp.org iburst

server 127.127.1.0

fudge 127.127.1.0 stratum 10

pool ntp.ubuntu.com

restrict -4 default kod notrap nomodify nopeer noquery limited

restrict -6 default kod notrap nomodify nopeer noquery limited

restrict 192.0.0.0 mask 255.0.0.0 nomodify notrap

restrict 127.0.0.1

restrict ::1
restrict source notrap nomodify noquery
```

CLIENT CONFIGURATION (/etc/chrony/chrony.conf)

```
server 192.168.0.100 minpoll 2 maxpoll 4
initstepslew 2 192.168.0.100
keyfile /etc/chrony/chrony.keys

commandkey 1
driftfile /var/lib/chrony/chrony.drift
maxupdateskew 5
dumponexit
dumpdir /var/lib/chrony
pidfile /var/run/chronyd.pid
logchange 0.5
rtcfile /etc/chrony.rtc
rtconutc
rtcdevice /dev/rtc
sched_priority 1
local stratum 10
allow 127.0.0.1/8
```

← Master IP

TIME SYNCHRONIZATION

The chrony configuration file can be found in \etc\chrony\chrony.conf

Then stop and restart chrony to make those changes effective:

$ sudo service chony stop

$ sudo service chony start

Then to monitor how synchronization is doing:

$ chronyc tracking

LATCHED MESSAGES

ROBOTICS

POLITECNICO
MILANO 1863

latched messages

```
ros::Publisher chatter_pub = n.advertise<std_msgs::String>("chatter",  1, true);
```

for low frequency publisher (e.g., maps)

subscriber receive last published messages when subscribe

subscriber does not need to wait for new published messages

publish latched messages

Asynchronous spinner

ROBOTICS

POLITECNICO
MILANO 1863

WHY DO WE NEED ASYNCHRONOUS SPINNER?

- Standard ROS: 1 node -> 1 thread
- Real thread implementation, without the need of actually writing threads
- Multiple subscriber which has long computation
- Scenarios where action is not suitable, everything should be in the same node

Standard node with two subscribers

```
#include <ros/ros.h>
#include <std_msgs/String.h>
void callbackTalker1(const std_msgs::String::ConstPtr &msg)
{
   ROS_INFO_STREAM("Message from callback 1:" );
   ros::Duration(2.0).sleep();
   ROS_INFO("%s", msg->data.c_str());
}

void callbackTalker2(const std_msgs::String::ConstPtr &msg)
{
   ROS_INFO_STREAM("Message from callback 2:");
   ROS_INFO("%s", msg->data.c_str());
}

int main(int argc, char **argv)
{
   ros::init(argc, argv, "talker_subscribers");
   ros::NodeHandle nh;
   ros::Subscriber counter1_sub = nh.subscribe("talker1", 1, callbackTalker1);
   ros::Subscriber counter2_sub = nh.subscribe("talker2", 1, callbackTalker2);
   ros::spin();
}
```

Time consuming callback

Sleep simulate long computation time

Fast callback

Two subscribers

Node with asynch spinner

```
#include <ros/ros.h>
#include <std_msgs/String.h>
void callbackTalker1(const std_msgs::String::ConstPtr &msg)
{
   ROS_INFO_STREAM("Message from callback 1:" );
   ros::Duration(2.0).sleep();
   ROS_INFO("%s", msg->data.c_str());
}

void callbackTalker2(const std_msgs::String::ConstPtr &msg)
{
   ROS_INFO_STREAM("Message from callback 2:");
   ROS_INFO("%s", msg->data.c_str());
}

int main(int argc, char **argv)
{
   ros::init(argc, argv, "talker_subscribers");
   ros::NodeHandle nh;
    ros::AsyncSpinner spinner(0);
   spinner.start();

   ros::Subscriber counter1_sub = nh.subscribe("talker1", 1, callbackTalker1);
   ros::Subscriber counter2_sub = nh.subscribe("talker2", 1, callbackTalker2);
   ros::waitForShutdown();
}
```

No changes in the callback

Create the spinner

Start the spinner, before subscriber, no need for ros::spin