ROS

Robot Operating System

ROS

- Software framework/middleware for robot applications (initial release 2007, first version Box Turtle 2010)

- Programming languages:
  - Mainly C++, Python, LISP
  - Experimental Java, C#, Ruby, R, Lua, Go etc.

- Package management (over 3000 packages available)

- Powerful build system on top of CMake

- Message passing:
  - easy de-/serialization
  - Publisher/Subscriber or Service invocation concept

- Big community, developed and documented by thousands of contributors

ROS

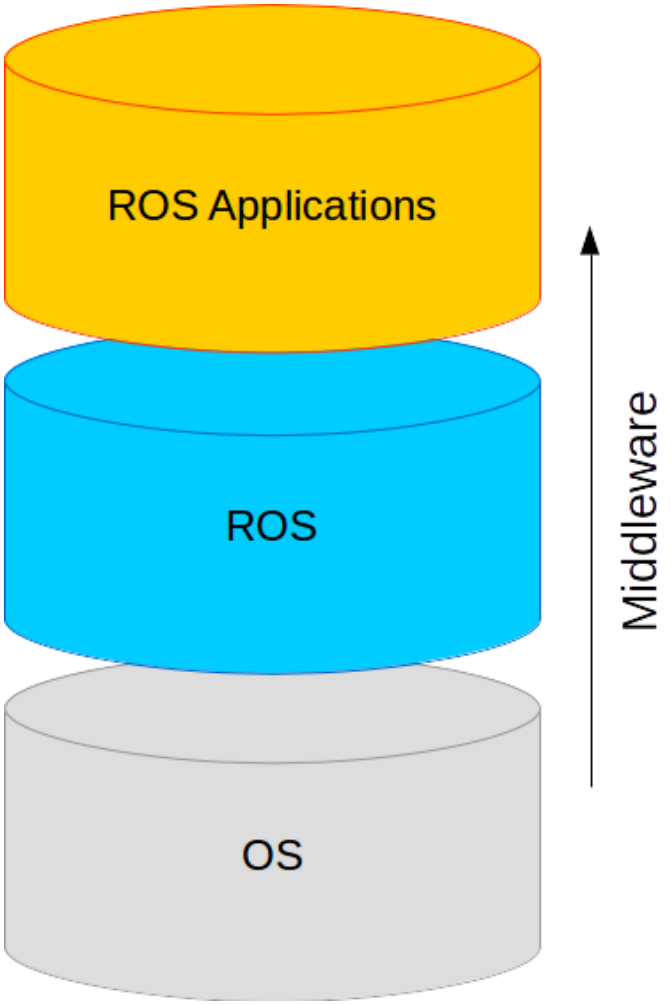
- Many existing libraries and tools, for example:
  - motion planning
  - object recognition
  - hardware interfaces
  - plotting
  - 3D visualization
  - data serialization

- https://youtu.be/3ydRXC76MV0?t=1m30s

ROS - Usage

- Academic Research: http://robots.ros.org/
- ROS-INDUSTRIAL: http://rosindustrial.org/
- Autonomous Cars: http://www.ros.org/news/robots/autonomous-cars/
- NASA: http://www.ros.org/news/2014/09/ros-running-on-iss.html

ROS - Software Stack



ROS

- We use ROS because it allows for easier hardware abstraction and code reuse

- In ROS, all major functionality is broken up into a number of chunks that communicate with each other using messages

- Each chunk is called a node and is typically run as a separate process

- Matchmaking between nodes is done by the ROS Master Node

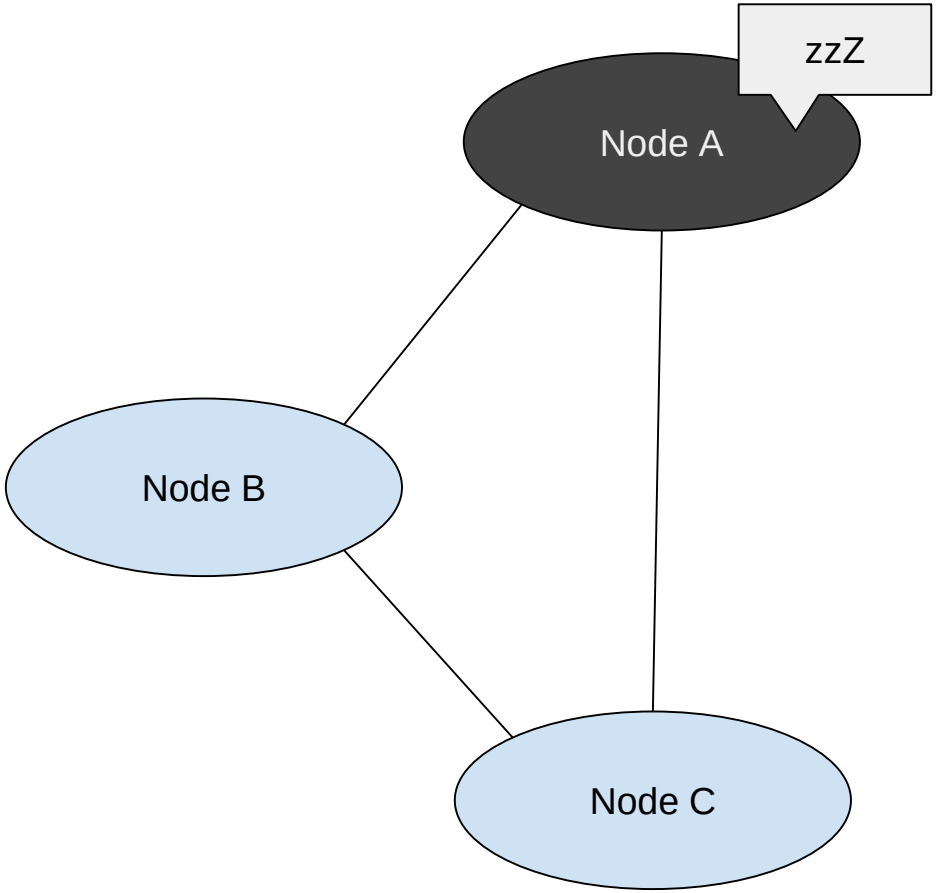
ROS - System Architecture

- Modular node concept

- Node runs a specific computation and share data with the network

- Nodes can be added, removed while ROS is running

- Can run on different machines too (distributed system)

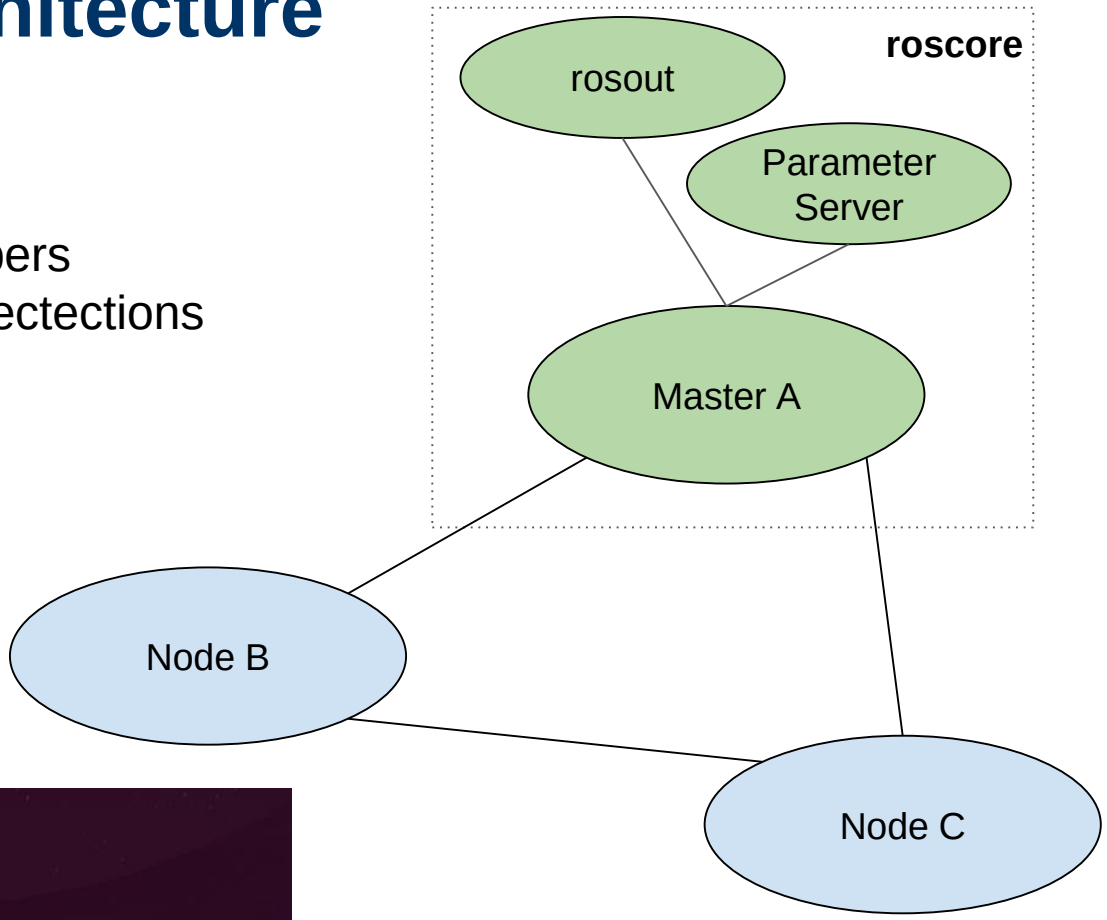


ROS - System Architecture

roscore:
- the **master** node:
    - tracks publishers / subscribers
    - enables peer-to-peer connectections between nodes

- **parameter** server

- logging node **rosout**



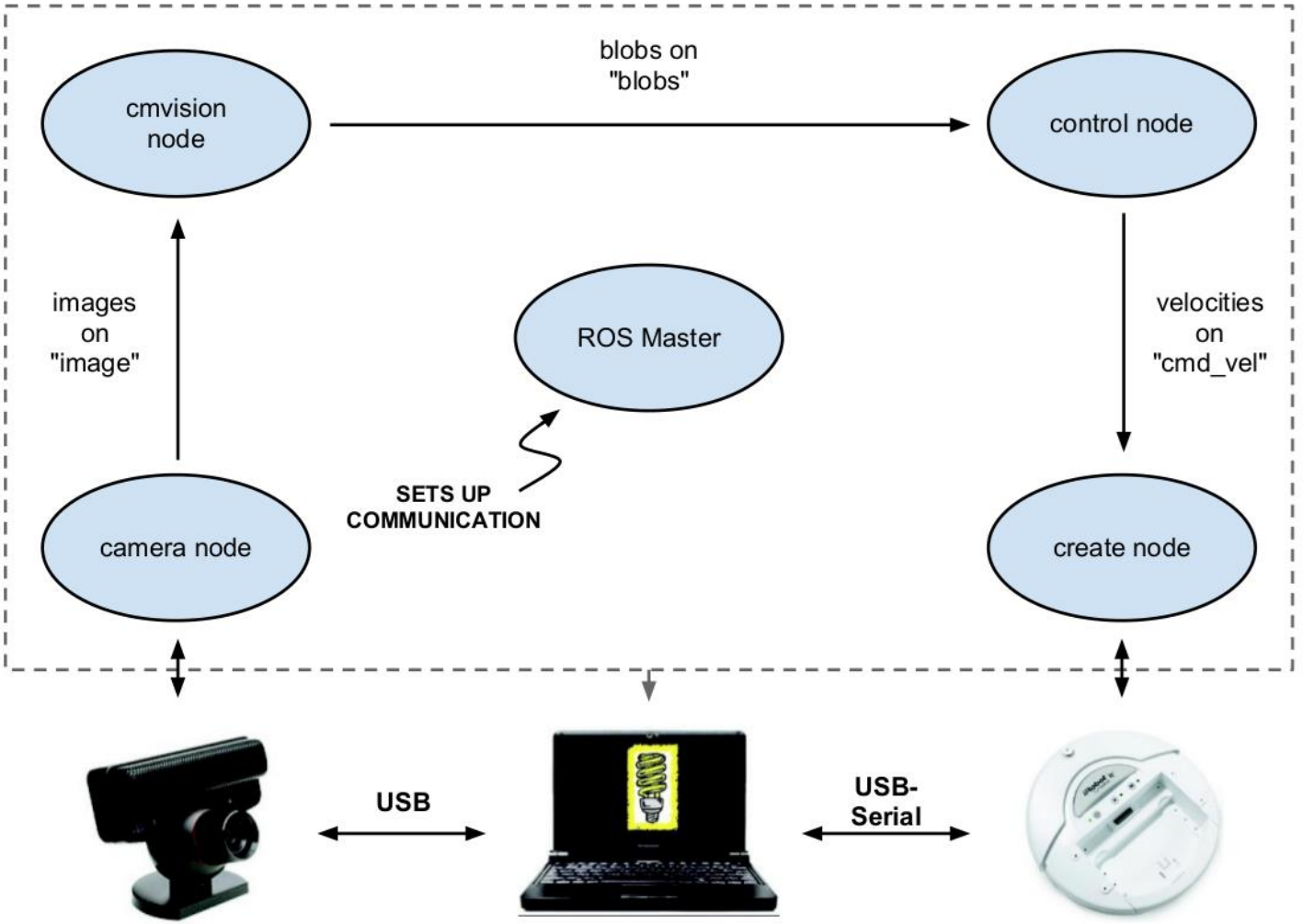
- run it by:
    $ roscore

```
SUMMARY
========

PARAMETERS
 * /rosdistro: indigo
 * /rosversion: 1.11.20

NODES

auto-starting new master
process[master]: started with pid [17250]
ROS_MASTER_URI=http://yamaha:11311/

setting /run_id to d0790a12-4828-11e6-a18f-64006a78d249
process[rosout-1]: started with pid [17263]
started core service [/rosout]
```

ROS - Node

- A node is a process that performs some computation.
- Typically we try to divide the entire software functionality into different modules - each one is run over a single or multiple nodes.
- Nodes are combined together into a graph and communicate with one another using streaming topics, RPC services, and the Parameter Server
- These nodes are meant to operate at a fine-grained scale; a robot control system will usually comprise many nodes

ROS - System Architecture



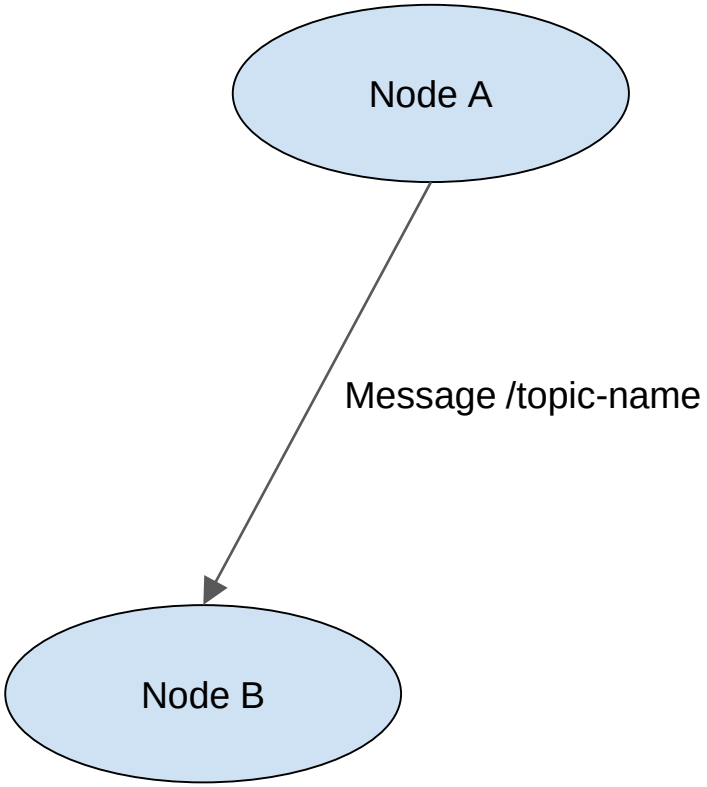
[adapted from slide by Chad Jenkins]

ROS - Message passing

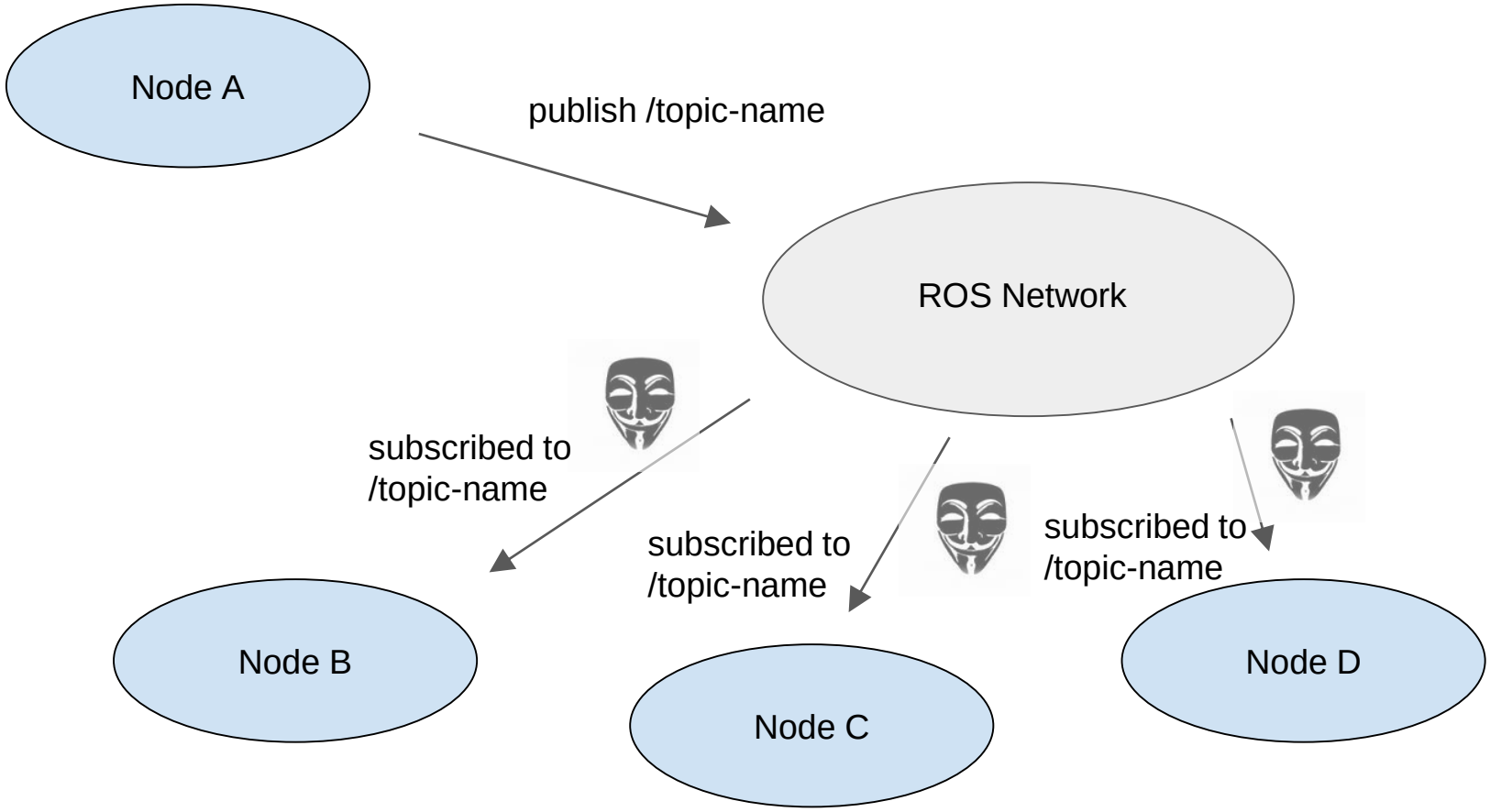
- Default by TCP, but UDP possible too

- Publish/Subscribe model:
  - data is sent by defined message type **(Topic)**
  - receiver gets only messages of subscribed topics
  - anonymous, receiver and sender doesn't know of each other

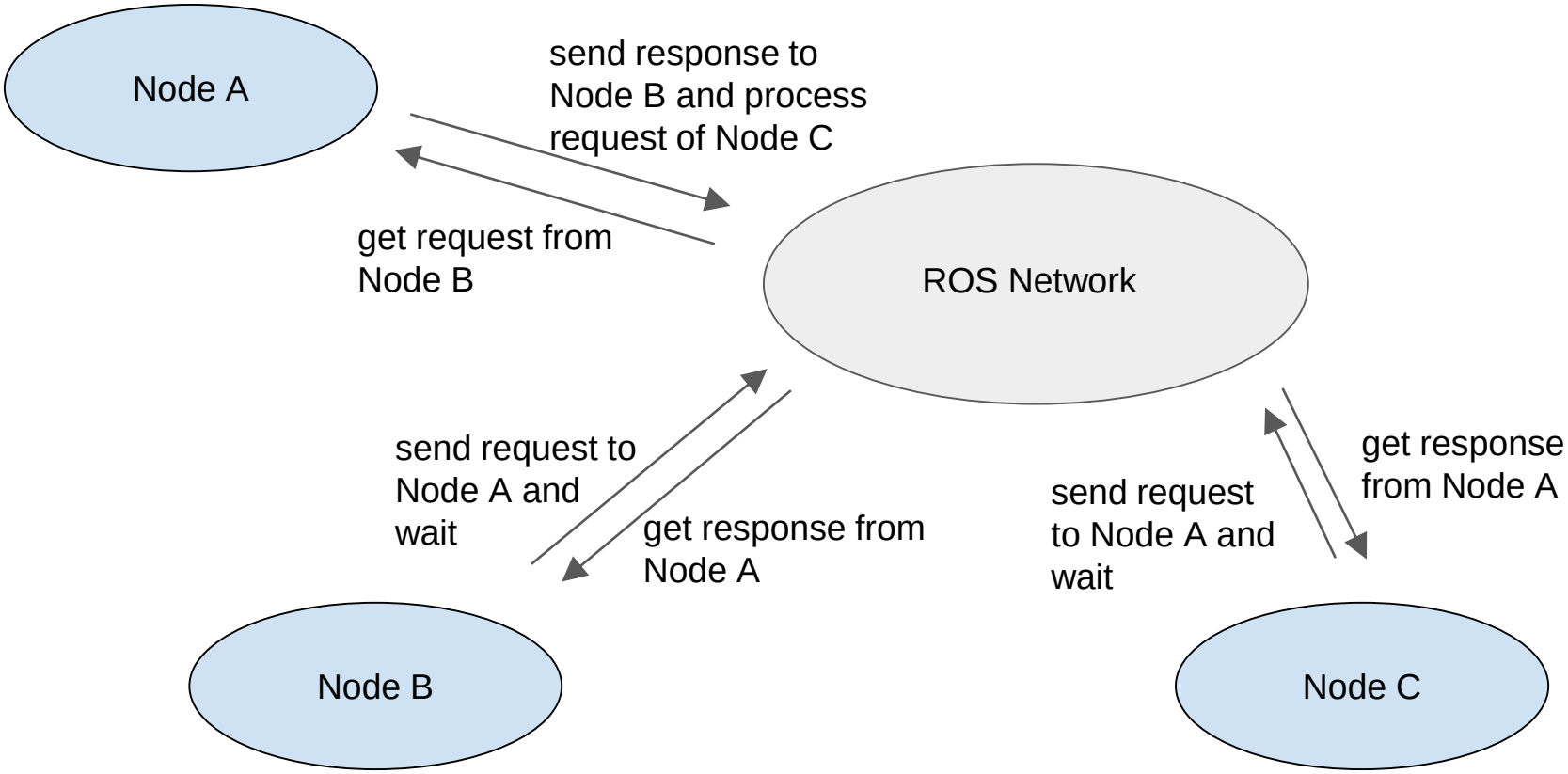
- Service model:
  - **service type** definitions (contains request and response topics)
  - server/client semantic:
    server waits for client calls on specific service name and then sends response



ROS - Publish/Subscribe



ROS - Service


Node A
send response to Node B and process request of Node C
get request from Node B
ROS Network
send request to Node A and wait
get response from Node A
send request to Node A and wait
get response from Node A
Node B
Node C

ROS - Topics

- Topics are named buses over which nodes exchange messages
    - same topics from different robots can be put into namespaces:
        - /robotA/speed
        - /robotB/speed

- Topics have anonymous publish/subscribe semantics - A node does not care which node published the data it receives or which one subscribes to the data it publishes

- There can be multiple publishers and subscribers to a topic

- Each topic is strongly typed by the ROS message it transports
    - Integer, String, Image, 3D-Pose etc.

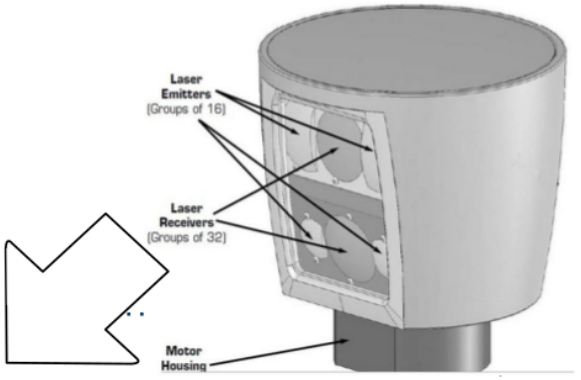
- Transport is done using TCP or UDP

ROS - Messages

- Nodes communicate with each other by publishing messages to topics
- A message is a simple data structure, comprising typed fields.
- Messages may also contain a special field called header which gives a timestamp and frame of reference

```
# Detected or simulated object
################################################################################

# Header
# header.frame_id defines reference frame
Header header

# Object frame (analogous to child_frame_id in nav_msgs/Odometry)
string object_frame_id


# ID for tracking
uint16 object_id

# duration this object has been tracked for
duration age

# duration since last update/confirmation by measurement
# (set to 0 as soon as a measurement update is available)
duration prediction_age


# odometry of the object (position, orientation, linear and angular velocities)
# odom.header.frame_id is header.frame_id
# child_frame_id is object_frame_id
nav_msgs/Odometry odom

# maximal size of the object (x,y,z) or (depth, width, height) [m]
# relative to the object frame, i.e. the orientation of the object is taken into
account
geometry_msgs/Vector3 size


# The contour points of the object [m]
geometry_msgs/Point[] contour_points
```

ROS - Example Publisher/Subscriber



roscore

velodyne | velodyne_common

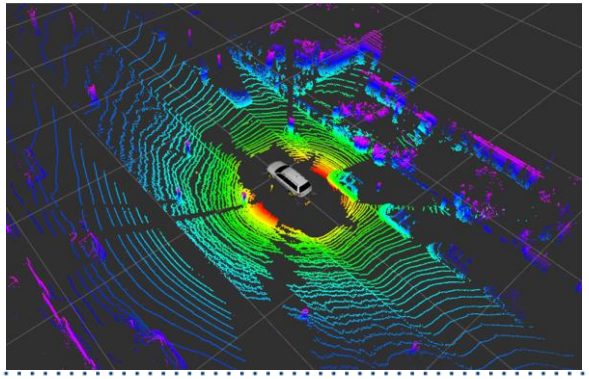
read

/velodyne/raw_scan

velodyne | velodyne_common

cloud

/velodyne/cloud
sensor_msgs::PointCloud



velodyne | velodyne_heightmap

height_map

/velodyne/obstacles
sensor_msgs::PointCloud

ROS - Example Publisher/Subscriber

```python
#!/usr/bin/env python

# RAW SCAN PUBLISHER

import rospy
from laser_scanner import ScannerDriver
from laser_scanner_msgs import ScannerRawMsg

# Initialize node
rospy.init_node("read_scan_node")

# Start driver
driver = ScannerDriver.load()

# Initialize publisher
publisher = rospy.Publisher("/velodyne/raw_scan", ScannerRawMsg)

while not rospy.is_shutdown():
    msg = driver.getData()
    publisher.publish(msg)
    rospy.sleep(0.5)    # sleep a half second
```

```python
#!/usr/bin/env python

# RAW SCAN SUBSCRIBER / POINTCLOUD PUBLISHER

import rospy
from sensor_msgs import PointCloud
from laser_scanner_msgs import ScannerRawMsg
from laser_scanner_decoder import PointCloudConverter


def callback(raw_msg):
    point_cloud_msg = PointCloudConverter.convert(raw_msg)
    publisher.publish(point_cloud_msg)

# Initialize node
rospy.init_node("point_cloud_node")

# Run subscriber
rospy.Subscriber("/velodyne/raw_scan", ScannerRawMsg, callback)
publisher = rospy.Publisher("/velodyne/cloud", PointCloud)

# spin() simply keeps python from exiting until this node is stopped
rospy.spin()
```

ROS - Example Service

```python
#!/usr/bin/env python

# REQUEST WITH RAW SCAN / RESPONSE WITH POINT CLOUD

import rospy
from sensor_msgs import PointCloud
from laser_scanner_msgs import ScannerRawMsg
from laser_scanner_services import HeightMapService, HeightMapServiceRequest, HeightMapServiceResponse
from laser_scanner_decoder import HeightMapConverter


def callback(request):
    point_cloud_msg = HeightMapConverter.convert(request.raw_msg)
    return HeightMapServiceResponse(point_cloud_msg)

# Initialize node
rospy.init_node("height_map_node")

# Run subscriber
rospy.Service("/velodyne/get_height_map", HeightMapService, callback)

# spin() simply keeps python from exiting until this node is stopped
rospy.spin()


```

ROS - rostopic command

- List all currently advertised topics

```
$ rostopic list
/model_car/yaw
/scan
/usb_cam/image_raw
/usb_cam/image_compressed
...
```

ROS - rostopic command

- Get data type of topic

```
$ rostopic type /model_car/yaw
std_msgs/Float32
```

ROS - rostopic command

- Subscribe to topic and print out content

```
$ rostopic echo /scan
```


```
header:
  seq: 1373
  stamp:
    secs: 137
    nsecs: 400000000
  frame_id: base_laser_link
angle_min: -2.35619449615
angle_max: 2.35619449615
angle_increment: 0.052948191762
time_increment: 0.0
scan_time: 0.0
range_min: 0.0
range_max: 5.0
ranges: [1.8039969205856323, 1.7785133123397827, 1.798710584640503, 1.8245443105697632, 1.8139444589614868, 1.8081258535385132, 1.8516205549240112,
 1.857111930847168, 1.868300437927246, 1.9343751668930054, 1.9605001211166382, 2.0474910736083984, 2.0933594703674316, 2.1237003803253174, 2.232451
6773223877, 2.340627908706665, 2.3985755443573, 2.622255563735962, 2.7261335849761963, 2.89371395111084, 3.28377628326416, 3.6559736728668213, 4.36
3884925842285, 5.0, 5.0, 5.0, 5.0, 5.0, 5.0, 5.0, 5.0, 5.0, 5.0, 4.877347469329834, 5.0, 5.0, 5.0, 1.5433844327926636, 1.3364330530166626, 1.163908
60080719, 1.0746499300003052, 1.021209955215454, 0.9772032499313354, 0.9312910437583923, 0.8900294899940491, 0.9040037989616394, 0.8717102408409119
, 0.8307997584342957, 0.8073323369026184, 0.8310391902923584, 0.8131420612335205, 0.8191946148872375, 0.8273055553436279, 0.8170567750930786, 0.849
764347076416, 0.8441159725189209, 0.8809255957603455, 0.8800222277641296, 0.9216613173484802, 0.966161847114563, 1.0139291286468506, 1.065423369407
6538, 1.1626973152160645, 1.2782059907913208, 1.47539222240448, 5.0, 5.0, 4.7716965675354, 4.389033317565918, 4.121235370635986, 3.9568285942077637
, 3.7908356189727783, 3.634945869445801, 3.4623069763183594, 3.3141911029815674, 3.1623823642730713, 2.9592769145965576, 2.8229079246520996, 2.6797
64747619629, 2.5945394039154053, 2.498901844024658, 2.434621334075927 7, 2.399679660797119, 2.351926565170288, 2.2914767265319824, 2.258337259292602
5, 2.2117855548858643, 2.191577911376953, 2.1975369453430176, 2.2097389698028564]
intensities: [1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 0.0, 0.0, 0.0, 0.0
, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0,
1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 0.0, 0.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.
0, 1.0, 1.0, 1.0, 1.0]
---
header:
  seq: 1374
  stamp:
    secs: 137
    nsecs: 500000000
  frame_id: base_laser_link
angle_min: -2.35619449615
angle_max: 2.35619449615
```

ROS - rosrun command

- Start a node (roscore must be already running!)

```
                # for python scripts
$ rosrun package_name script.py



# for c++ executables
$ rosrun package_name executable_name
```

ROS - roslaunch command

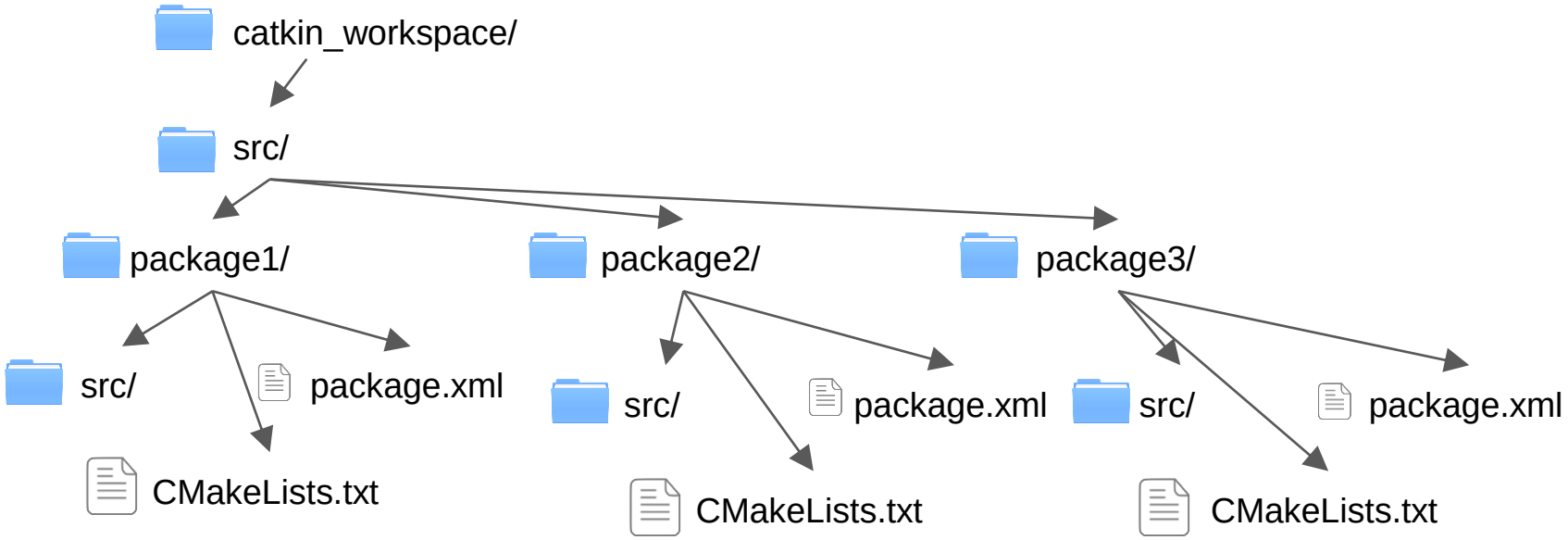
- Do I have to open 100 terminals, when I want to run 100 nodes?
  - Use a .launch file instead!

```
$ roslaunch package_name start_all.launch
```

```xml
<?xml version="1.0" encoding="UTF-8" ?>
<launch>

    <node name="node_a" pkg="package1" type="node_a.py">
    </node>

    <node name="node_b" pkg="package1" type="node_b.py">
    </node>

    <node name="node_c" pkg="package2" type="subscriber.py">
    </node>

</launch>
```

ROS - Workspace

- A workspace contains the source code of packages
- Workspace allows to compile all packages with one command
- Directory structure:



ROS - Build packages

- For compiling there exists several tools: **rosbuild, catkin_make, catkin**
- We use the newest one: **catkin**

http://catkin-tools.readthedocs.io/en/latest/migration.html

- Build all packages in the workspace:

```
$ cd path/to/workspace
$ catkin build
```

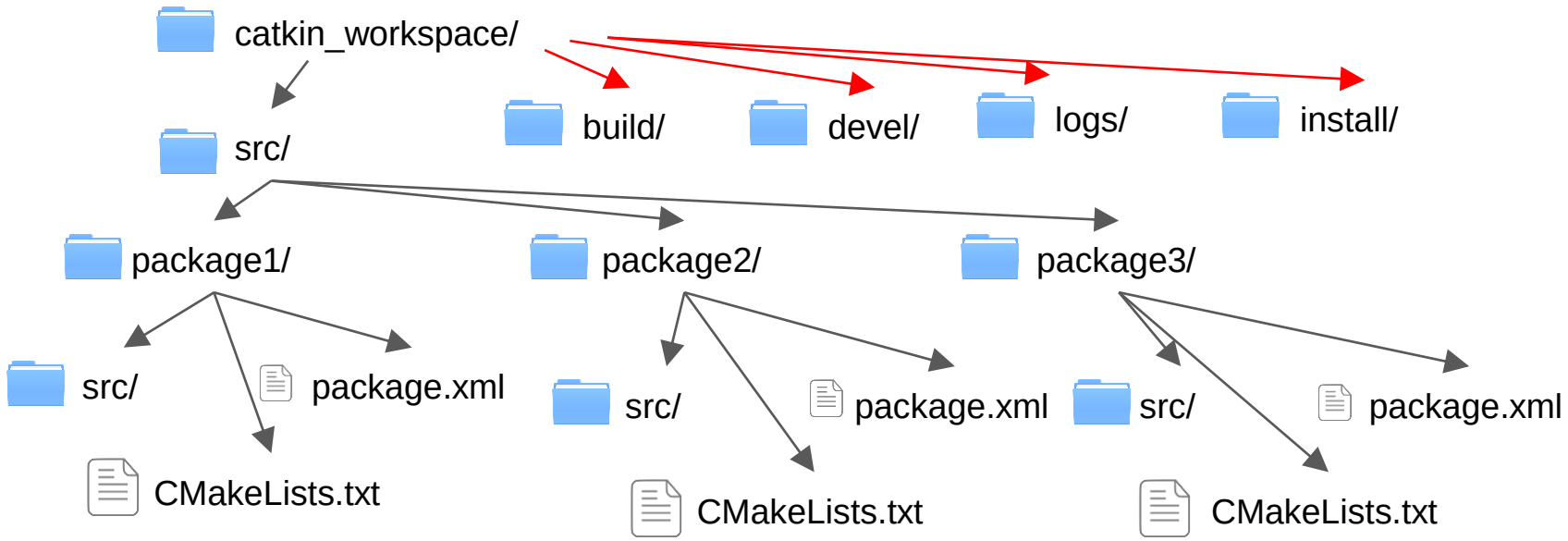
- Build one or more packages:

```
$ cd path/to/workspace
$ catkin build package2 package3
```

- Catkin uses internally the cmake → make toolchain for compiling (that's why a CMakeLists.txt for each package is needed)

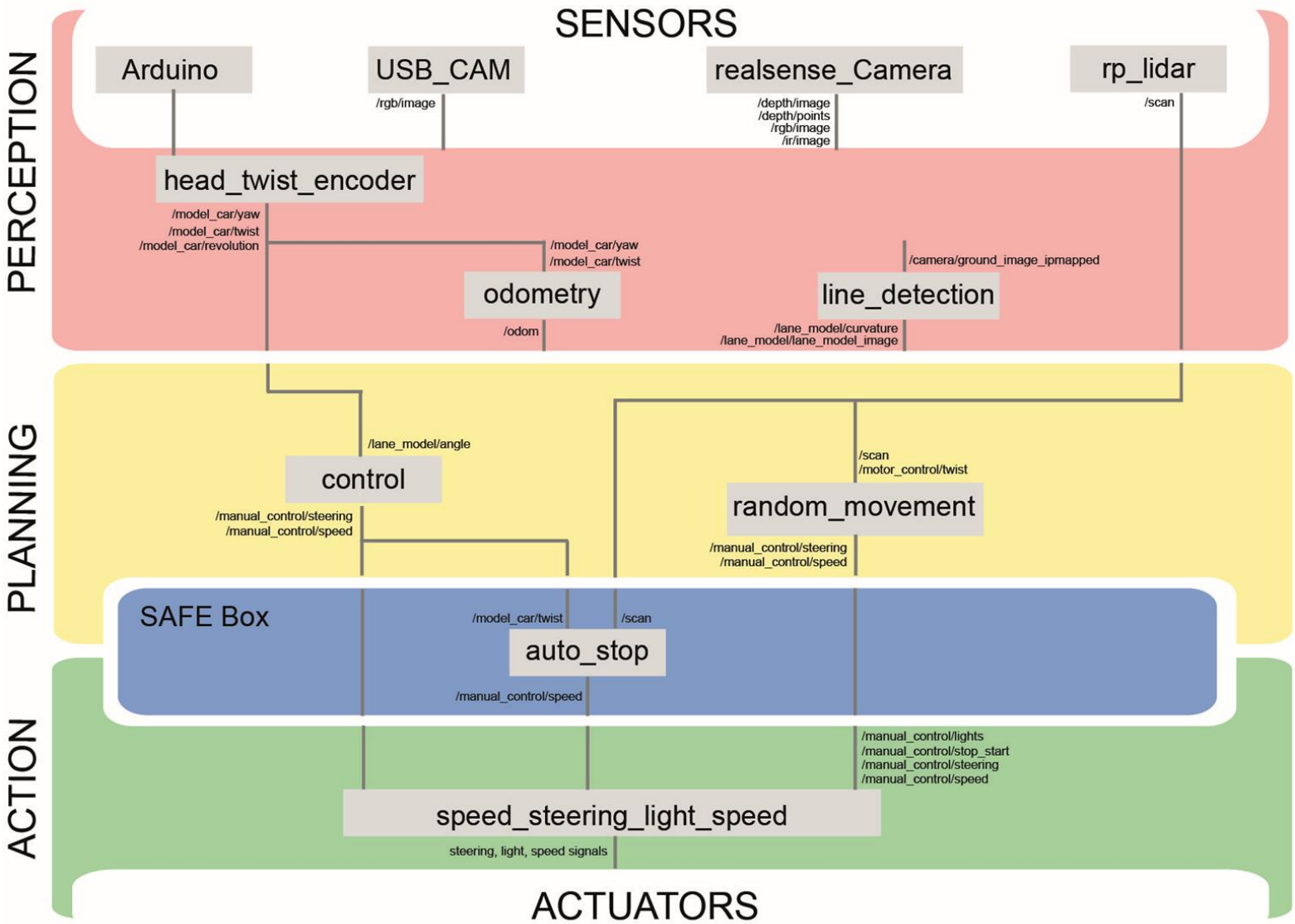
ROS - Workspace

- After completed build, catkin created additional folders which contains the compiled files



- **build/** contains intermediate files
- **logs/** contains log files of compiling messages
- depending on catkin workspace configuration, compiled results are in **devel/** or **install/** directory (devel for development is the default setting)

ROS - Model Car Project packages



References

- Introduction to ROS Programming - UT Computer Science
  - http://www.cs.utexas.edu/~todd/cs378/slides/Week8a.pdf

- ROS Homepage
  - http://www.ros.org/