University of Ljubljana
Faculty of Electrical Engineering

COLLABORATIVE ROBOTICS CENTRE

robolab

FE-ROS 2021
ROS orodja

Sebastjan Šlajpah

Univerza v Ljubljani
Fakulteta za elektrotehniko
Laboratorij za robotiko

sebastjan.slajpah@fe.uni-lj.si

www.robolab.si
www.cobotic.si

Vsebina

- Nodes
- Topics
- Services
- Msg & Srv files
- Actions
- Parameter
- Launch files

- Konzolni ukazi za posamezne funkcionalnosti

- Python 2.7

robolab

ROSMASTER

ROSMASTER

```
>> roscore
```

- Povezava med posameznimi funkcionalnostmi
- Lahko je samo en naenkrat
- Povezava med več ROS sistemi

```
>> echo $ROS_MASTER_URI
```

CATKIN WORKSPACE

- CATKIN – official build system for ROS

```
>> mkdir catkin_ws
>> cd catkin_ws
>> mkdir src
>> catkin_make
```

CATKIN WORKSPACE

- Povezava konzole z ROS spremenljivkami

```
>> cd devel
>> source setup.bash
```

- Dodaj v bashrc.sh (avtomatsko, ko se odpre konzola)

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

Generiranje sistema

- Kodo pišeš v **`/catkin_ws/src`** mapi

```
>> catkin_make
```

robolab

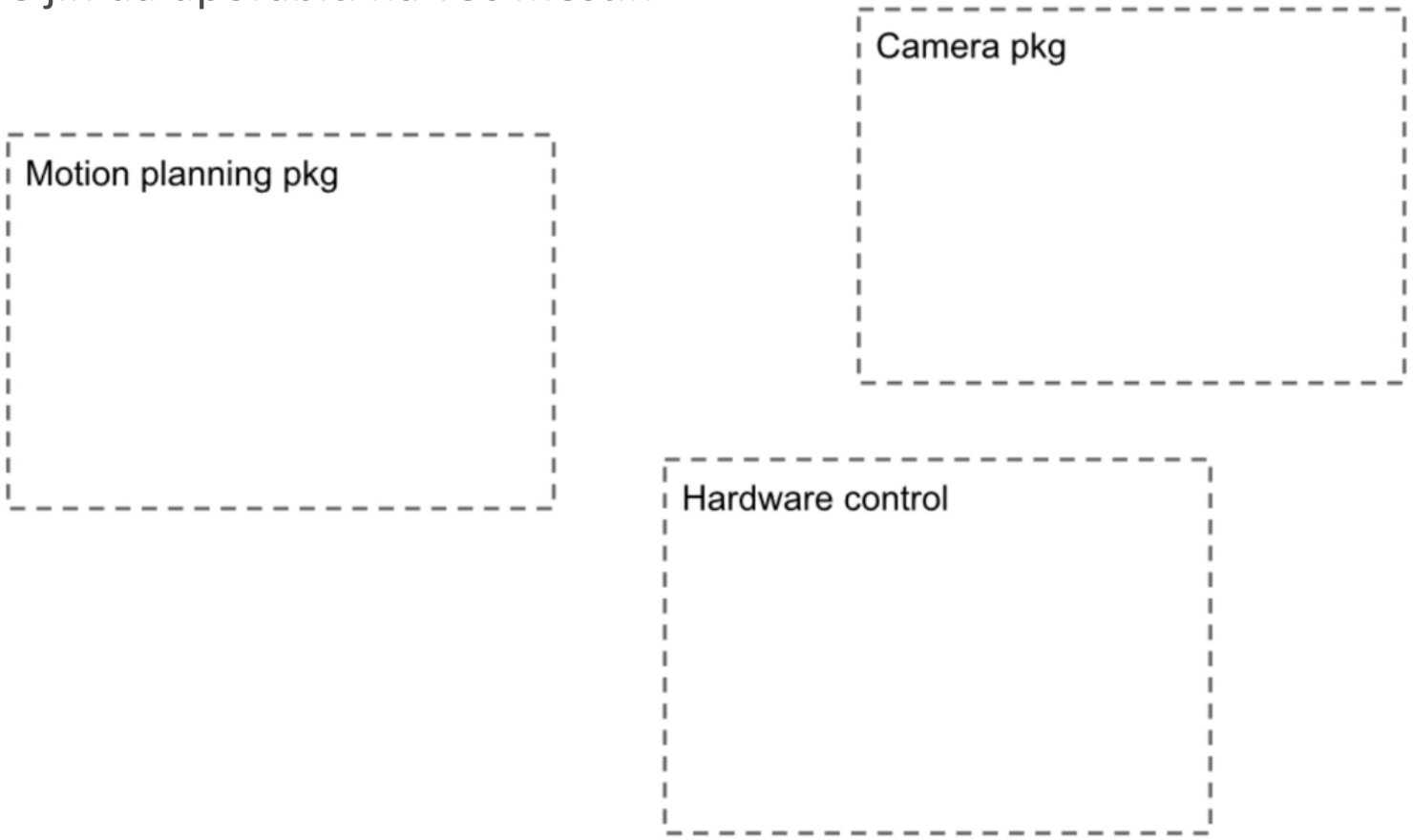
PACKAGES

Packages

- Neodvisne enote, ki se jih da uporabiti na več mestih

Packages

- Neodvisne enote, ki se jih da uporabiti na več mestih



Nov paket

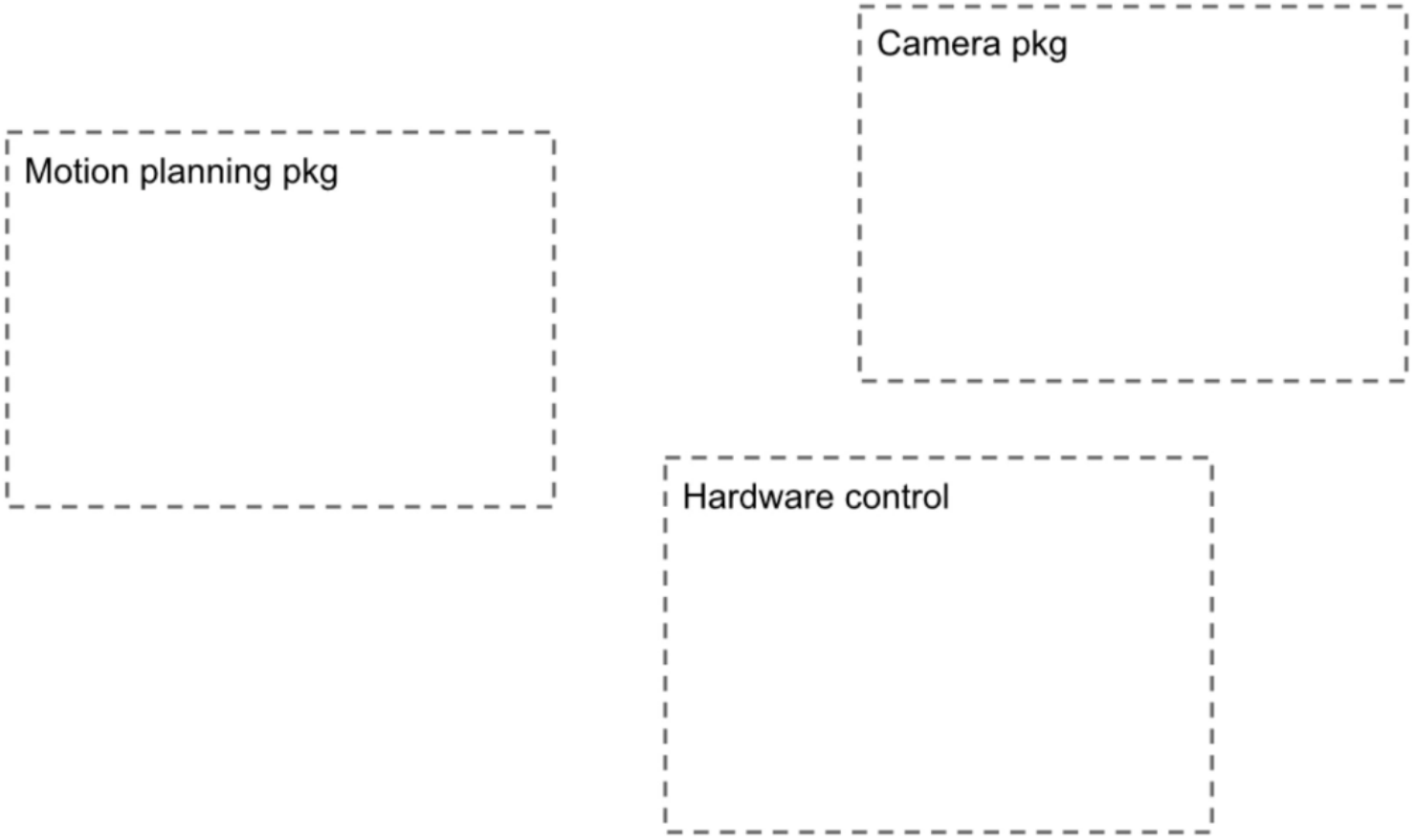
```
>> catkin_create_pkg <ime_paketa> <razširitve>

>> catkin_make
```

```
>> catkin_create_pkg rpi_feros rospy std_msgs actionlib_msgs
```

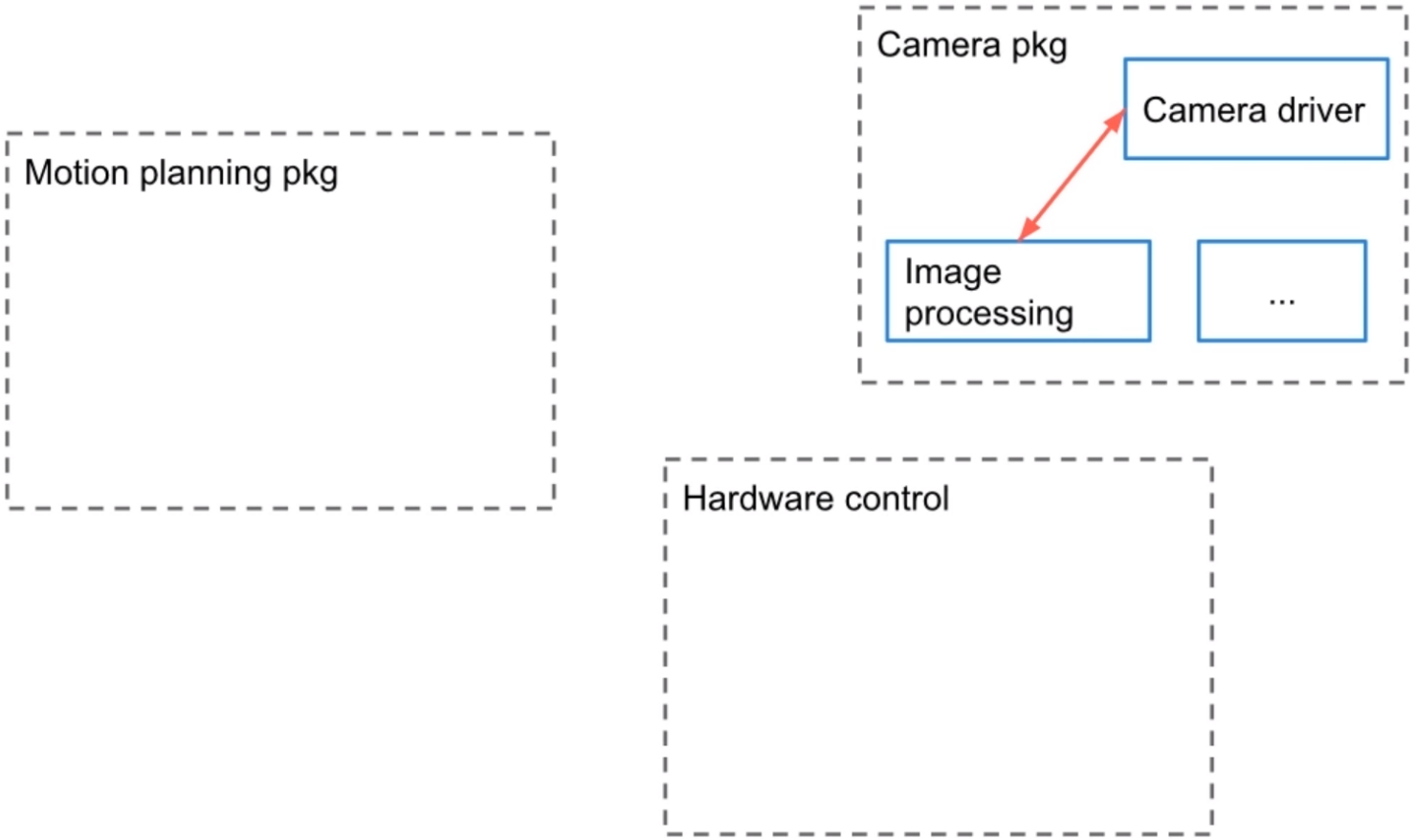
robolab

NODE

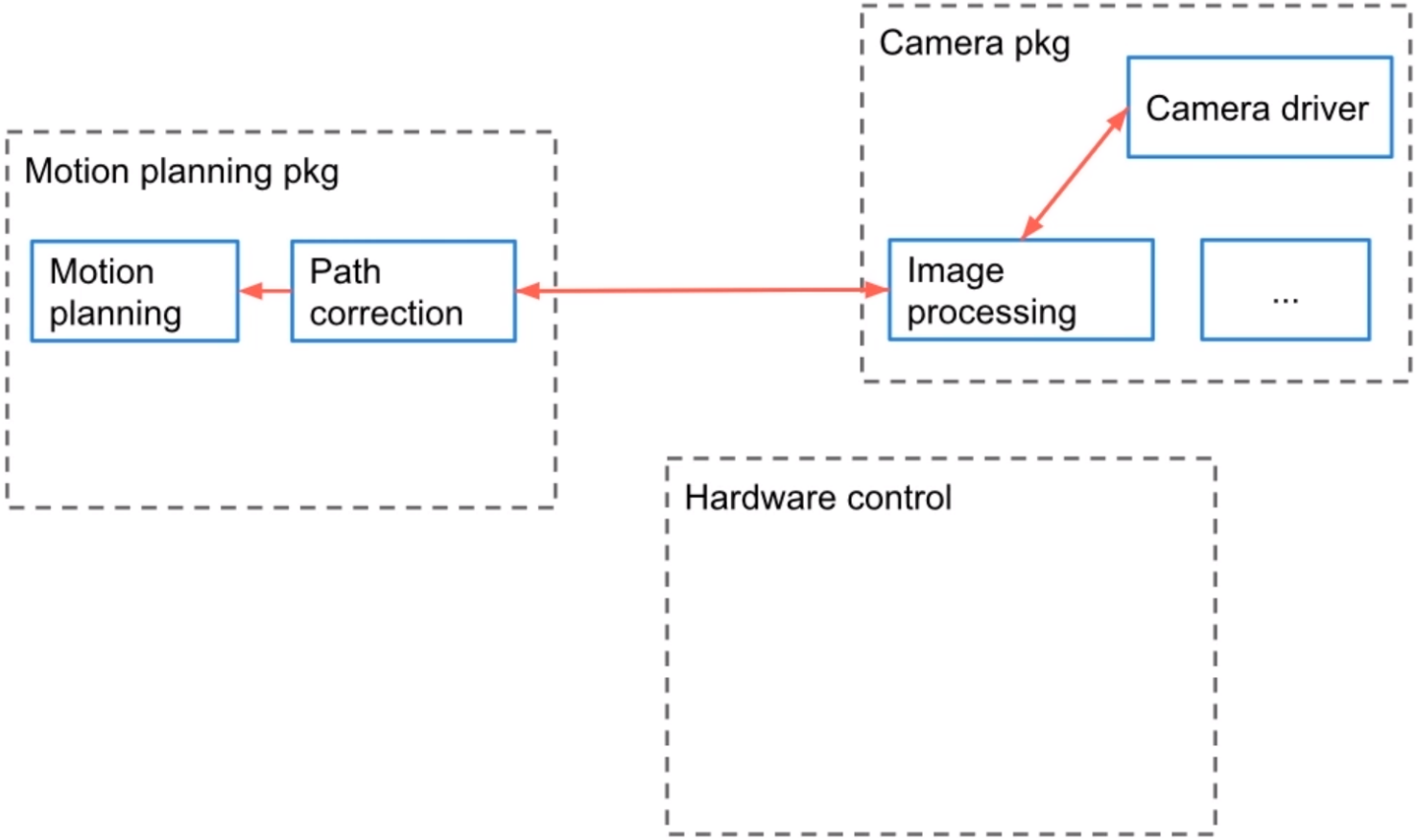
Node



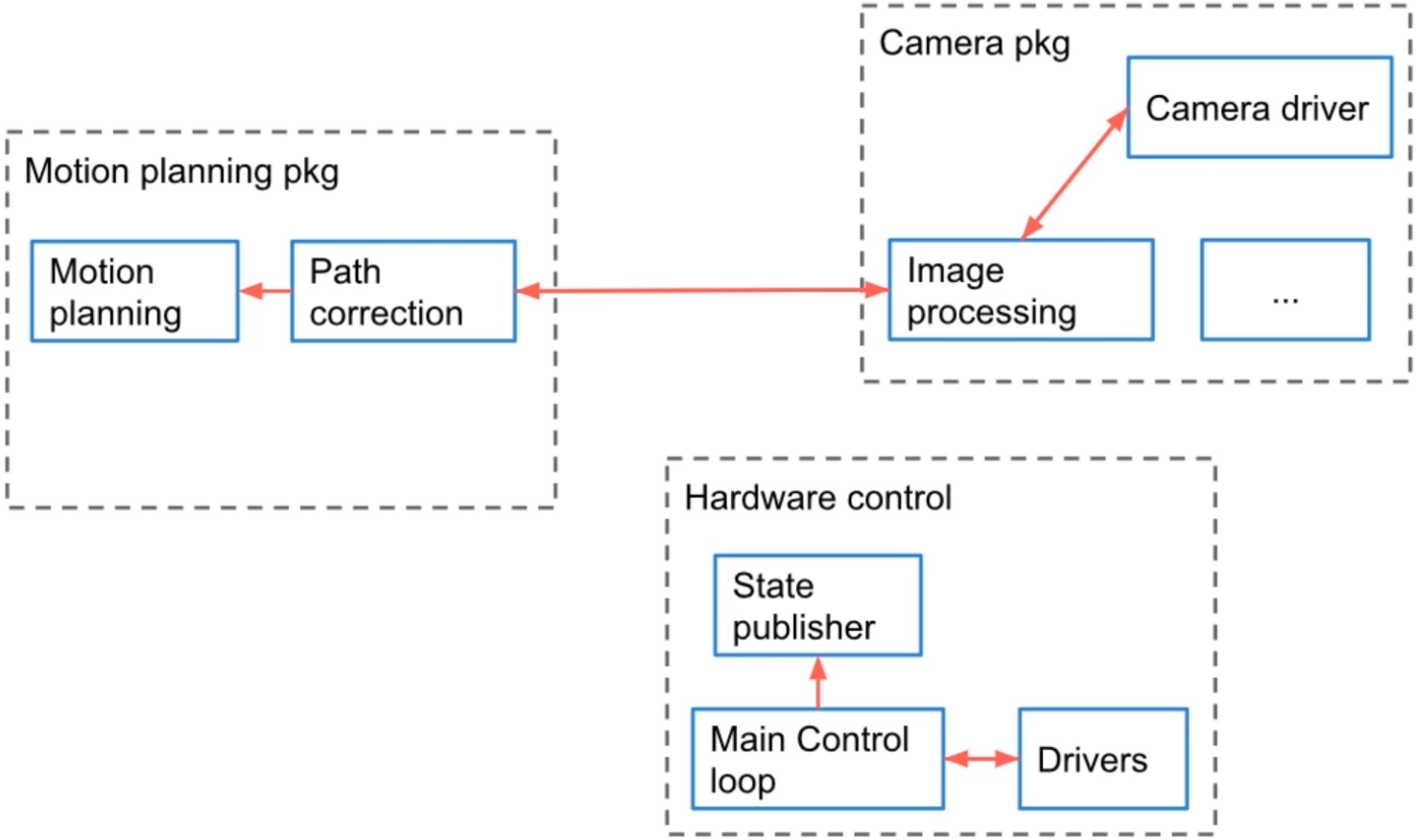
Node


Motion planning pkg
Camera pkg
Camera driver
Image processing
...
Hardware control

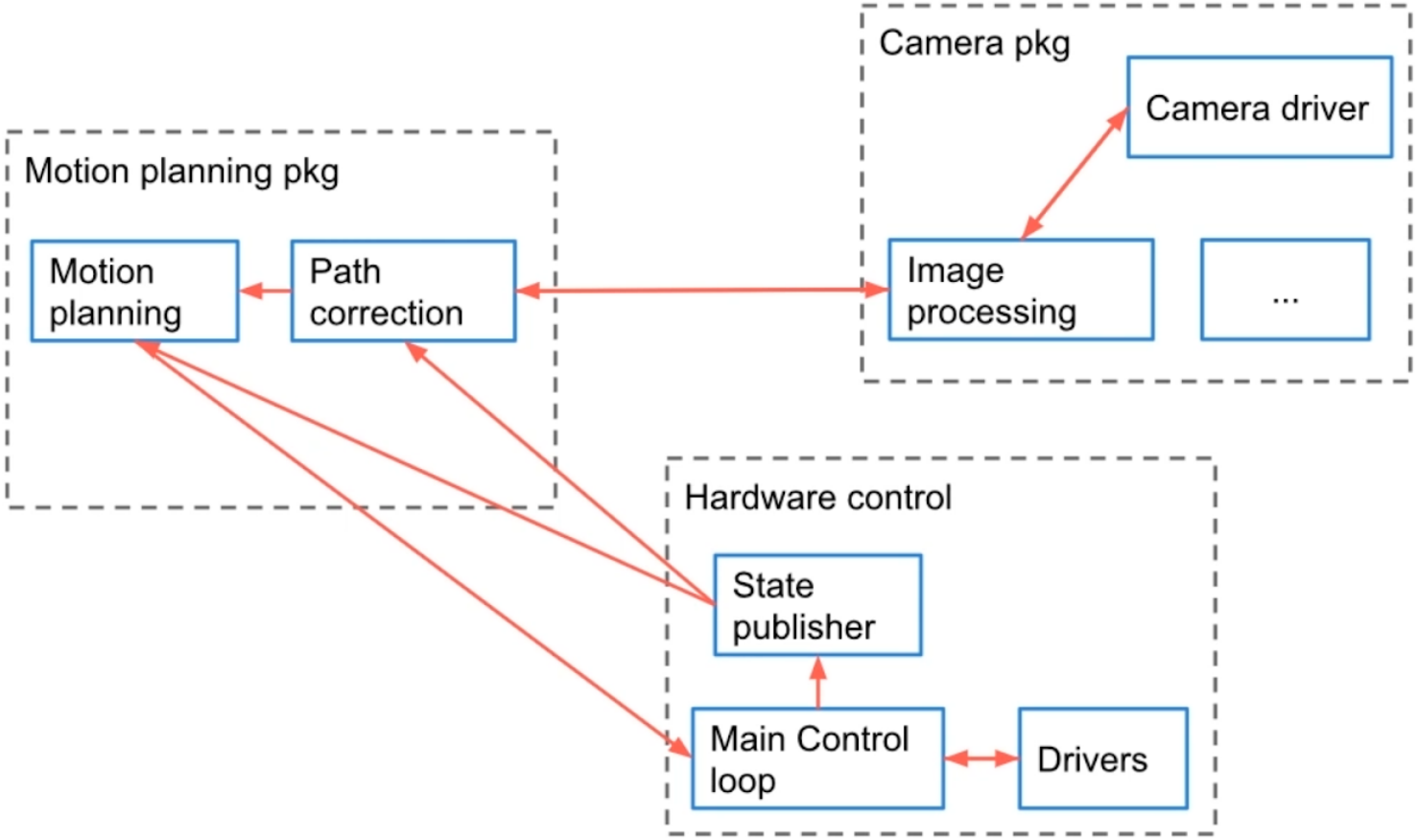
Node


Camera pkg
Camera driver
Motion planning pkg
Motion planning
Path correction
Image processing
...
Hardware control

Node


Camera pkg
Camera driver
Motion planning pkg
Motion planning
Path correction
Image processing
...
Hardware control
State publisher
Main Control loop
Drivers

Node


Camera pkg
Camera driver
Motion planning pkg
Motion planning
Path correction
Image processing
...
Hardware control
State publisher
Main Control loop
Drivers

Node

- proces, ki izvaja računanje
- program, ki teče znotraj robotske aplikacije
- združeni so v pakete
- med seboj komunicirajo (topics, servers, actions, parameter servers)

- zmanjšujejo kompleksnost kode
- koda je bolj odporna na napake
- uporaba različnih programskih jezikov

robolab

Nov Node

```
>> mkdir scripts
>> cd scripts
>> touch my_first_node.py
>> chmod +x my_first_node.py
>> code my_first_node.py
```

robolab

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

import rospy

if __name__ == '__main__':
    rospy.init_node('my_first_python_node')

    rospy.loginfo('This node has been started.')
    rospy.sleep(1)
    print('Exit now')
```

>> python my_first_node.py

DEBUG Node

- `rosrun <pkg name> <node name>` … poganjanje
- `rosnode list` … seznam vseh aktivnih
- `rosnode info <node name>` … info o node
- `rosnode kill <node name>` … ugasni node
- `rosnode ping <node name>` … ping (preveri, če deluje)

Node

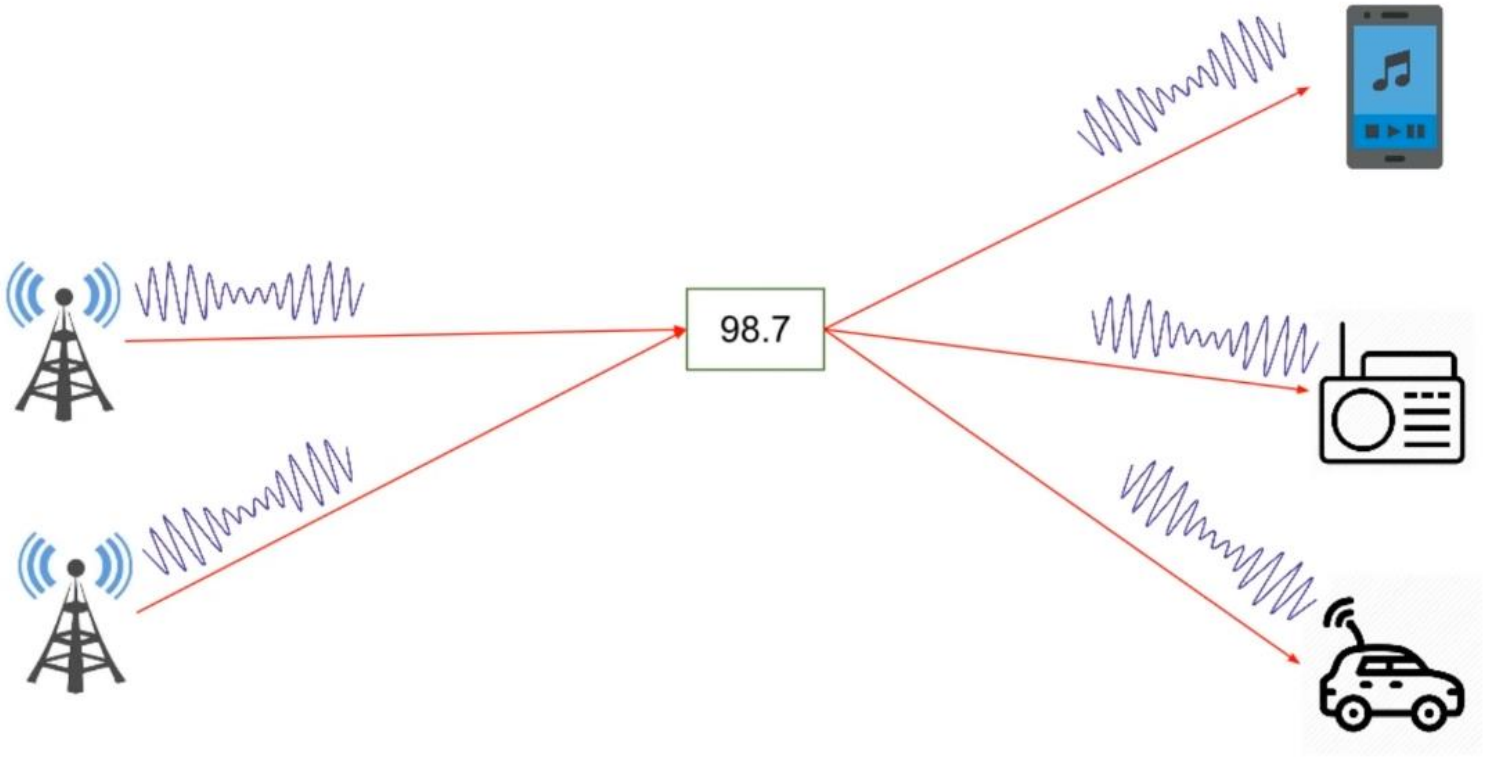
- naenkrat se lahko izvaja samo en node z določenim imenom
- če želiš več istih, jih je potrebno preimenovati

```
rospy.init_node('my_first_python_node', anonymous=True)
```

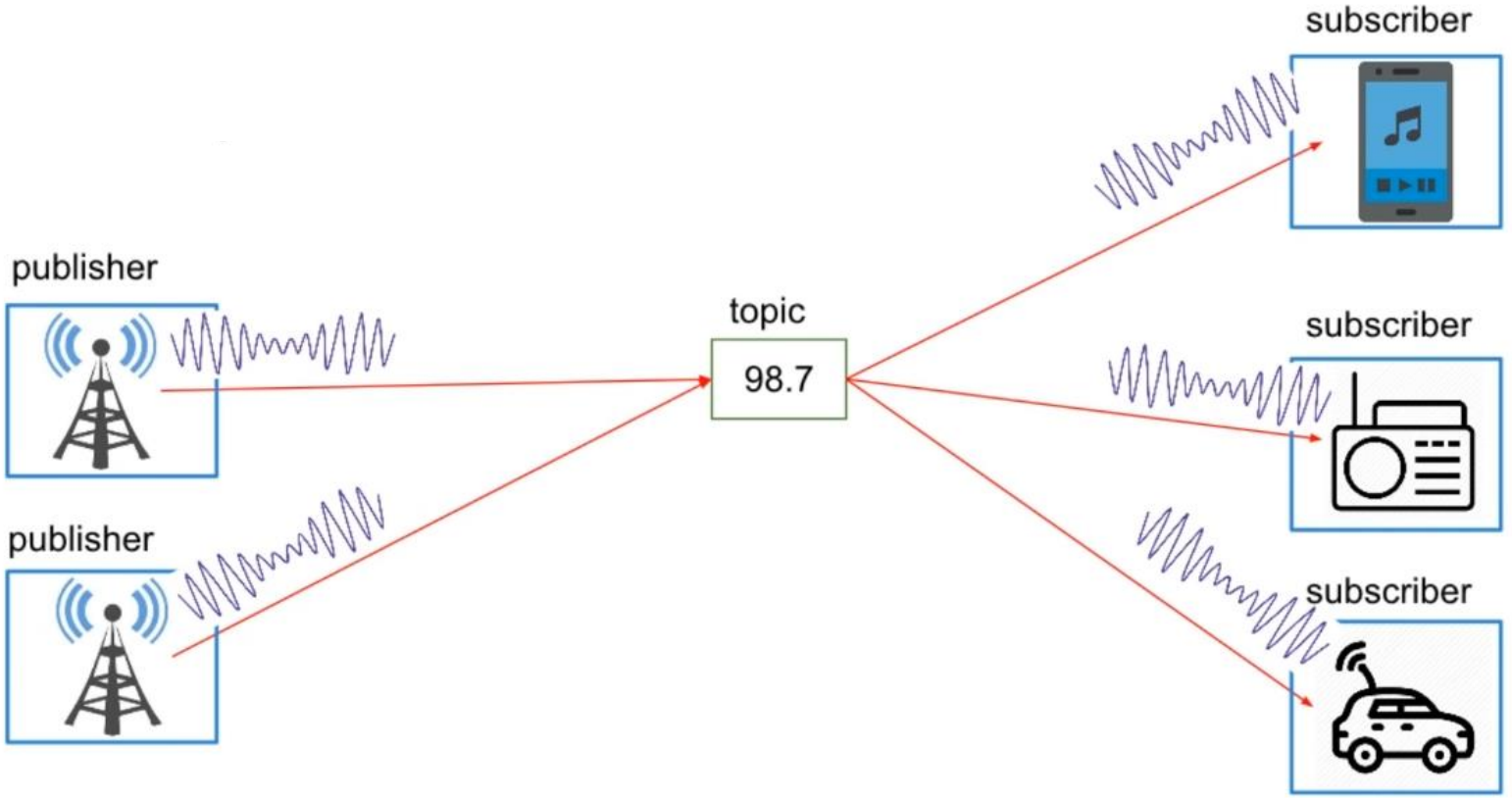
robolab

TOPICS

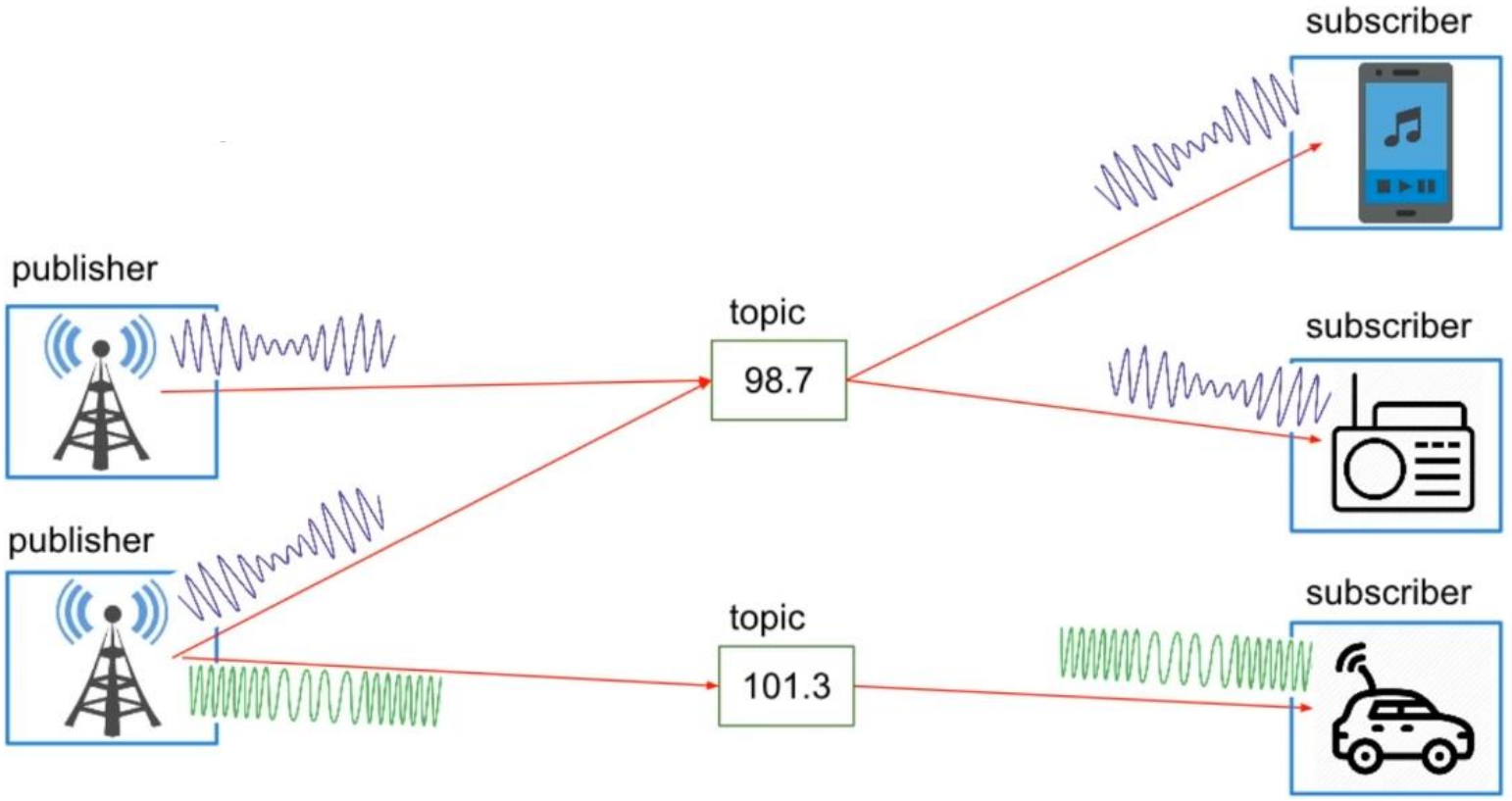
Topic



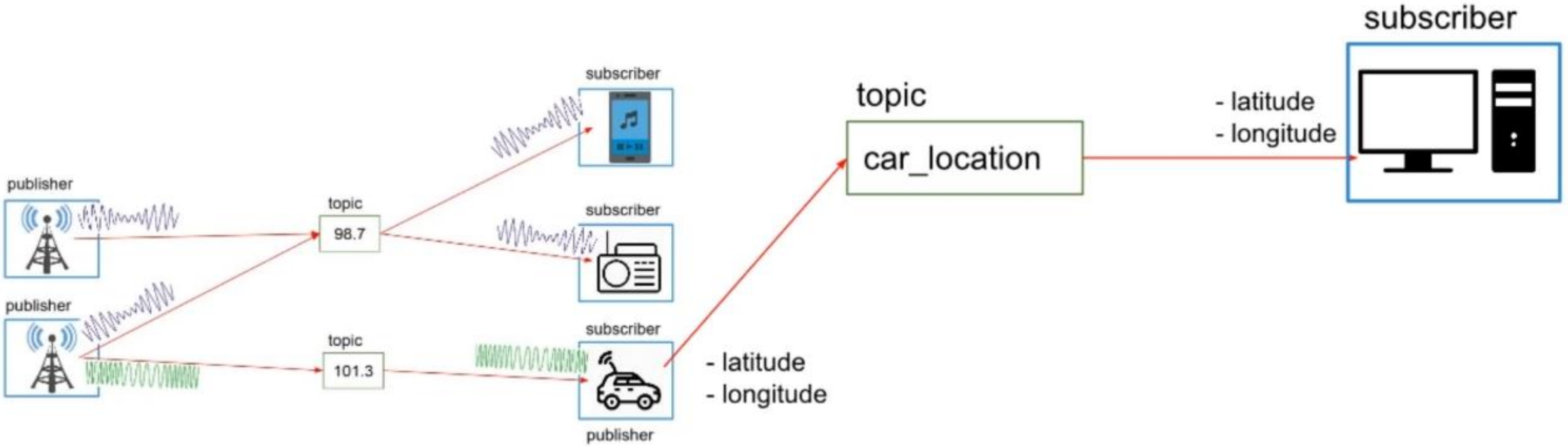
Topic


publisher
publisher
topic
98.7
subscriber
subscriber
subscriber

Topic


subscriber
publisher
topic
98.7
subscriber
publisher
topic
101.3
subscriber

Topic


publisher
publisher
topic
98.7
topic
101.3
subscriber
subscriber
subscriber
publisher
- latitude
- longitude
topic
car_location
- latitude
- longitude
subscriber

robolab

Topic

- Vodilo, preko katerega si nodi izmenjujejo sporočila
- Enosmerni prenos sporočil (publisher > subscriber)
- Anonimno
- Topic ima svoj tip sporočila
- ROS master skrbi za ustrezni povezavo publisher/subscriber
- Vsak node ima lahko več publishers/subscribers za različne topics

Publisher

```
pub = rospy.Publisher('topic_name', msg_type, queue_size=10)
```

Messages types:

http://wiki.ros.org/std_msgs

robolab

ROS Message Types

Bool
Byte
ByteMultiArray
Char
ColorRGBA
Duration
Empty
Float32
Float32MultiArray
Float64
Float64MultiArray
Header
Int16
Int16MultiArray
Int32
Int32MultiArray
Int64
Int64MultiArray
Int8
Int8MultiArray
MultiArrayDimension
MultiArrayLayout
String
Time
UInt16
UInt16MultiArray
UInt32
UInt32MultiArray
UInt64
UInt64MultiArray
UInt8
UInt8MultiArray

Subscriber

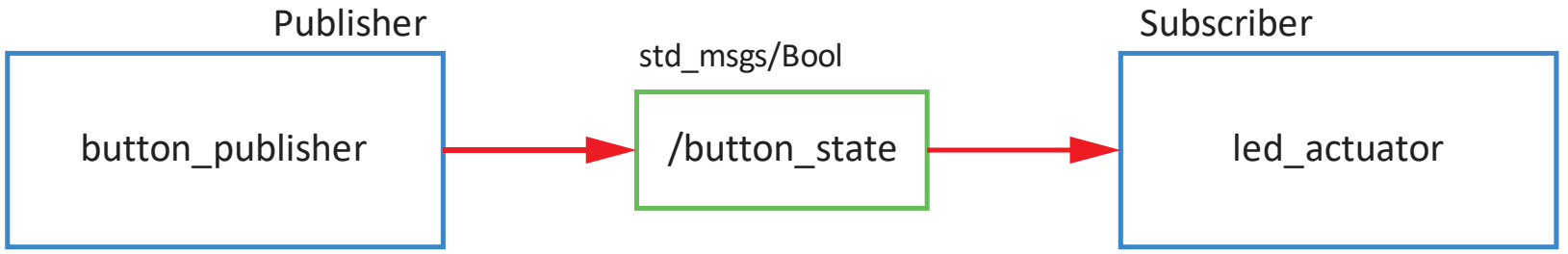
```
sub = rospy.Subscriber('topic_name', msg_type, callback_fcn)
```

DEBUG Topic

- `rostopic -h`
- `rostopic list`
- `rostopic echo <ime topica>`
- `rostopic info <ime topica>` ... kateri tip pošilja
- `rostopic pub <ime topica>` + Tab za autocomplete
  - `-1` ... Enkrat pošlje
  - `-r 5` ... Pošilja s 5 Hz

Primer

- ob pritisku tipke prižgi LED



Rpi GPIO

```
import RPi.GPIO as GPIO

# Green 1 - GPIO 2
# Green 2 - GPIO 3
# Yellow 1 - GPIO 4
# Yellow 2 - GPIO 5
# Red 1 - GPIO 6
# Red 2 - GPIO 7

# Gumb 1 - GPIO 11
# Gumb 2 - GPIO 12
```

```
def resetLed():
    # nastavi in resetiraj vse LED
    for ii in range(2,8):
        GPIO.setup(ii,GPIO.OUT)
        GPIO.output(ii,False)

GPIO.setmode(GPIO.BCM)
GPIO.setup(BTN_GPIO, GPIO.IN)
btn_state = GPIO.input(BTN_GPIO)
GPIO.cleanup()
```

SICK NanoScan3 – namestitev

```
>> sudo apt-get install ros-melodic-sick-safetyscanners

>> source /opt/ros/melodic/setup.bash
>> cd ~/catkin_ws/src/
>> git clone https://github.com/SICKAG/sick_safetyscanners.git
>> cd ..
>> catkin_make install
>> source ~/catkin_ws/install/setup.bash
```

SICK NanoScan3

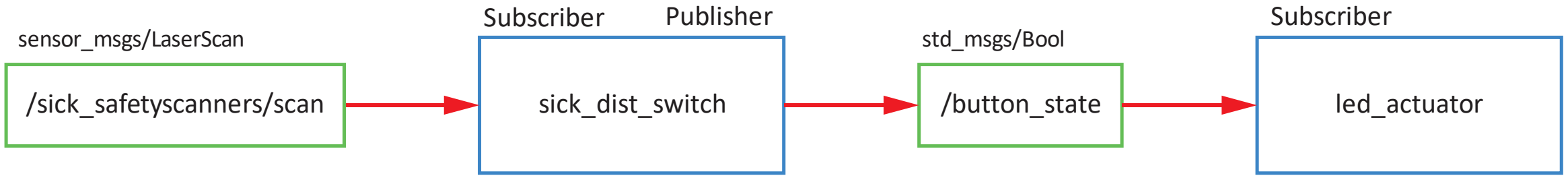
- 1651 meritev
- Korak meritev: 0,002909 rad
- Kot skeniranja: 275°

- Topic: `/sick_safetyscanners/scan`
- Msg: `from sensor_msgs.msg import LaserScan`

```
>> roslaunch sick_safetyscanners sick_safetyscanners.launch
sensor_ip:=192.168.65.YYY host_ip:=192.168.65.XX
```

Naloga

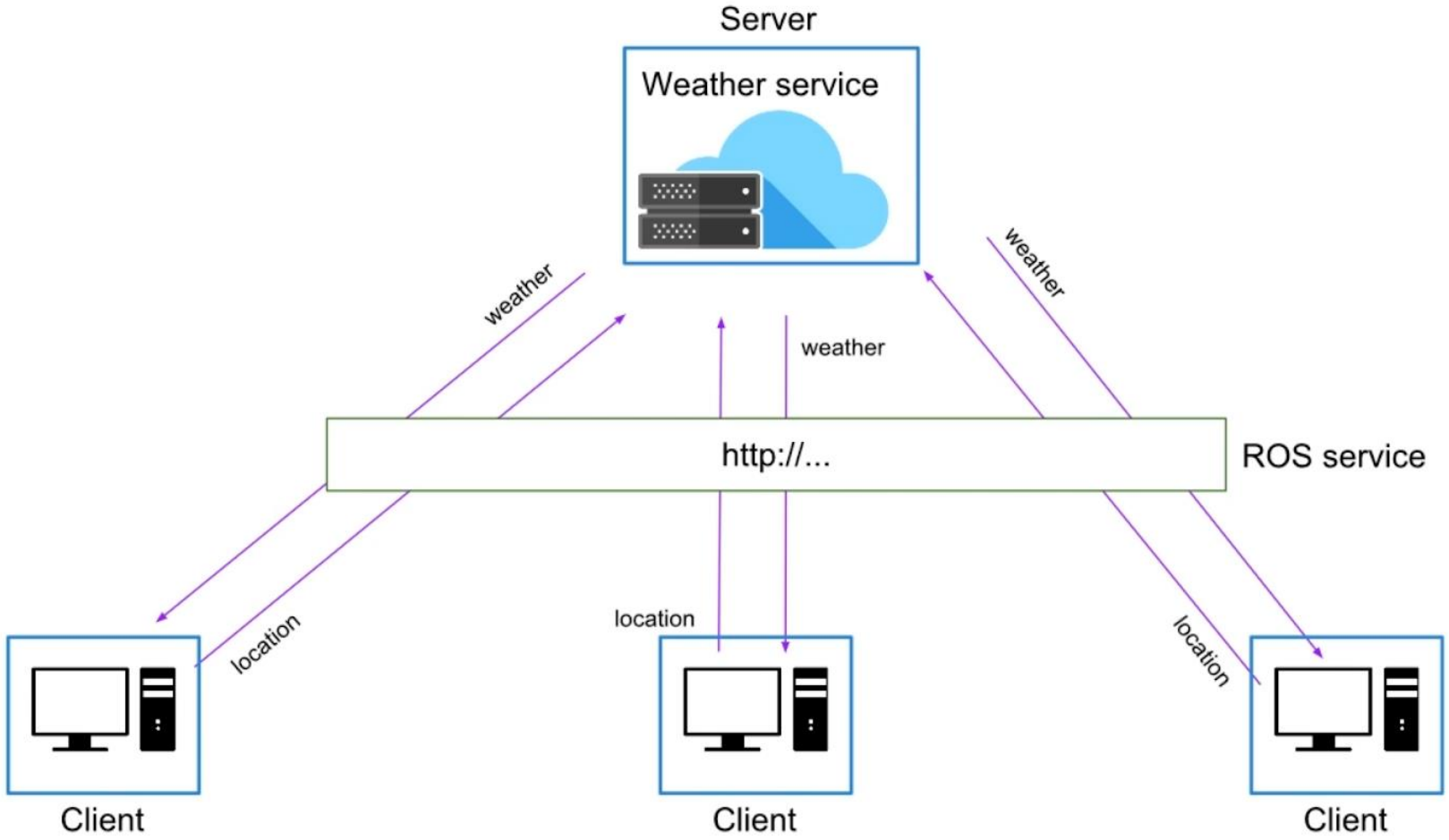
- prižgi LED, ko je objekt bližje kot 0,2 m



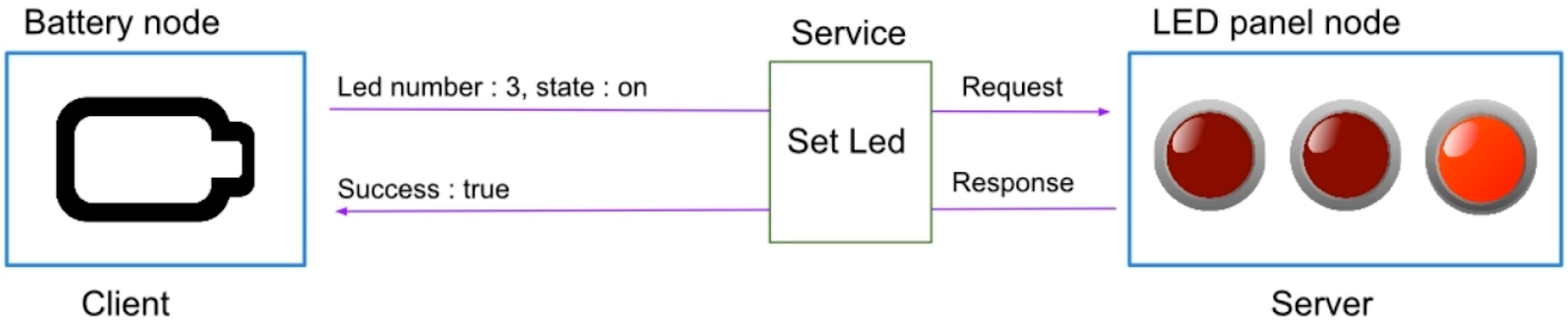
robolab

SERVICES

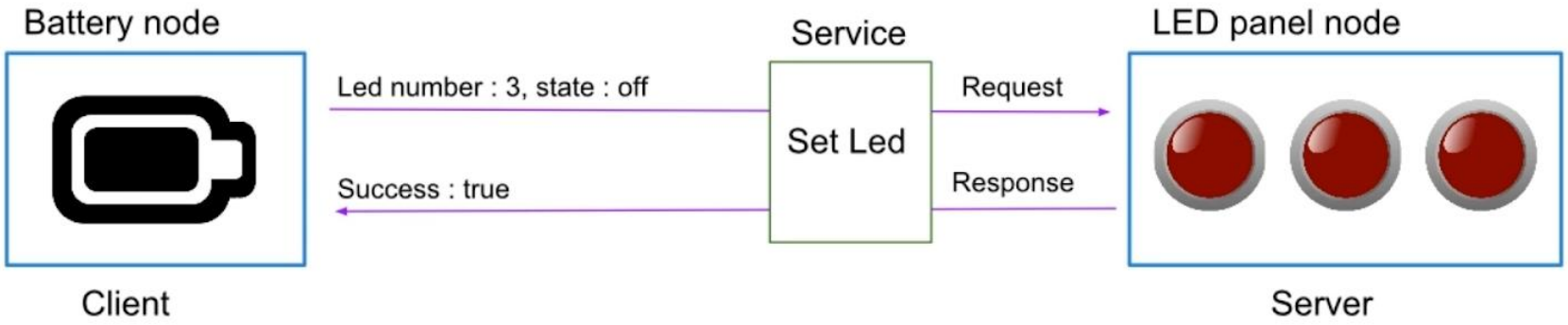
Services


Server
Weather service
weather
weather
weather
http://...
ROS service
location
location
location
Client
Client
Client

Services


Battery node
Service
LED panel node
Led number : 3, state : on
Request
Set Led
Success : true
Response
Client
Server

Services


Battery node
Service
LED panel node
Led number : 3, state : off
Request
Set Led
Success : true
Response
Client
Server

robolab

Services

Services

- Sistem server/klient
- Sinhrono delovanje
- Za izračunavanje in hitre akcije
- En tip sporočila za Request, drug tip sporočila za Response
- Server je samo eden, ki lahko odgovarja več klientom

Server

```
service = rospy.Service('ime_service', msg_type, handle_fcn)
```

Klient

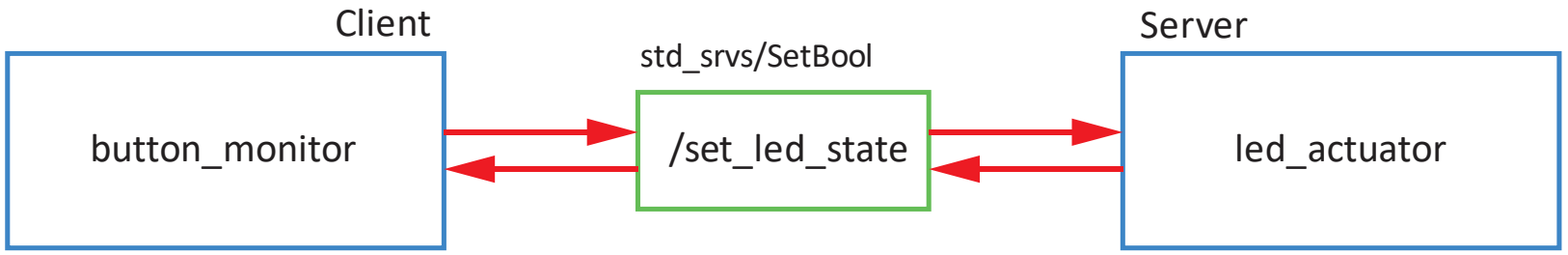
```
rospy.wait_for_service('ime_service')

try:
    client = rospy.ServiceProxy('ime_service',msg_type)
    ...
except rospy.ServiceException as e:
    rospy.logwarn('Service failed' + str(e))
```

DEBUG Services

- `rosservice list` … vse registrirane services
- `rosservice info <ime service>` … info o service
- `rosservice call <ime service>` … klic iz konzole (brez klienta)

Primer

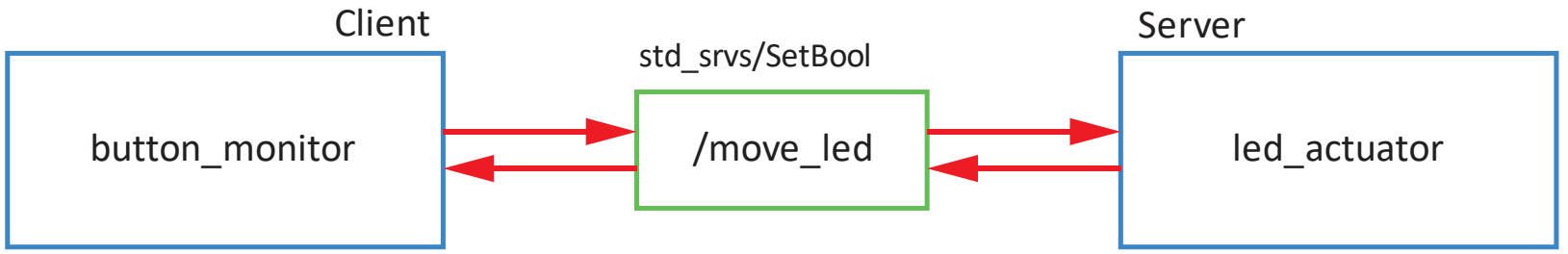
- prižgi/ugasni LED ob pritisku tipke (prekinitve)




```
GPIO.add_event_detect(BTN_GPIO, GPIO.RISING, callback=btn_cbk, bouncetime=500)
```

Naloga

- s tipkami premikajte prižgano LED levo/desno



robolab

MSG & SRV

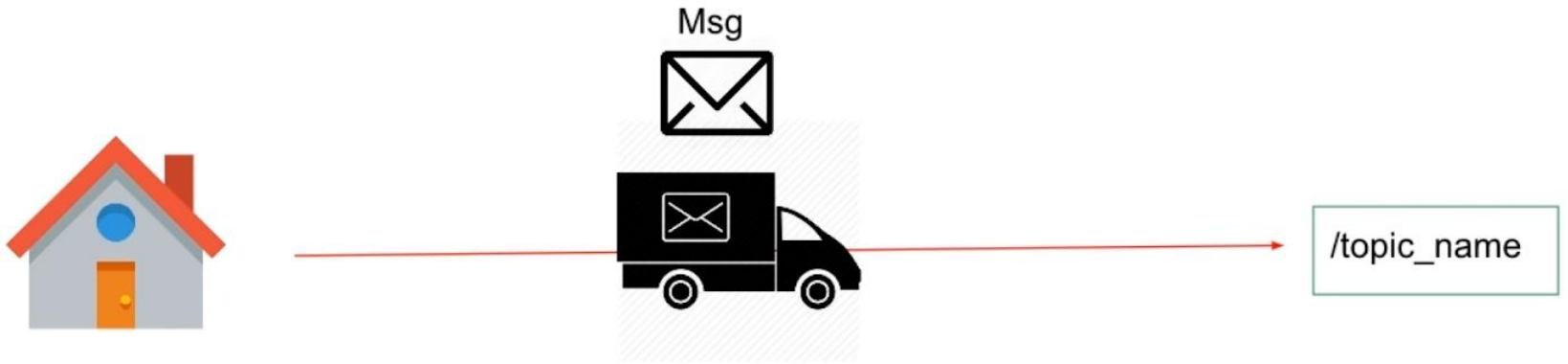
MSG in SRV

- Topic:
  - Ime: `/sick_safetyscanners/scan`
  - Definicija sporočila MSG: `sensors_msgs/LaserScan`

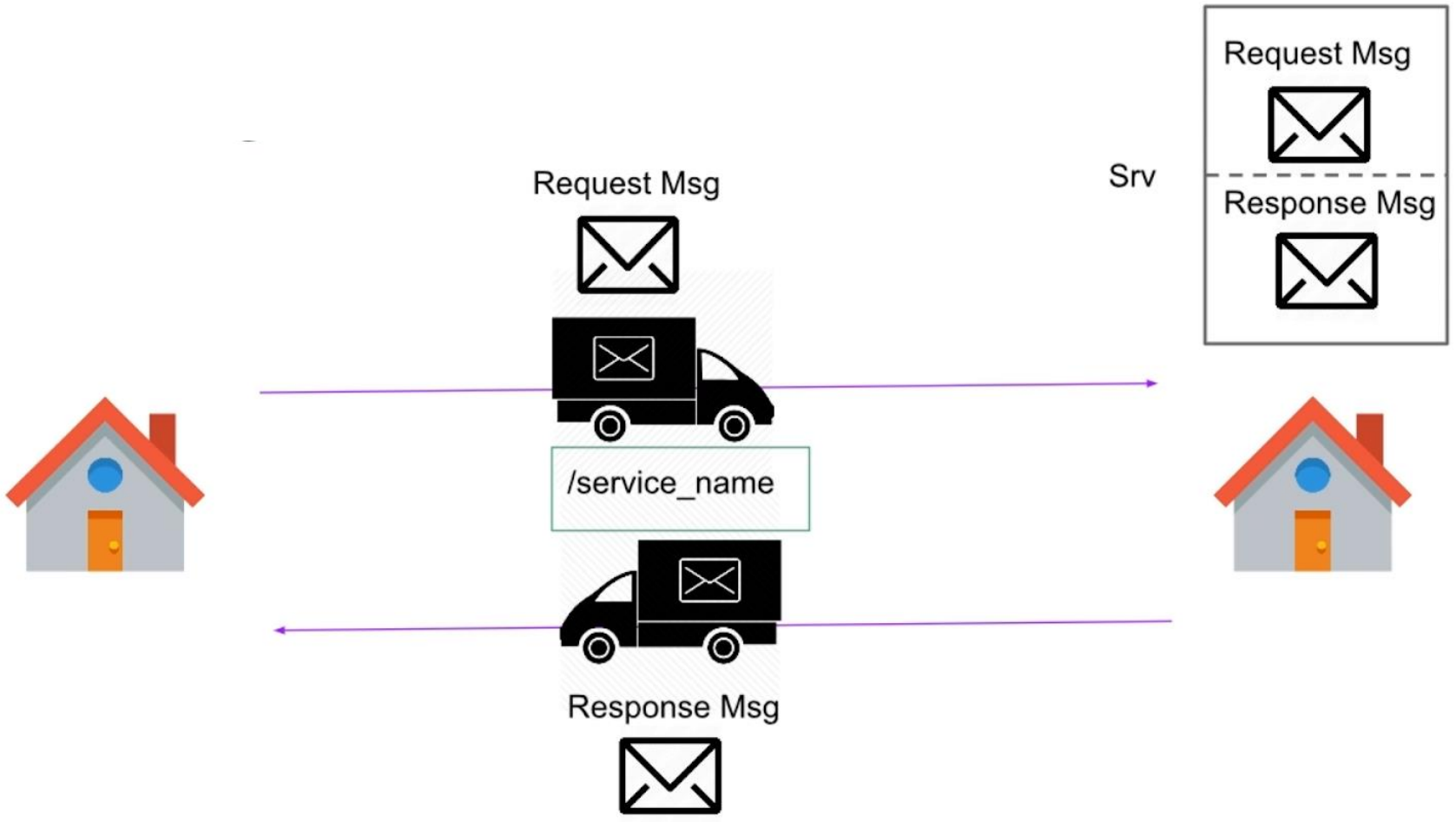
- Service:
  - Ime: `/set_led_state`
  - Definicija sporočila SRV: `std_srvs/SetBool`
    - Request: MSG
    - Response: MSG

```
Request: msg
---
Response msg
```

MSG



SRV


Request Msg
Srv
Request Msg
Response Msg
/service_name
Response Msg

MSG in SRV

- Uporaba MSG primitivov za definiranje sporočil
- Sporočila se lahko definira z uporabo obstoječih sporočil

- MSG:
  - std_msgs
  - sensor_msgs
  - geometry_msgs
  - actionlib_msgs
  - …
- SRV:
  - std_srvs
  - …

SRV

```
Request
---
Response
```

Nov MSG in SRV

```
>> catkin_create_pkg rpi_msgs rospy std_msgs
```

package.xml (rpi_msgs)

```
<build_depend>message_generation</build_depend>

<exec_depend>message_runtime</exec_depend>
```

CMakeLists.txt (rpi_msgs)

```
find_package(catkin REQUIRED COMPONENTS
  rospy
  std_msgs
  message_generation
)

…

# Generate messages in the 'msg' folder
add_message_files(
  FILES
  IME_SPOROCILA.msg
)
```

CMakeLists.txt (rpi_msgs)

```
# Generate added messages and services with any dependencies listed here
generate_messages(
  DEPENDENCIES
  std_msgs
)
...
catkin_package(
#  INCLUDE_DIRS include
#  LIBRARIES my_robot_msgs
 CATKIN_DEPENDS rospy std_msgs message_runtime
#  DEPENDS system_lib
```

Nov MSG (rpi_msgs)

```
>> roscd rpi_msgs
>> mkdir /msg
>> touch ledStatus.msg
>> code ledStatus.msg
```

```
int64 ledNumber
string ledStatus
```

```
>> catkin_make
```

Uporaba

package.html (rpi_feros)

```
<depend>my_robot_msgs</depend>
```

CMakeLists.txt (rpi_feros)

```
find_package(catkin REQUIRED COMPONENTS
 rospy
 std_msgs
 rpi_msgs
)
```

Nov SRV (rpi_msgs)

```
>> roscd rpi_msgs
>> mkdir /srv
>> touch safetyZone.srv
>> code safetyZone.srv
```

```
int16 zone
---
bool success
string message
```

```
>> catkin_make
```

Uporaba

- CMakeLists.txt (rpi_feros)

```
# Generate services in the 'srv' folder
add_service_files(
  FILES
  safetyZone.srv
)
```

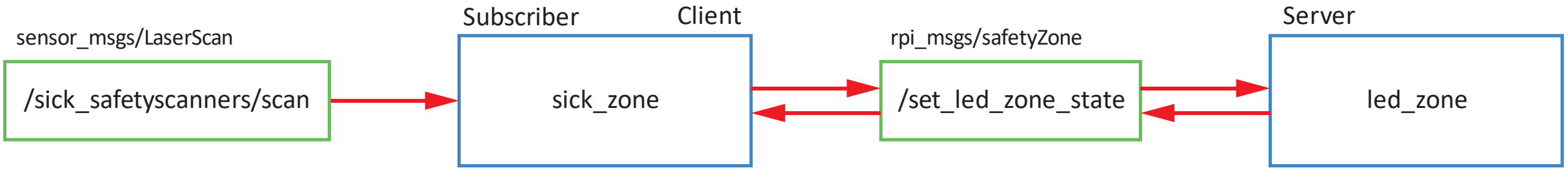
DEBUG MSG/SRV

- `rosmsg list`
- `rosmsg show <ime msg>`

- `rossrv list`
- `rossrv show <ime srv>`

Naloga

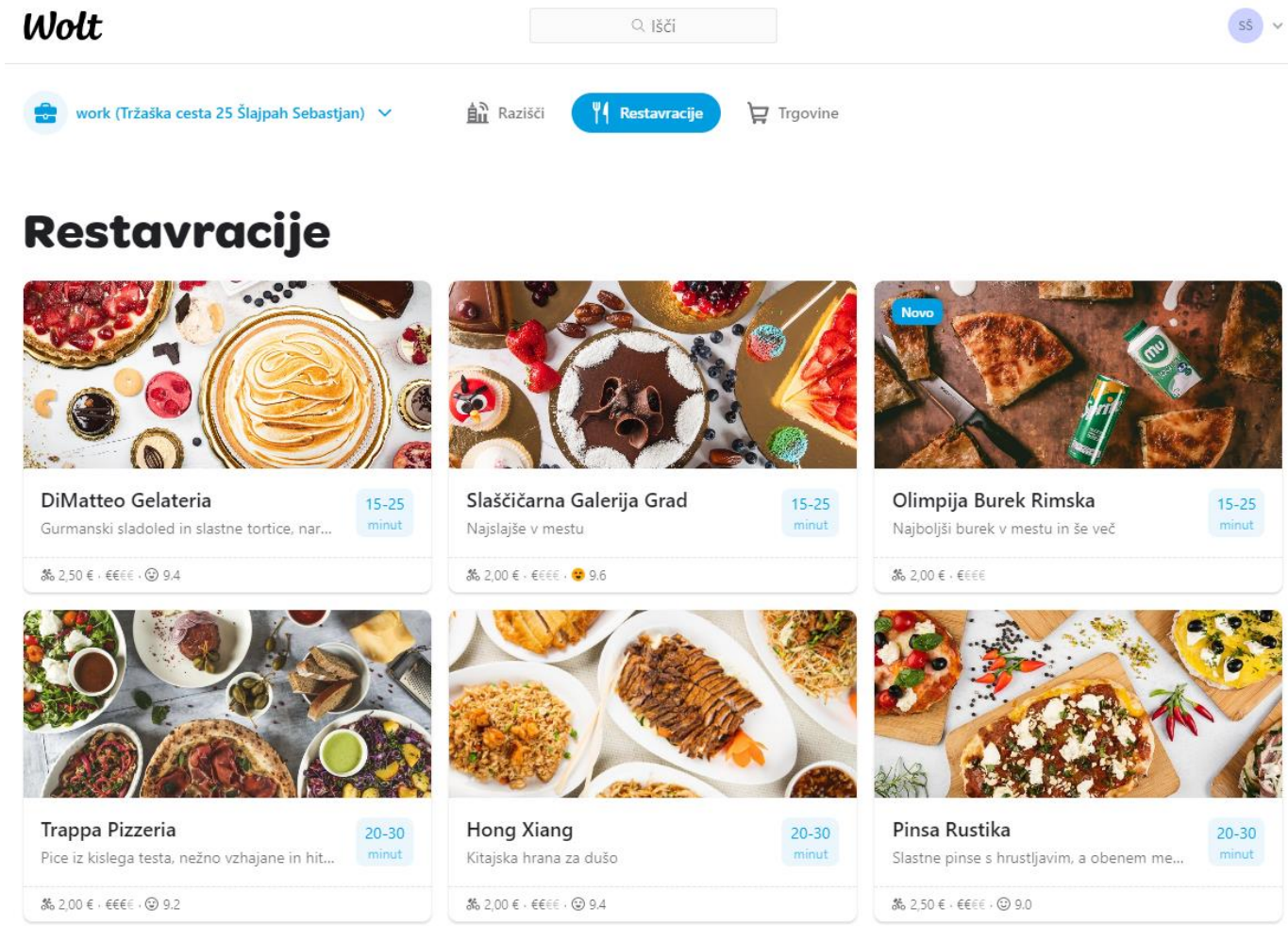
- prižgi LED glede na razdaljo *d*:
  - Zelena: *d* > 0,4 m
  - Rumena: 0,4 m > *d* > 0,2 m
  - Rdeča: *d* < 0,2 m



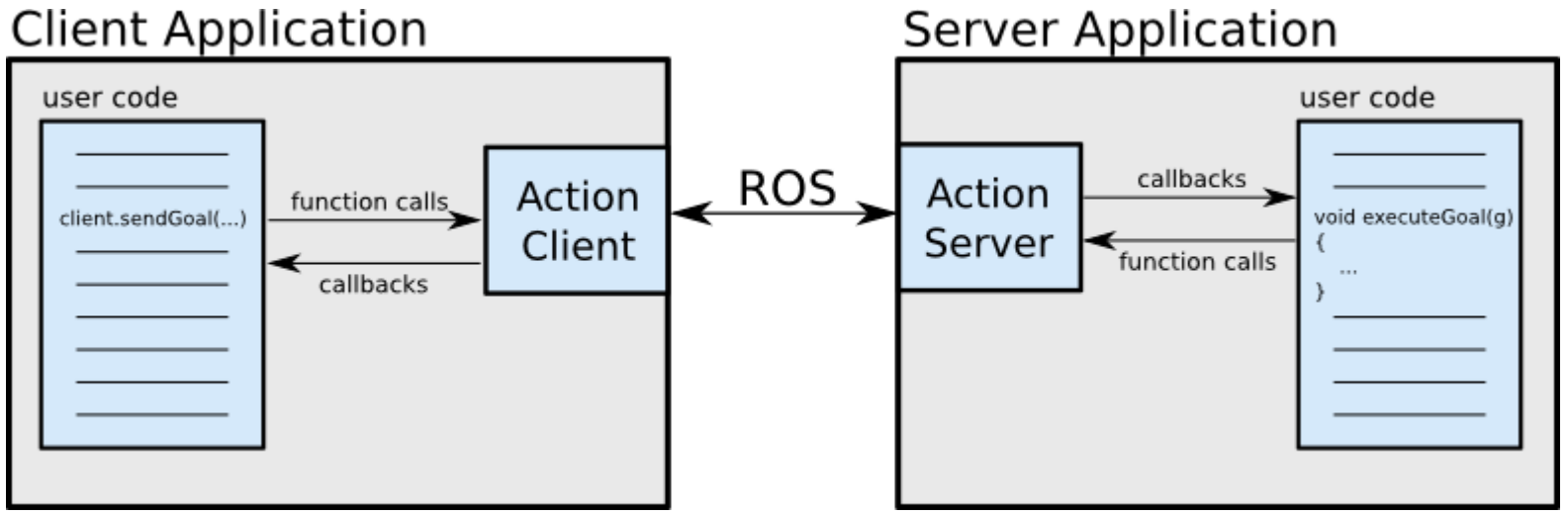
robolab

ACTIONS

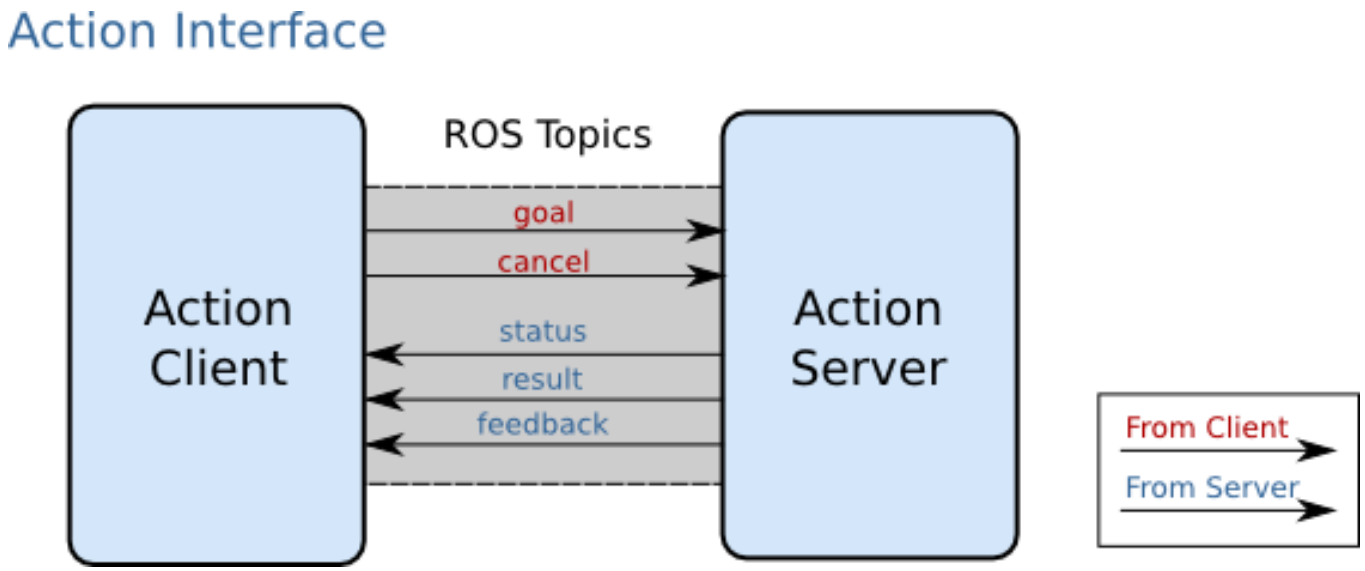
Actions


Wolt
Išči
SŠ
work (Tržaška cesta 25 Šlajpah Sebastjan)
Razišči
Restavracije
Trgovine
Restavracije
DiMatteo Gelateria
Gurmanski sladoled in slastne tortice, nar...
15-25 minut
2,50 € · €€€€ · 9.4
Slaščičarna Galerija Grad
Najslajše v mestu
15-25 minut
2,00 € · €€€€ · 9.6
Novo
Olimpija Burek Rimska
Najboljši burek v mestu in še več
15-25 minut
2,00 € · €€€€
Trappa Pizzeria
Pice iz kislega testa, nežno vzhajane in hit...
20-30 minut
2,00 € · €€€€ · 9.2
Hong Xiang
Kitajska hrana za dušo
20-30 minut
2,00 € · €€€€ · 9.4
Pinsa Rustika
Slastne pinse s hrustljavim, a obenem me...
20-30 minut
2,50 € · €€€€ · 9.0

Actions


Client Application
user code
client.sendGoal(...)
function calls
callbacks
Action Client
ROS
Action Server
Server Application
user code
callbacks
function calls
void executeGoal(g)
{
...
}

Actions


Action Interface
ROS Topics
Action Client
Action Server
goal
cancel
status
result
feedback
From Client
From Server

Actions

- Knjižnica `actionlib`
- Sistem server/klient
- Asinhrono delovanje
- Za funkcionalnosti, ki trajajo dlje časa
- Lahko izvajaš druge naloge, medtem ko je klicana osnovna funkcionalnost
- Posamezna sporočila za Goal, Feedback in Result

Kako prepoznati action?

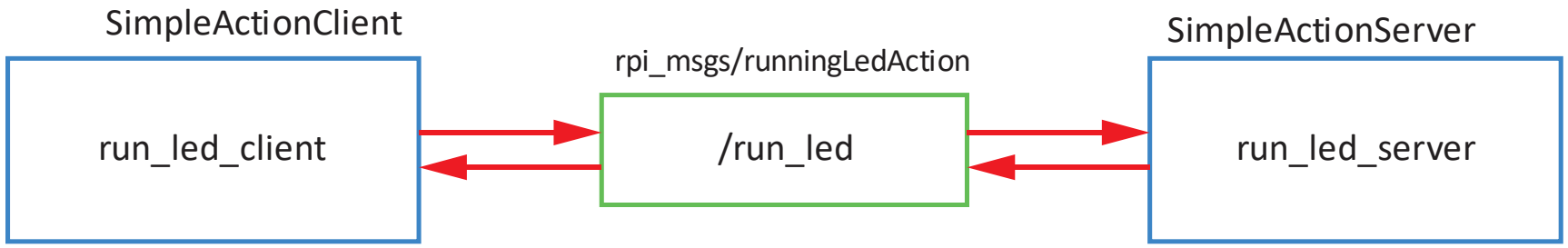
>> rostopic list

Struktura: namespace (`as_name`)

```
as_name/cancel
as_name/feedback
as_name/goal
as_name/result
as_name/status
```

Naloga

- n-kratno izvajanje sekvenčnega prižiganja LED



MSG .action (rpi_msgs)

```
>> mkdir action
>> cd ./action
>> touch runningLed.action
```

```
# goal
int16 numberOfRuns
---
# result
int16 finalRun
---
# feedback
int16 currentRun
```

CMakeLists.txt

```
find_package(
       actionlib_msgs
)


add_action_files(
       FILES
       NAME.action
)
```

```
generate_messages(
       DEPENDENCIES
       actionlib_msgs
)


catkin_package(
       CATKIN_DEPENDS rospy
)
```

package.xml

```
<build_depend>actionlib_msgs</build_depend>
```

SimpleActionServer

```
sas = actionlib.SimpleActionServer('name', actionSpec, goal_callback,
        auto_start=False)
sas.start()


def goal_callback()
        sas.publish_feedback(_feedback_)
        sas.set_succeeded(_result_)

        if sas.is_preempt_requested():
                sas.set_preempted()
```

SimpleActionServer

```
import actionlib
from rpi_msgs.msg import  runningLedAction, runningLedFeedback, runningLedResult,

ACserver = None
def goal_callback(goal):
    pass

if __name__ == '__main__':
    rospy.init_node('run_led_server')
    ACserver = actionlib.SimpleActionServer('run_led', runningLedAction, goal_callback, False)
    GPIO.setmode(GPIO.BCM)

    # start server
    ACserver.start()
    rospy.spin()
    GPIO.cleanup()
```

SimpleActionServer

```
def goal_callback(goal):
    runFeedback = runningLedFeedback()
    runResult = runningLedResult()

    # Do lots of awesome groundbreaking robot stuff here
    for ii in range(1,goal.numberofRuns+1):
        #
        # prizgi ustrezno LED
        #
        runFeedback.currentRun = ii
        ACserver.publish_feedback(runFeedback)

    # publish the result
    runResult.finalRuns = runFeedback.currentRun
    ACserver.set_succeeded(runResult)
```

SimpleActionClient

```
client = actionlib.SimpleActionClient('name', actionSpec)

client.send_goal(goal)          # Sends the goal to the action server.

client.wait_for_result()        # Waits for the server to finish performing the action.
client.get_result()             # Prints out the result of executing the action


client.get_state()              # Get current state of the server


# define action server status
PENDING = 0
ACTIVE = 1
DONE = 2
WARN = 3
ERROR = 4
```

SimpleActionClient

```
import actionlib
from rpi_msgs.msg import runningLedAction, runningLedGoal

client = None

def run_led_client(goalNum):
    pass

if __name__ == '__main__':
    rospy.init_node('run_led_client')
    GPIO.setmode(GPIO.BCM)

    try:
        result = run_led_client(goalNum = 10)
    except rospy.ROSInterruptException:
        print("Program interrupted before completion")
```

SimpleActionClient

```
def run_led_client(goalNum):
    global client
    client = actionlib.SimpleActionClient('run_led', runningLedAction)
    client.wait_for_server()       # Waits until the action server has started up and started
                                     listening for goals.
    goal = runningLedGoal()        # Creates a goal to send to the action server.
    goal.numberOfRuns = goalNum

    client.send_goal(goal)         # Sends the goal to the action server.
    client.wait_for_result()       # Waits for the server to finish performing the action.
    return client.get_result()     # Prints out the result of executing the action
```

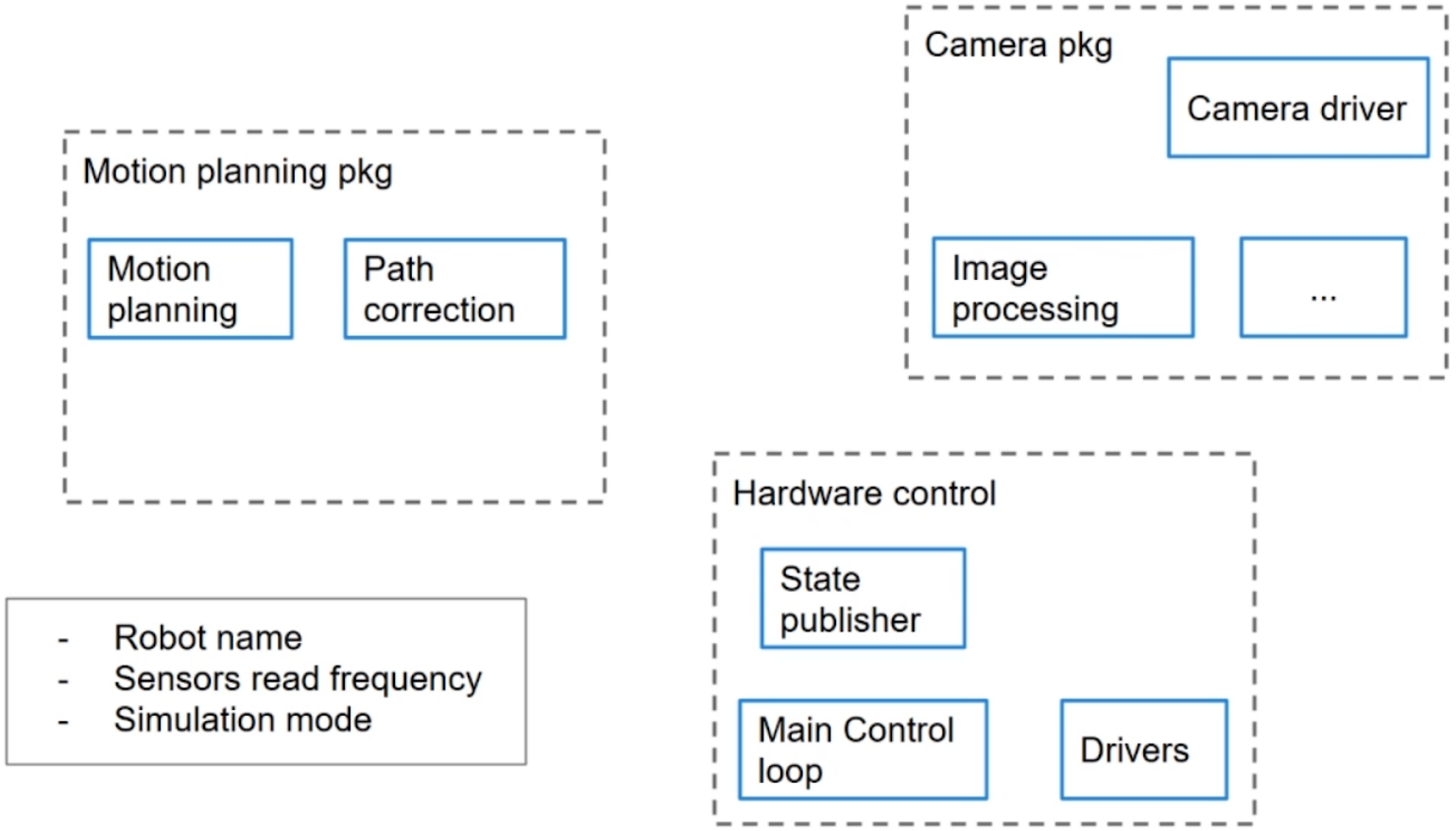
SimpleActionClient

```
def run_led_client(goalNum):
    #... client defininiton ...
    client.send_goal(goal)    # Sends the goal to the action server.

    # let us do some other stuff
    current_state = client.get_state()

    while current_state < DONE:
        # action is still active, let us do something
        current_state = client.get_state()
        rate.sleep()

    if current_state == WARN:
        rospy.logwarn("[Warn] Warning on the action server side.")

    if current_state == ERROR:
        rospy.logerr("[Error] Error on the action server side.")

    return  client.get_result()
```

robolab

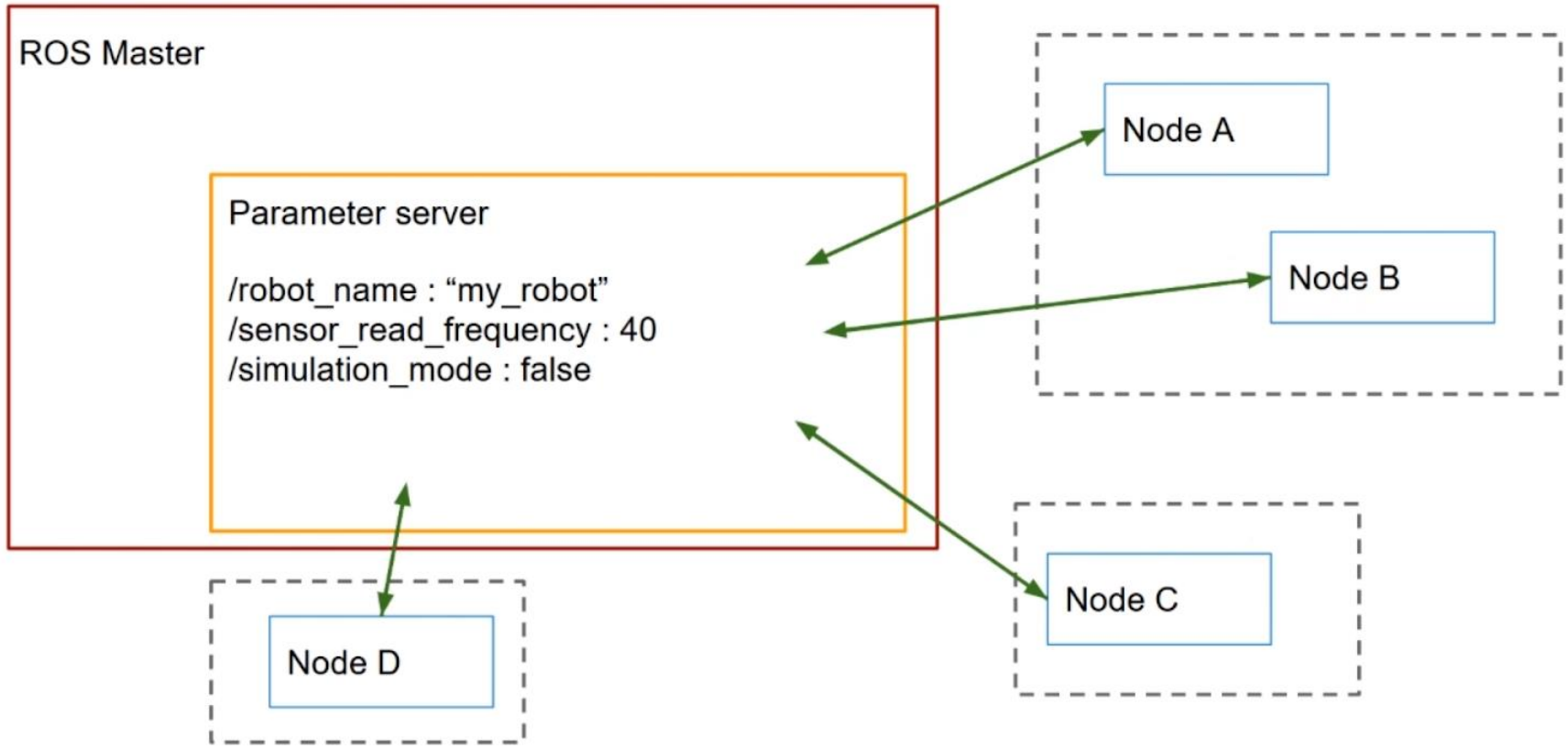
PARAMETERS and LAUNCH FILES

Parameters


Motion planning pkg
Motion planning
Path correction
Camera pkg
Camera driver
Image processing
...
- Robot name
- Sensors read frequency
- Simulation mode
Hardware control
State publisher
Main Control loop
Drivers

robolab

Parameters



Parameters

- Parameter server: slovar znotraj ROS master, globalno dosegljiv
- ROS parameter: ena spremenljivka znotraj parameter serverja
- Tipi:
  - Boolean
  - Int
  - Double
  - String
  - Lists
  - …

Parameter

```
>> rosparam set <param name> <value>
>> rosparam get <param name>
>> rosparam list
```

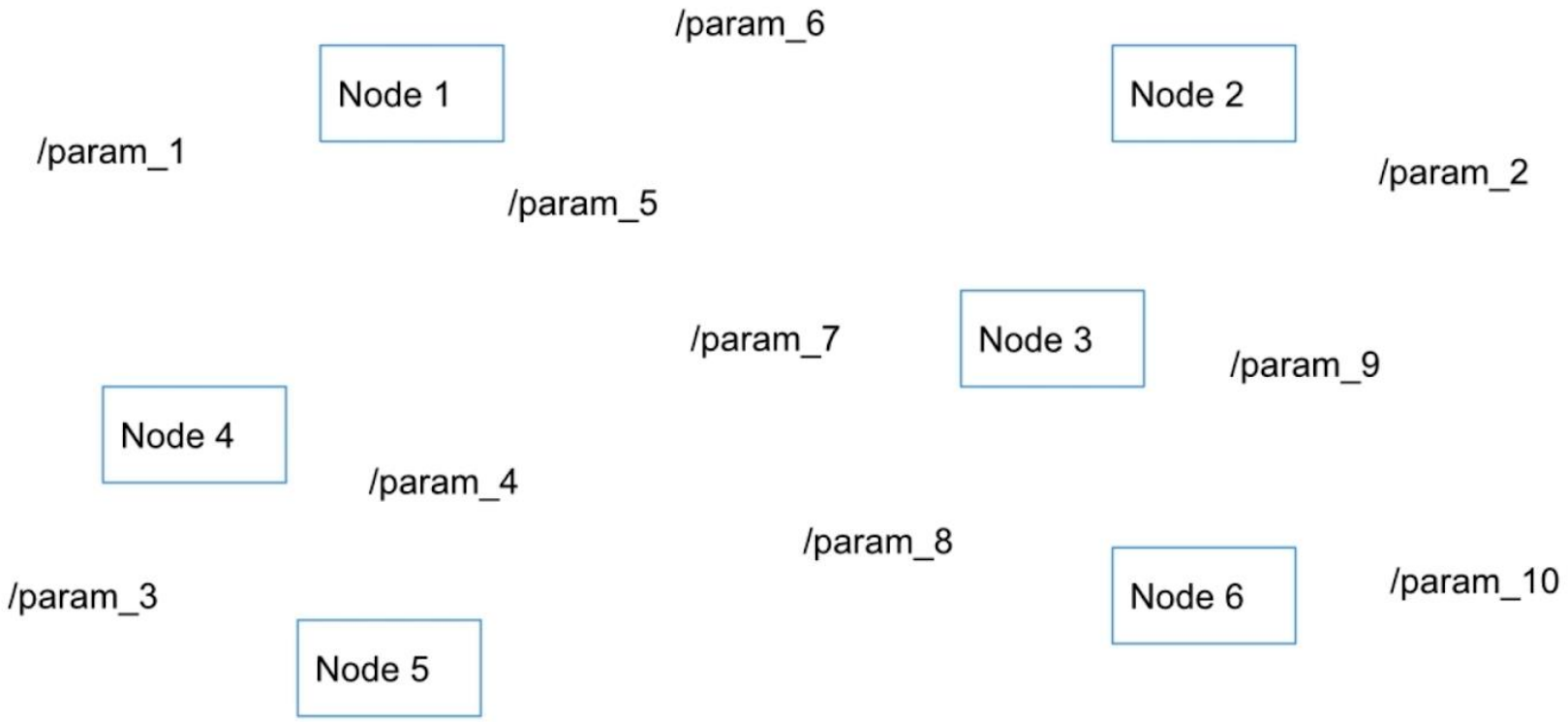
… tako ga tudi ustvariš

- Primer:

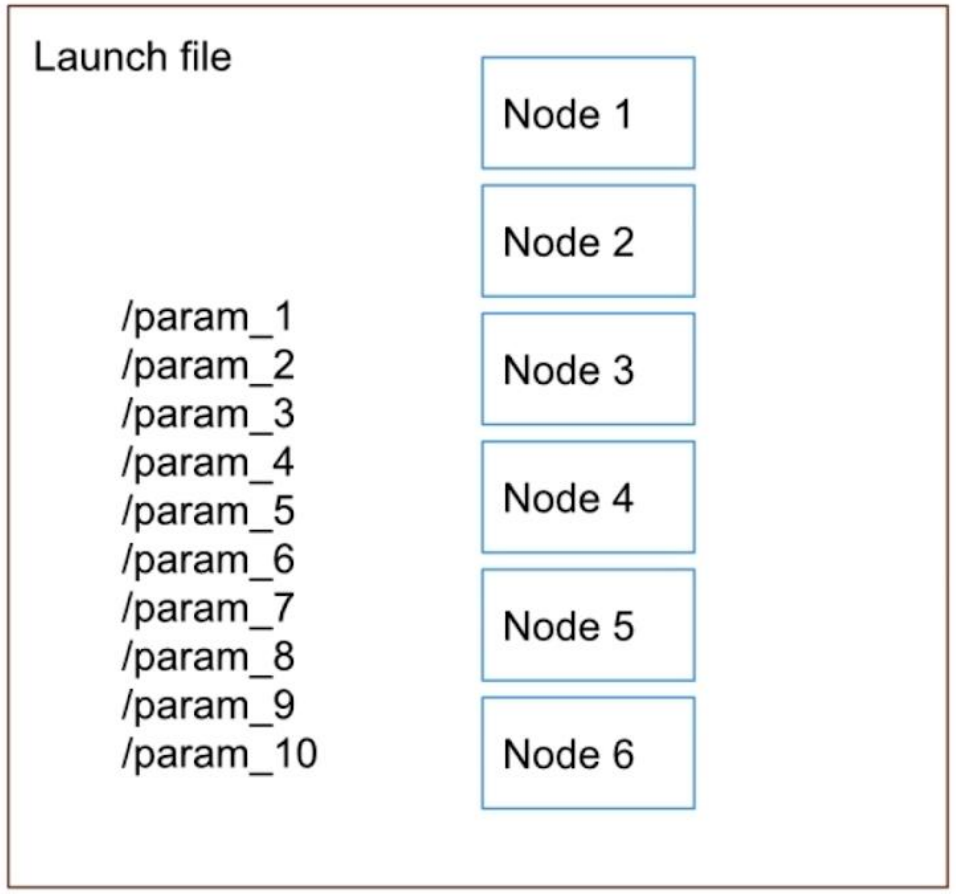
```
publish_freq = rospy.get_param('/number_publish_freq')
```

```
>> rosparam set /number_publish_freq 2
```

Launch file


/param_6
Node 1
Node 2
/param_1
/param_2
/param_5
/param_7
Node 3
/param_9
Node 4
/param_4
/param_8
/param_3
Node 6
/param_10
Node 5

Launch file


Launch file
Node 1
Node 2
/param_1
/param_2
/param_3
/param_4
/param_5
/param_6
/param_7
/param_8
/param_9
/param_10
Node 3
Node 4
Node 5
Node 6

Nov .launch

```
>> catkin_create_pkg rpi_feros_bringup
>> catkin_make
>> mkdir launch
>> touch feros.launch
```

```
<launch>
   <param name="/ime_parametra" type="tip_spremenljivke" value="vrednost"/>
   <node name="ime" pkg="paket" type="source_file.py" />
</launch>
```

```
>> roslaunch rpi_feros_bringup feros.launch
```

Naloga

- Nadgradite SimpleActionClient s parametrom
  - Število sekvenc: `/number_of_runs`
- Nadgradite SimpleActionServer s parametrom
  - Hitrost izvajanja sekvence: `/led_frequency`

- Naredite `.launch` datoteko za Action Server