

KDR ROS orodja

Sebastjan Šlajpah

Univerza v Ljubljani
Fakulteta za elektrotehniko
Laboratorij za robotiko

sebastjan.slajpah@fe.uni-lj.si

Vsebina

- Nodes
- Topics
- Services
- Msg & Srv files
- Actions
- Parameter
- Launch files
- Konzolni ukazi za posamezne funkcionalnosti
- Python 3.8

Literatura

- <https://github.com/sslajpah/>
- https://github.com/sslajpah/kdr_ros
- <https://sslajpah.github.io/ros-docs/>

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

```
$ cd  
$ 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
```

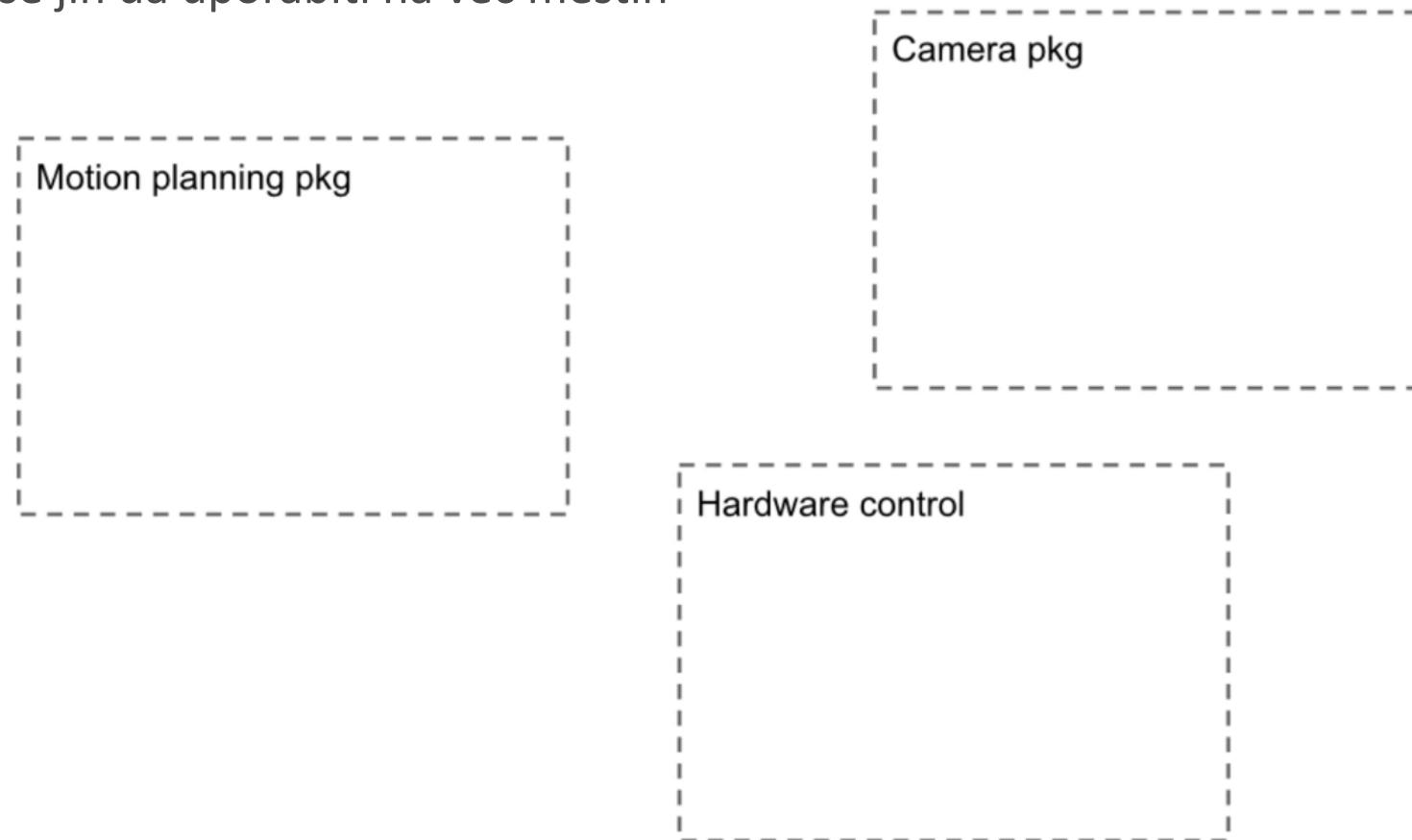
PACKAGES

Packages

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

Packages

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



Nov paket

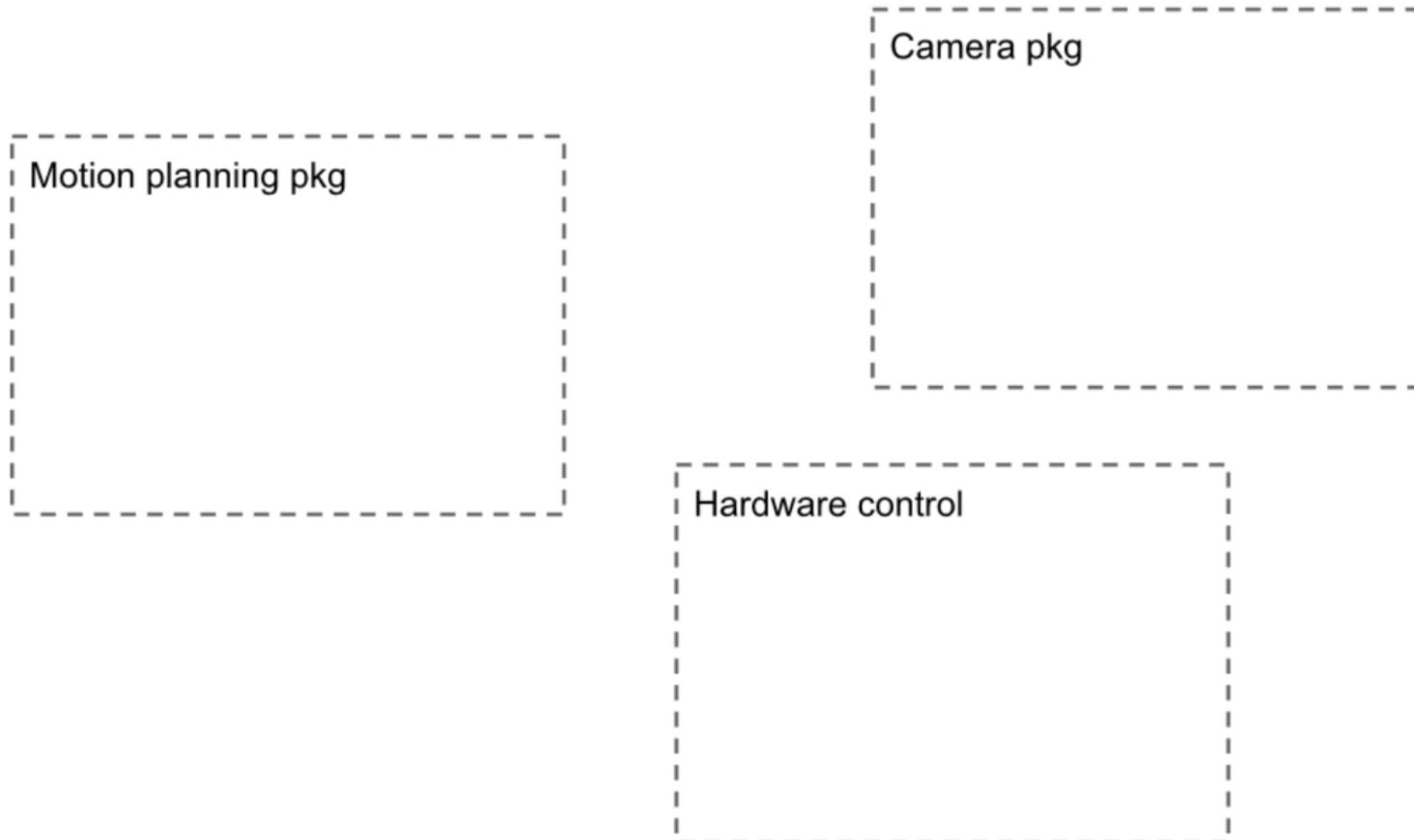
```
$ catkin_create_pkg <ime_paketa> <razširitve>
```

```
$ catkin_make
```

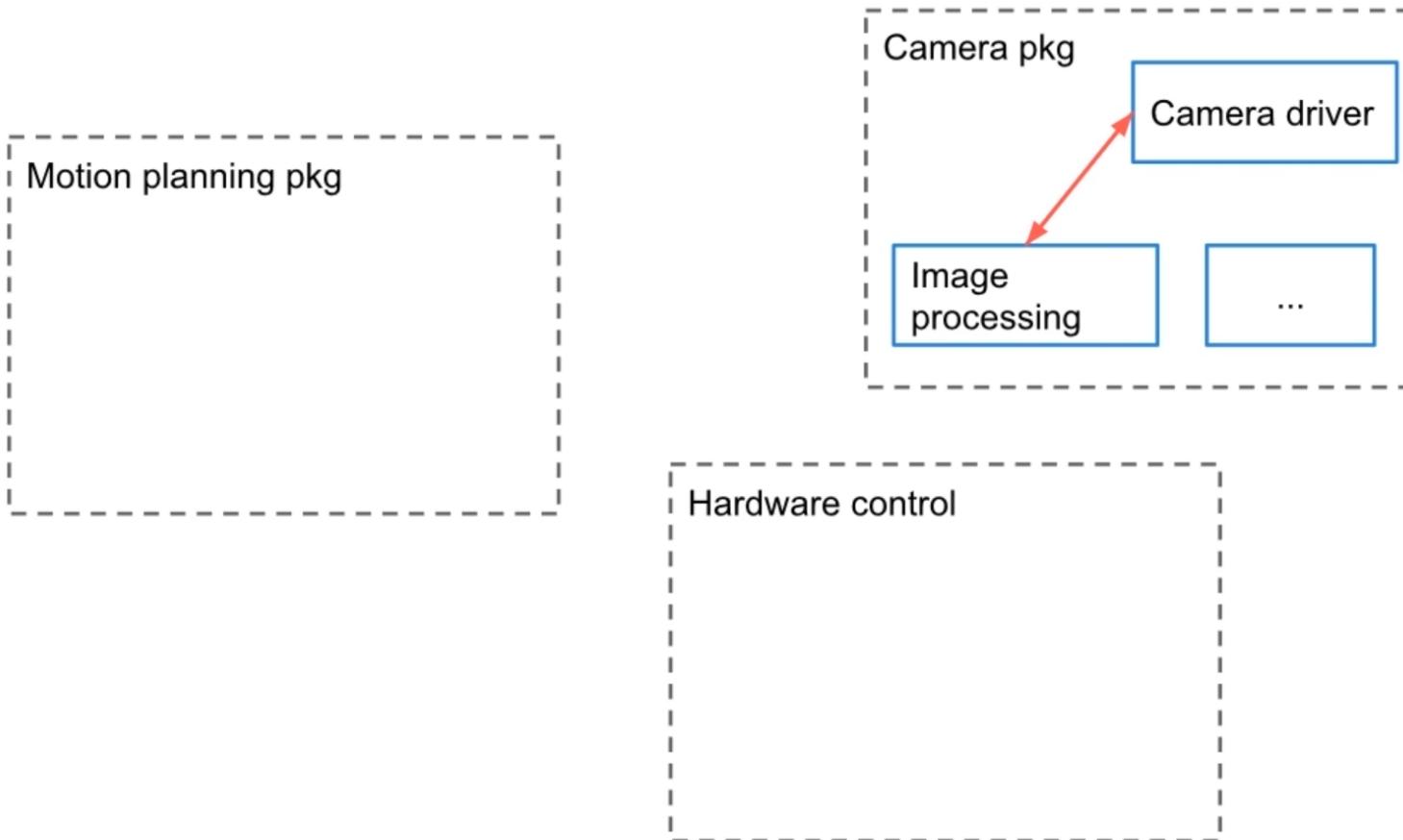
```
$ catkin_create_pkg kdr rospy std_msgs
```

NODE

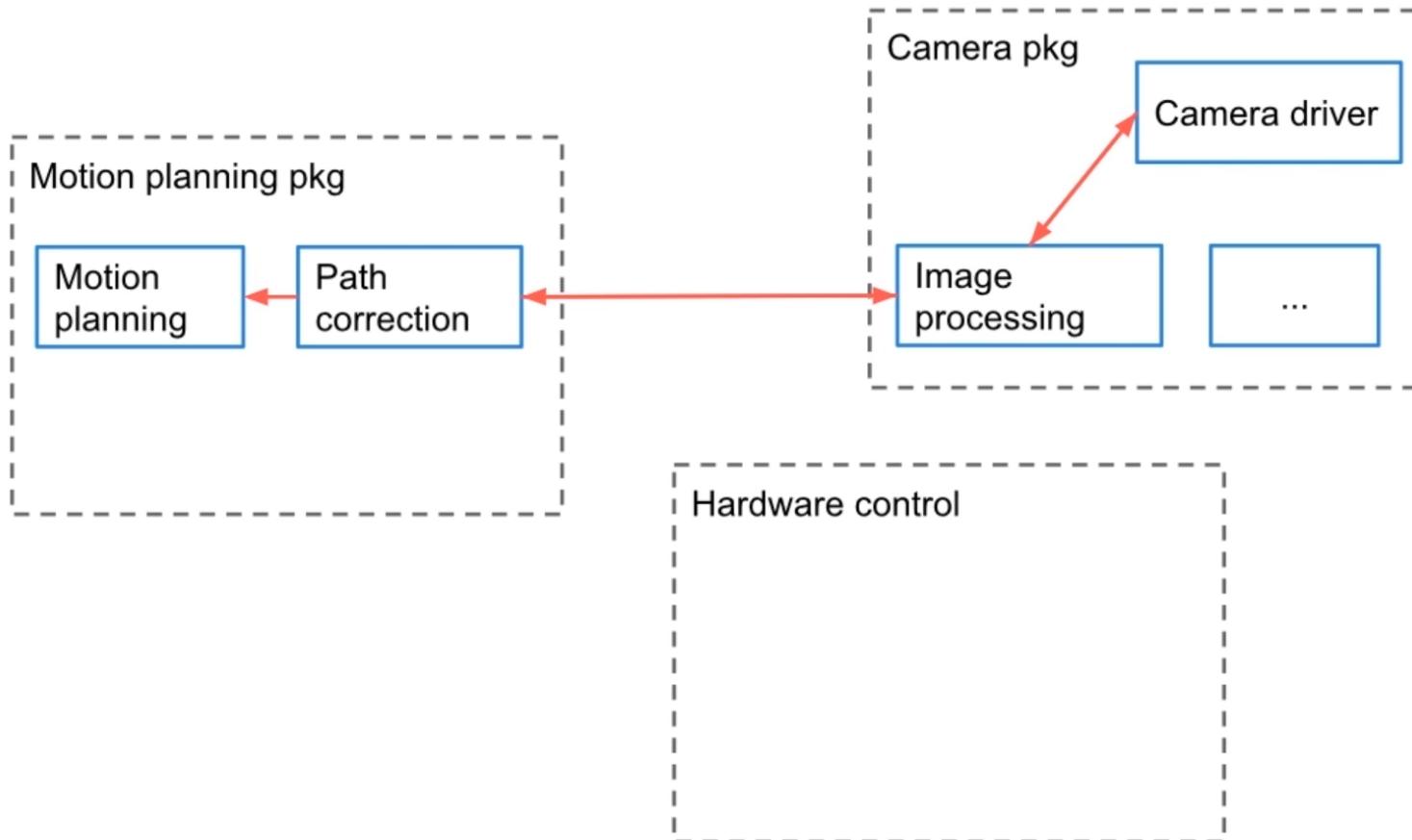
Node



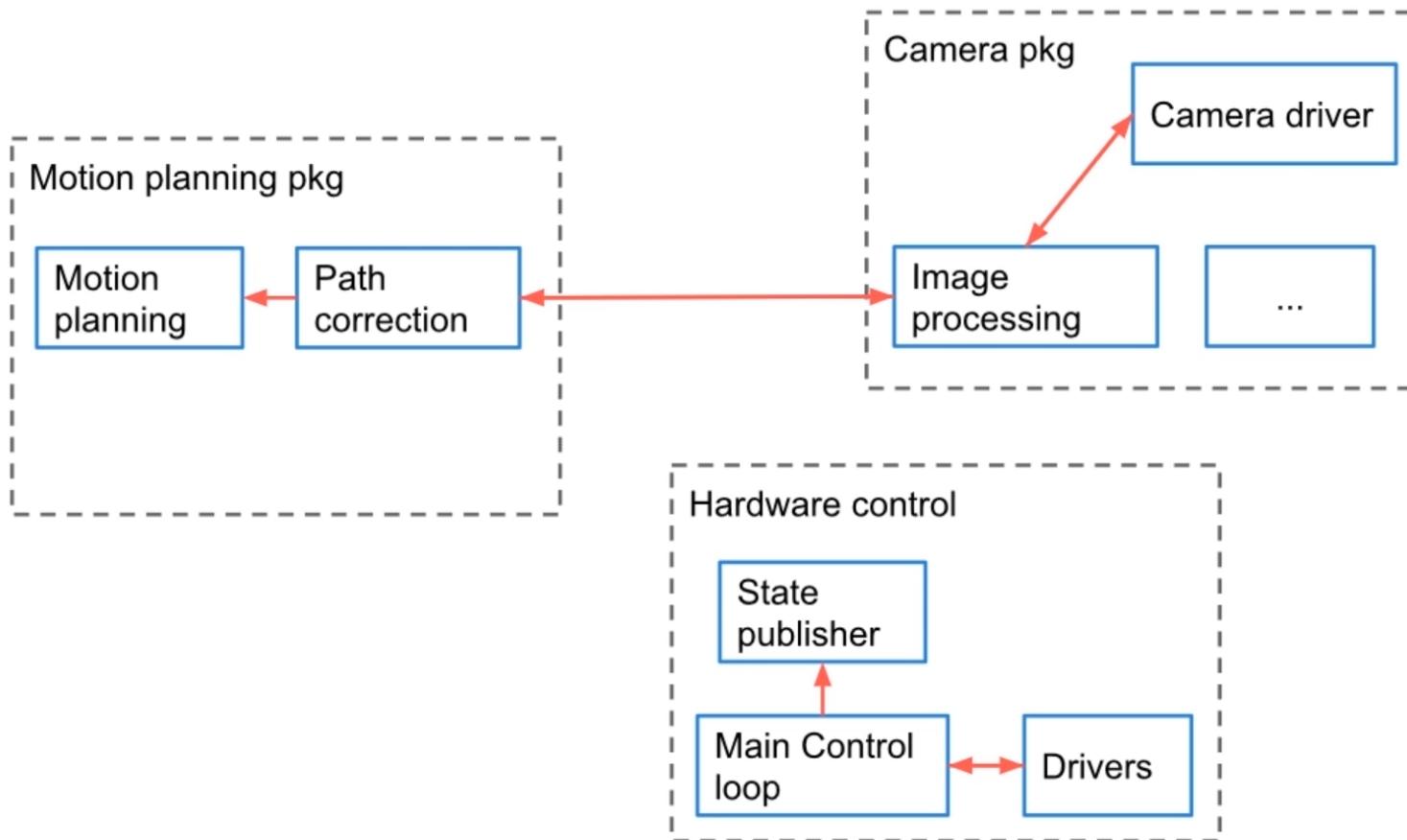
Node



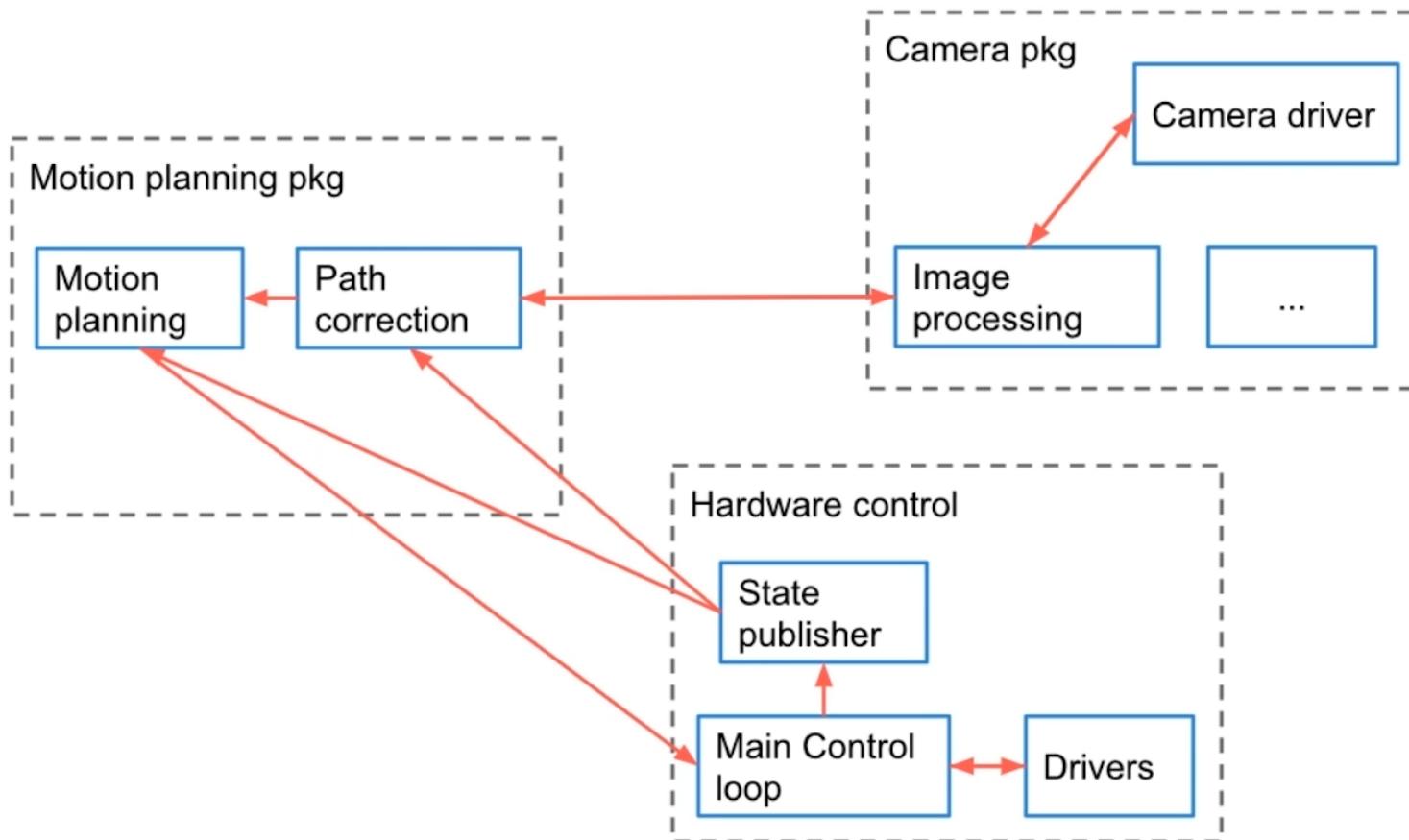
Node



Node



Node



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

Nov Node

```
$ rosdep init  
$ rosdep update  
$ roscd kdr  
$ mkdir scripts  
$ cd scripts  
$ touch my_first_node.py  
$ chmod +x my_first_node.py  
$ code my_first_node.py
```

```
#!/usr/bin/env python3

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')
```

```
$ python3 my_first_node.py
```

DEBUG Node

- **rosrun <pkg name> <node name>** ... poganjanje
- **rosnodes list** ... seznam vseh aktivnih
- **rosnodes info <node name>** ... info o node
- **rosnodes kill <node name>** ... ugasni node
- **rosnodes ping <node name>** ... ping (preveri, če deluje)

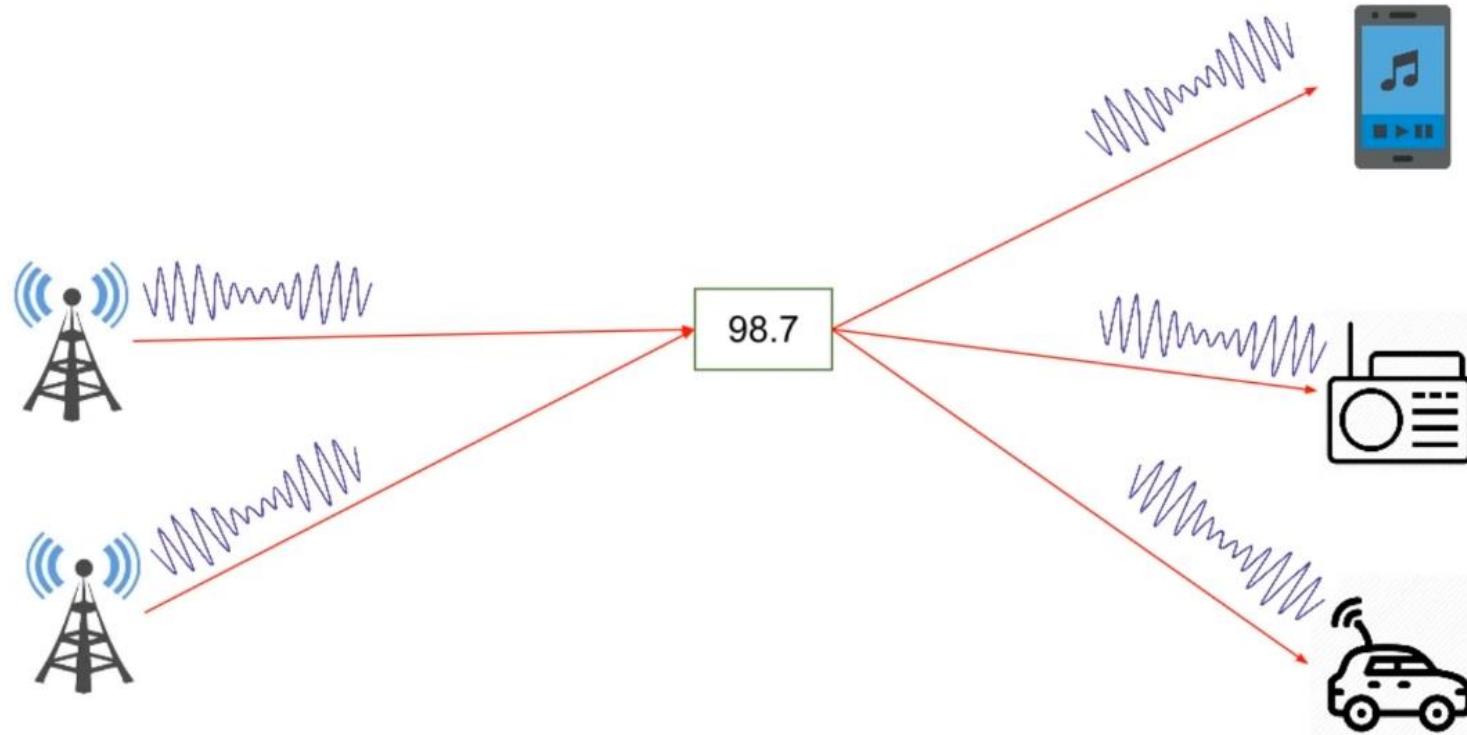
Node

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

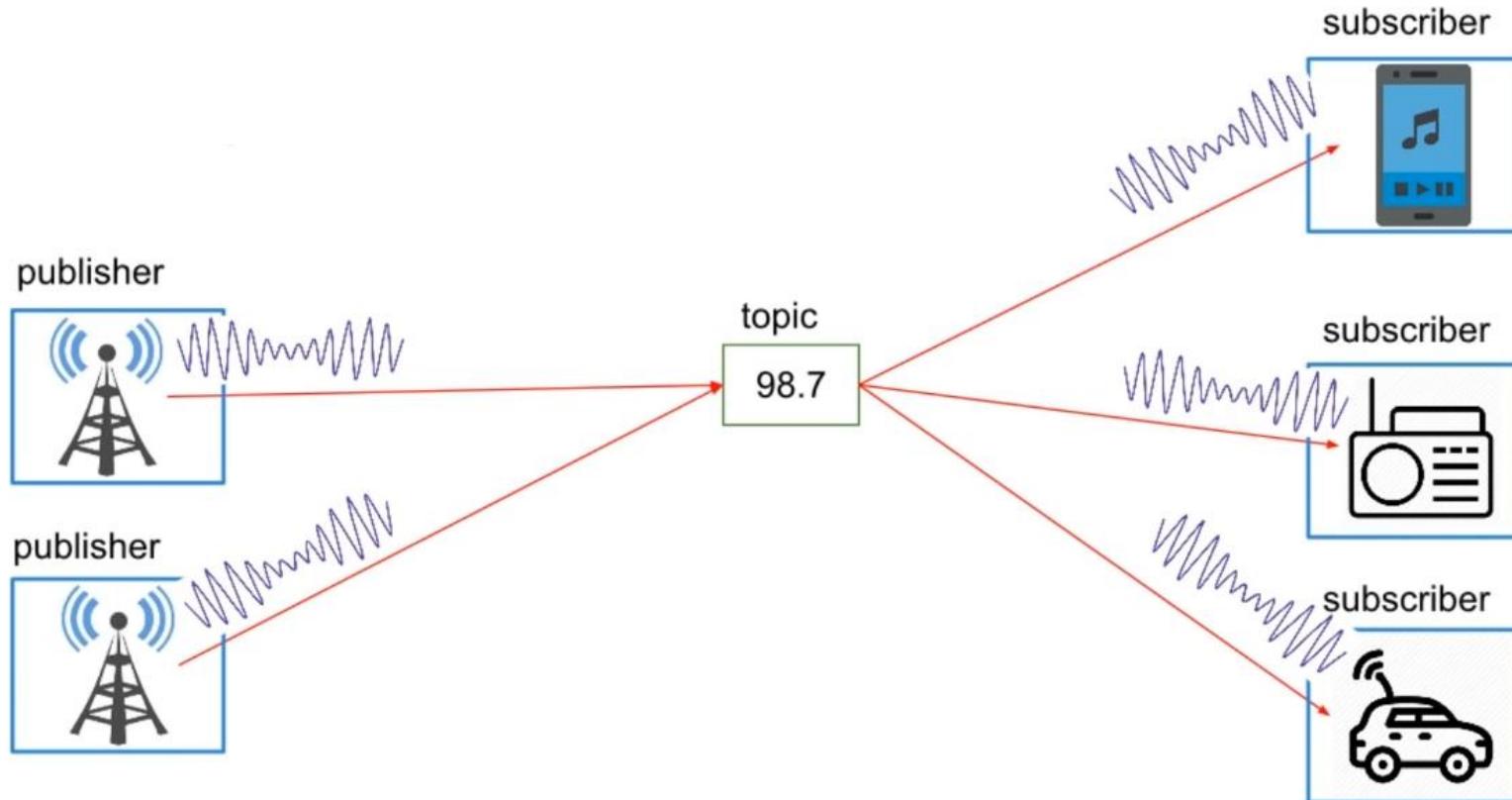
```
rospy.init_node('my_first_python_node', anonymous=True)
```

TOPICS

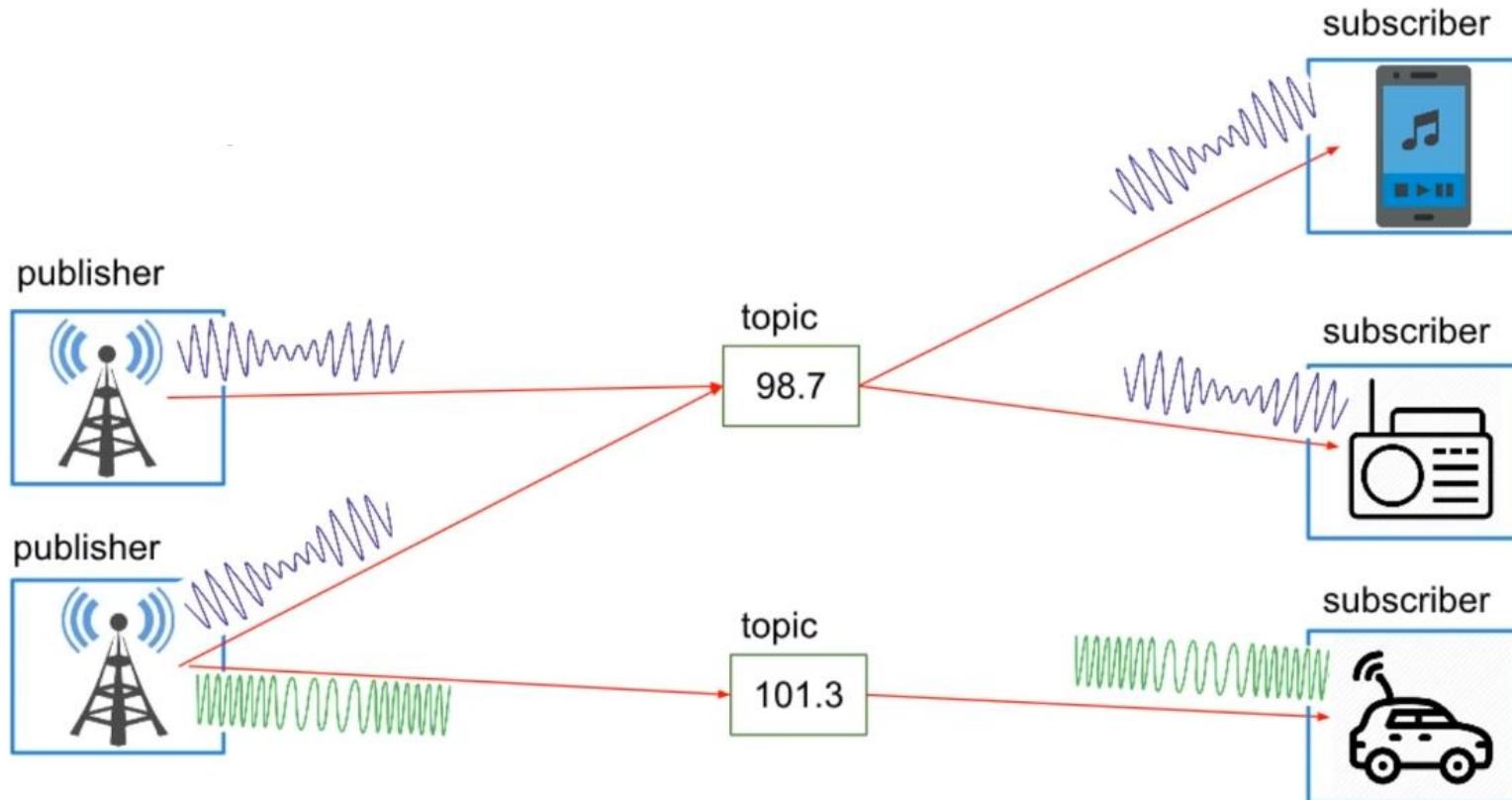
Topic



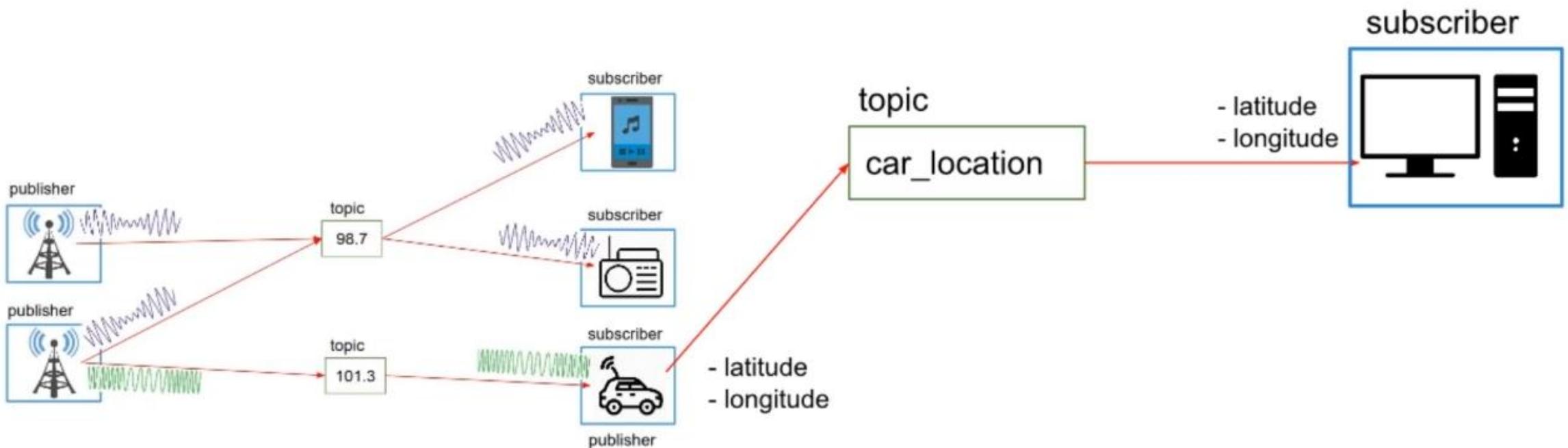
Topic



Topic



Topic



Topic

- Vodilo, preko katerega si nodi izmenjujejo sporočila
- Enosmerni prenos sporočil (publisher > subscriber)
- Anonimno
- Topic ima svoj tip sporočila
- ROS master skrbi za ustrezen 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

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

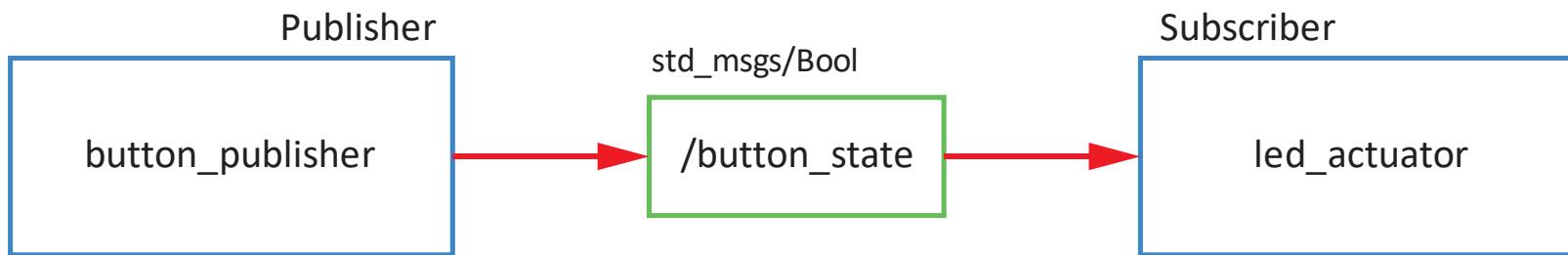
```
sub = rospy.Subscriber('topic_name', msg_type, callback_fcn)
```

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
 - `-r5` ... Pošilja s 5 Hz

Primer

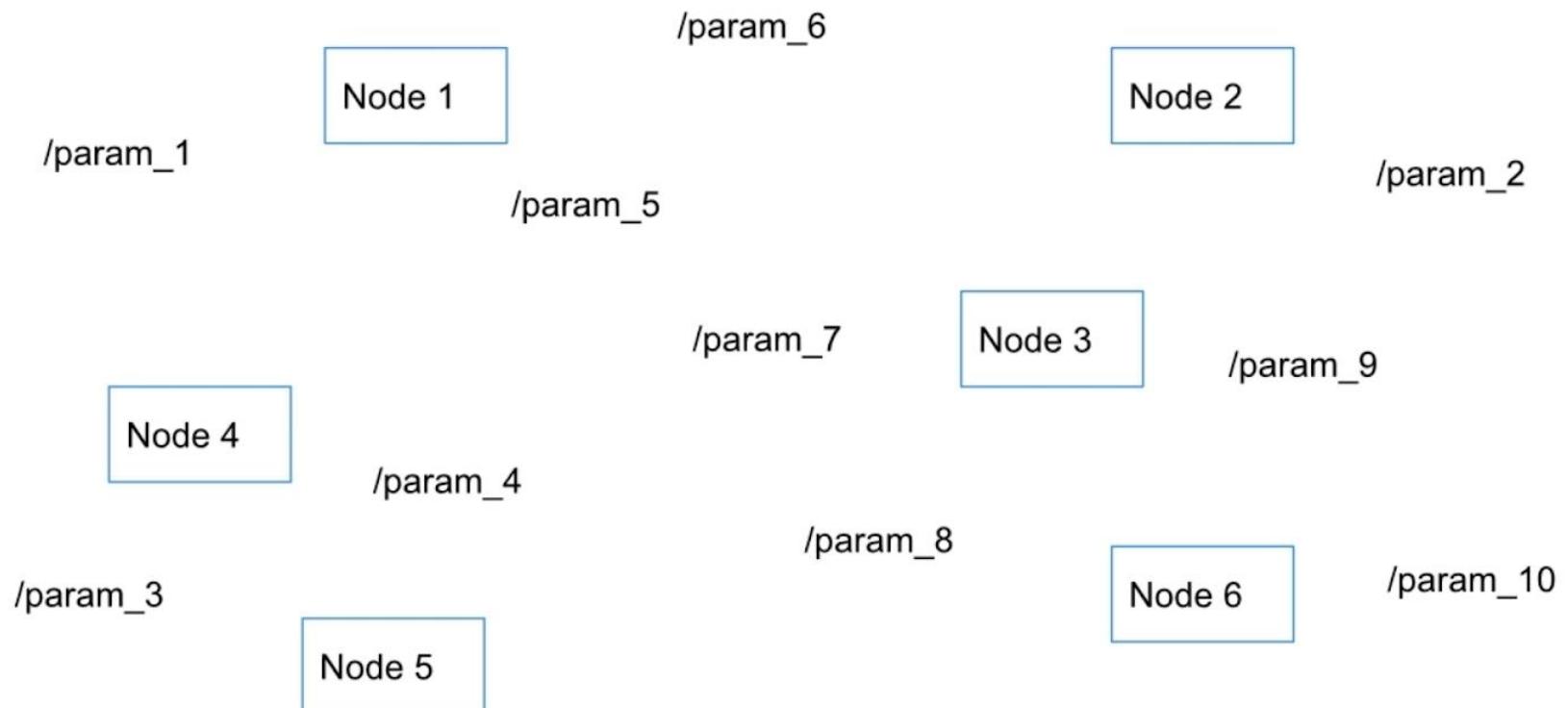
- ob pritisku tipke prižgi LED
- **button_publisher.py**
- **led_actuator.py**



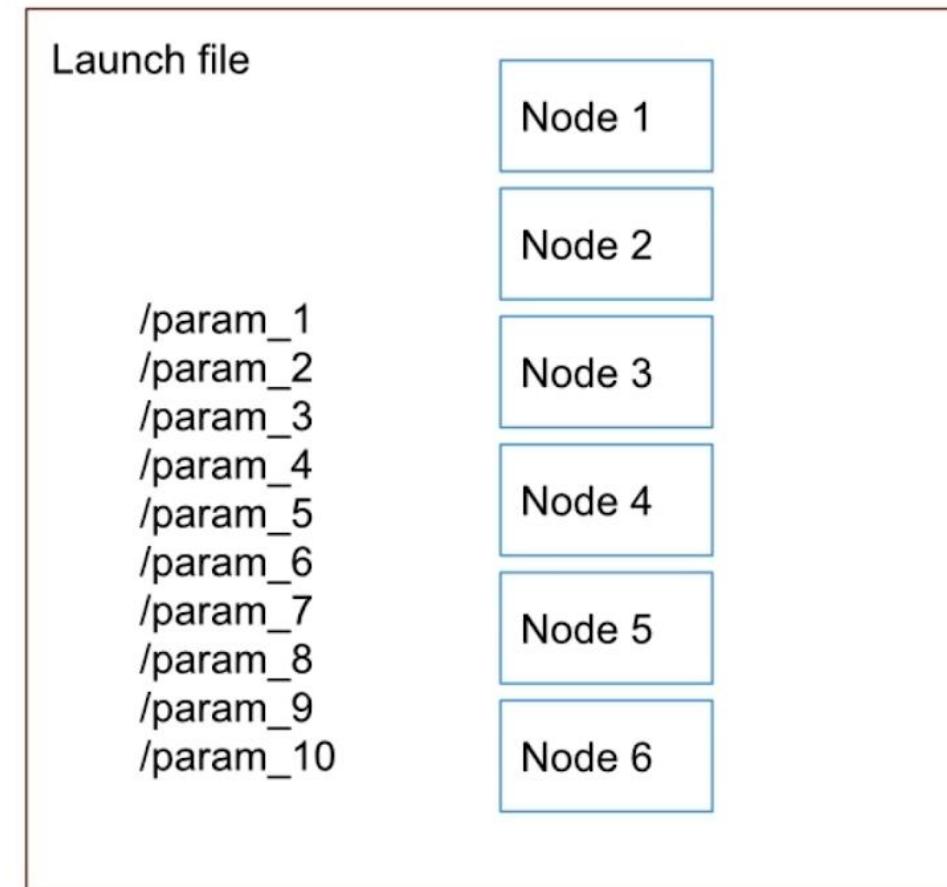


LAUNCH FILES

Launch file



Launch file



Nov .launch

```
$ roscd  
$ cd ..  
$ cd src  
$ catkin_create_pkg kdr_bringup  
$ cd ..  
$ catkin_make
```

Nov .launch

```
$ rosdep install kdr_bringup
$ mkdir launch
$ touch rpi_led.launch
```

```
<launch>
  <param name="/ime_parametra" type="tip_spremenljivke" value="vrednost"/>
  <node name="ime" pkg="paket" type="source_file.py" />
  <include file="included.launch">
    <arg name="arg_name" value="arg_val" />
  </include>
</launch>
```

```
$ roslaunch rpi_feros_bringup rpi_led.launch
```

ROS OMREŽJE

ROS OMREŽJE

- en ROS master v celotnem omrežju
- vsi nodi morajo uporabljati isti ROS master (ROS_MASTER_URI)
- popolna dvosmerna povezava med napravami
- vsaka naprava se mora predstaviti z imenom, ki ga ostale naprave prepoznajo

<http://wiki.ros.org/ROS/Tutorials/MultipleMachines>

<http://wiki.ros.org/ROS/NetworkSetup>

PING

Ping hostname:

```
$ ping lr-legs
```

Ping IP:

```
$ ping 192.168.65.110
```

TEŽAVE

Remapping:

```
$ sudo nano /etc/hosts
```

Dodate ustrezne pare IP - hostname

192.168.65.110	lr-legs
----------------	---------

TEŽAVE

Ubuntu firewall

```
$ sudo ufw status
```

```
$ sudo ufw disable
```

```
$ sudo ufw enable
```

ROS_MASTER_URI

Določi, kje je ROS master

- potrebno zagnati v vsaki konzoli

```
$ export ROS_MASTER_URI=http://[ime_master_racunalnika]:11311
```

```
$ export ROS_MASTER_URI=http://lr-legs:11311
```

Preverite nastavitev:

```
$ echo $ROS_MASTER_URI
```

ROS_MASTER_URI

Avtomatska postavitev ROS_MASTER_URI ob odprtju konzole:

```
$ sudo nano ~/.bashrc
```

Doda se vrstico:

```
export ROS_MASTER_URI=http://lr-legs:11311
```

POZOR!

Možne težave, če boste poganjali ROS master na lokalnem računalniku!

TEST

Potrebno preveriti povezave!

Master (publisher) >>> ostali (subscriber)
Master (subscriber) <<< ostali (publisher)

Publisher:

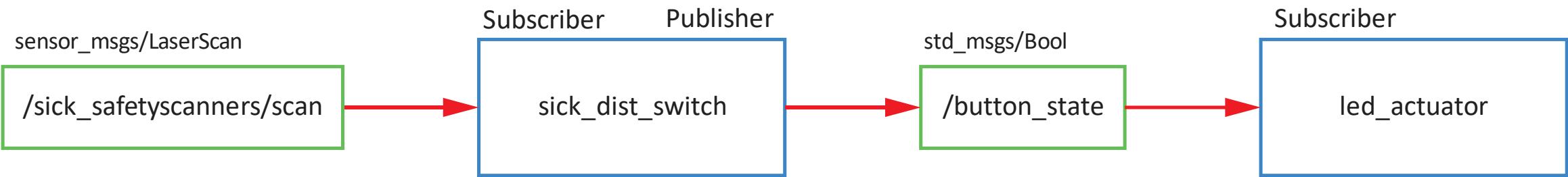
```
$ rostopic pub -r5 /test_connection std_msgs/Bool "data: True"
```

Subscriber:

```
$ rostopic list  
$ rostopic echo /test_connection
```

Naloga

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



SICK NanoScan3 – namestitev

```
$ sudo apt-get install ros-noetic-sick-safetyscanners  
  
$ source /opt/ros/noetic/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

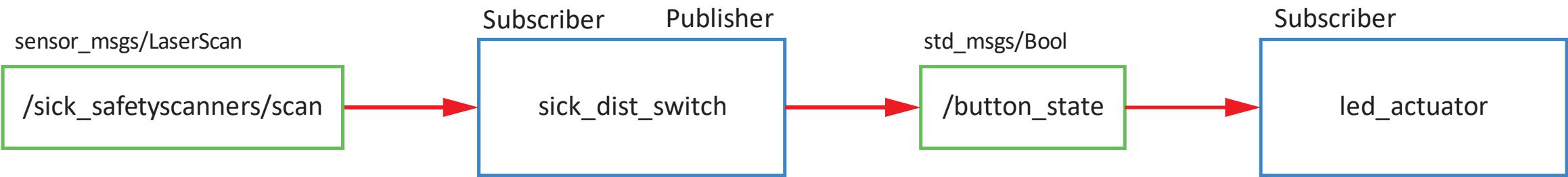
- 1651 meritev
 - Korak meritev: 0,002909 rad ($0,1667^\circ$)
 - Kot skeniranja: 275°
-
- Topic: /sick_safetyscanners/scan
 - Msg: from sensor_msgs.msg import LaserScan

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



Naloga

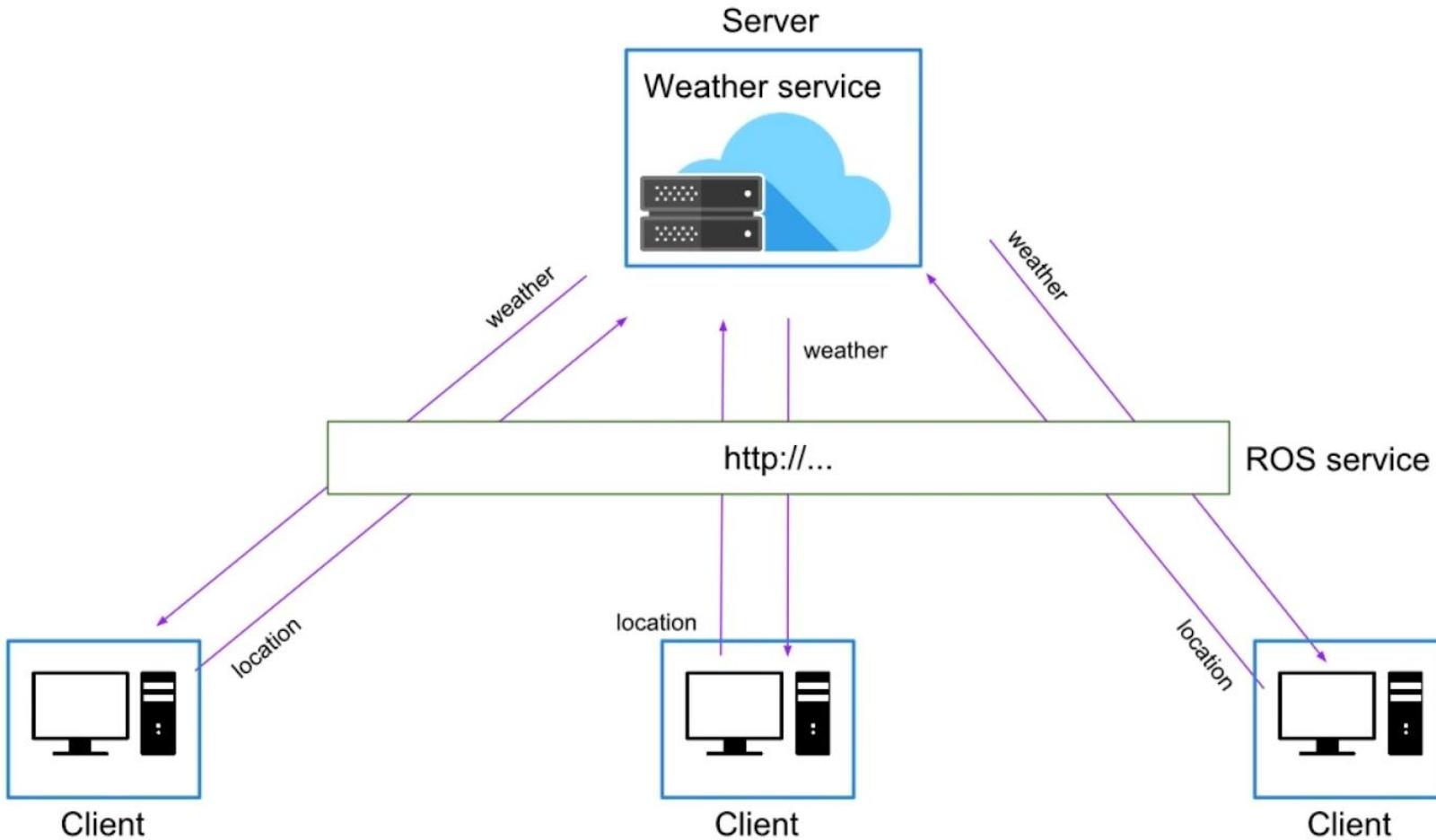
- prižgi LED, ko je objekt bližje kot 0,2 m



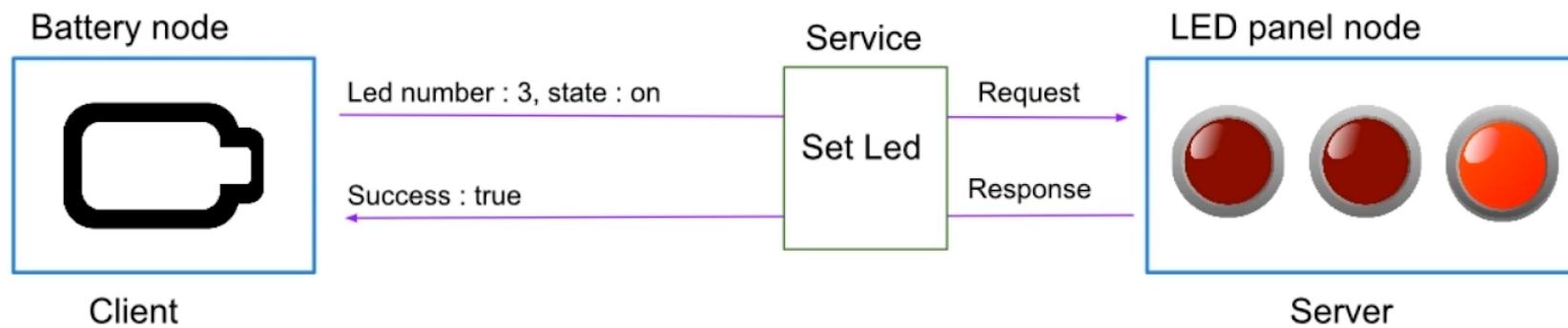


SERVICES

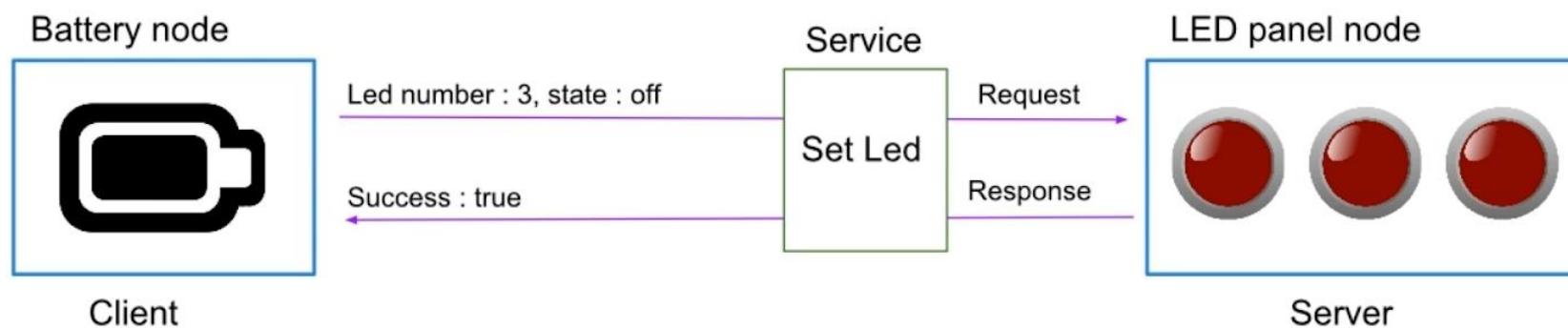
Services



Services



Services



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')

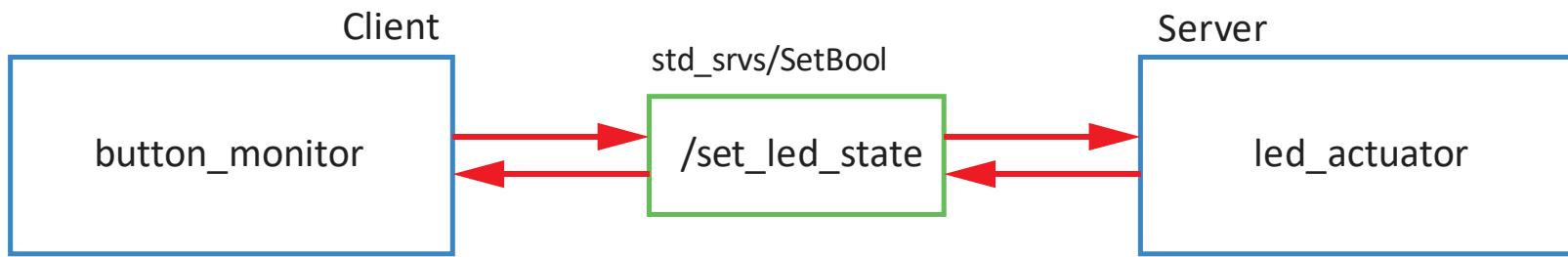
try:
    client = rospy.ServiceProxy('ime_service',msg_type)
    ...
except rospy.ServiceException as e:
    rospy.logwarn('Service failed' + str(e))
```

DEBUG Services

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

Primer

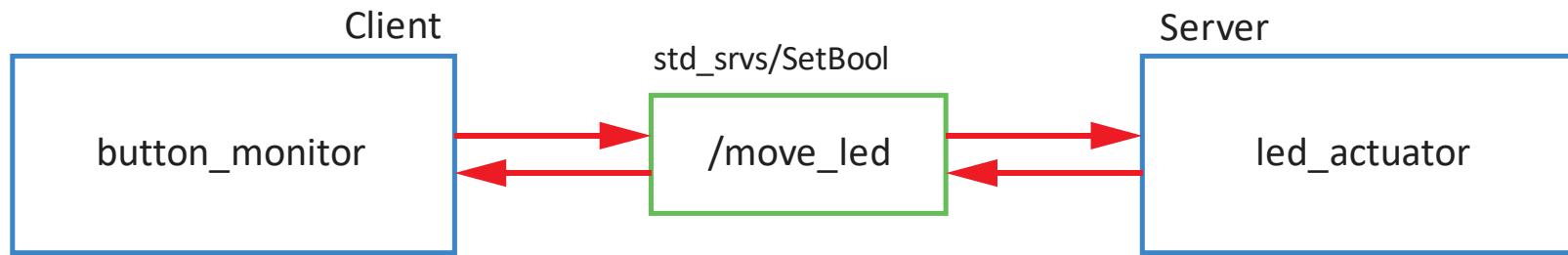
- prižgi/ugasni LED ob pritisku tipke (prekinitve)
- **led_service_server.py**
- **button_service_client.py**



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

Naloga

- s tipkami premikajte prižgano LED levo/desno



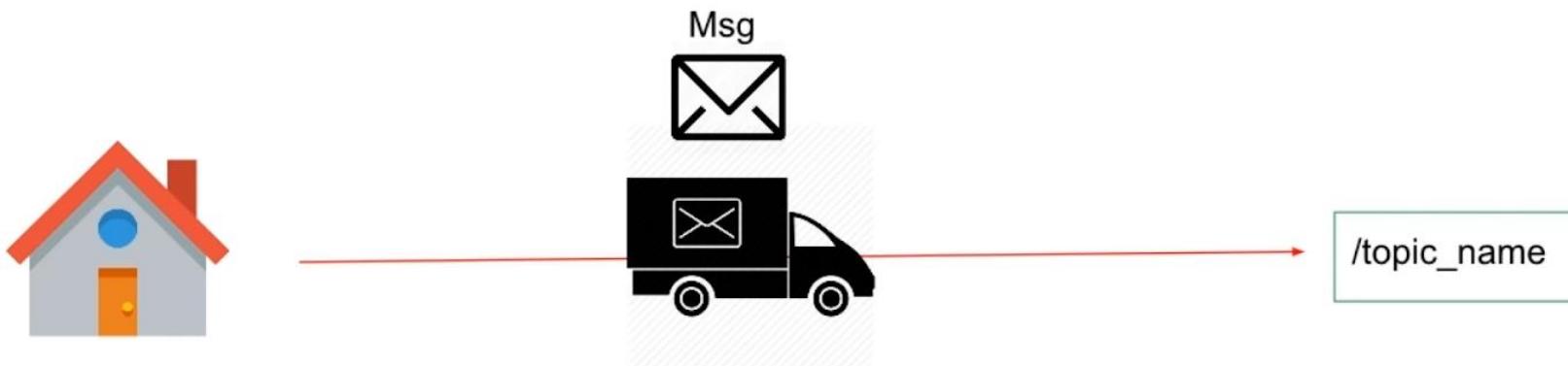
MSG & SRV

MSG in SRV

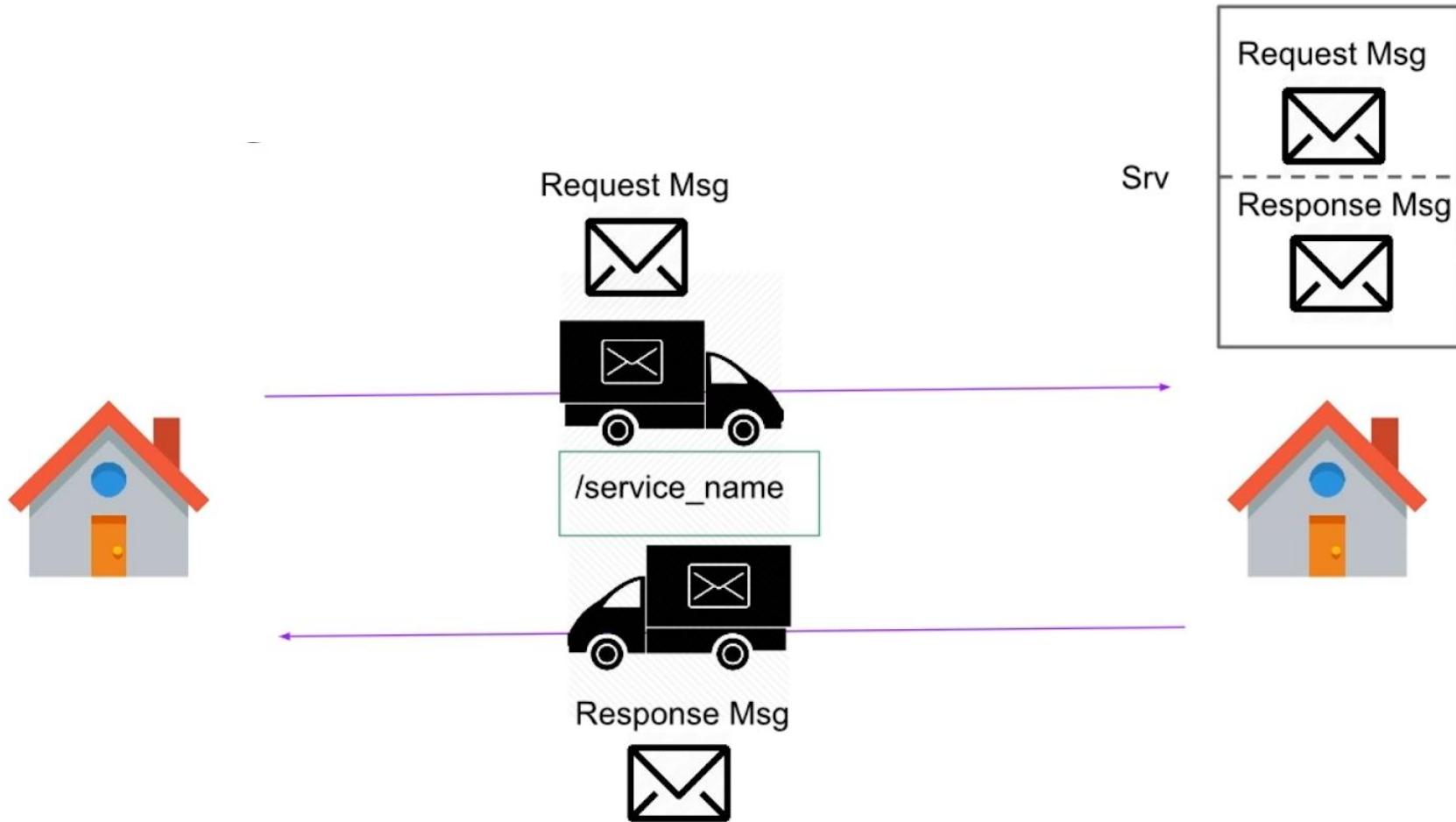
- Topic:
 - Ime: /sick_safetyscanners/scan
 - Definicija sporočila MSG: sensors_msgs/LaserScan
- Service:
 - Ime: /set_led_state
 - Definicija sporočila SRV: std_srvs/SetBool
 - Request: MSG
 - Response: MSG

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

MSG



SRV



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 kdr_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)

```
$ rosdep init  
$ rosdep update  
$ roscd kdr_msgs  
$ mkdir msg  
$ touch ledStatus.msg  
$ code ledStatus.msg
```

```
int64 ledNumber  
string ledStatus
```

```
$ catkin_make
```

Uporaba

package.html (kdr)

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

CMakeLists.txt (kdr)

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

Nov SRV (rpi_msgs)

```
$ roscd kdr_msgs  
$ mkdir srv  
$ touch safetyZone.srv  
$ code safetyZone.srv
```

int16 zone

bool success
string message

```
$ catkin_make
```

Uporaba

- CMakeLists.txt (kdr)

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

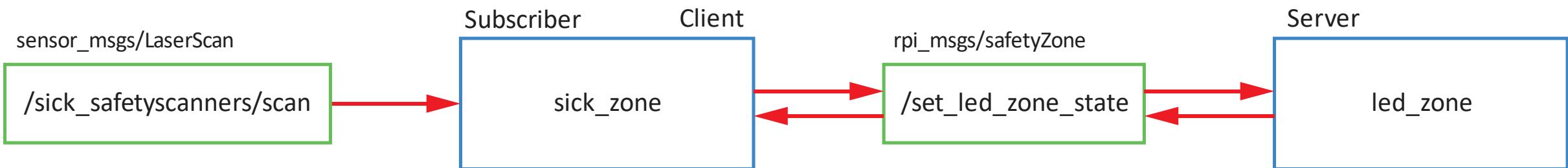
DEBUG MSG/SRV

- `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: $0,4 \text{ m} > d > 0,2$ m
 - Rdeča: $d < 0,2$ m



ACTIONS

Actions

Wolt

Išči

SS



work (Tržaška cesta 25 Šlajpah Sebastjan) ▾



Raziski



Restavracije



Trgovine

Restavracije



DiMatteo Gelateria

Gurmansi sladoledi in slastne tortice, nar...

15-25
minut

Slaščičarna Galerija Grad

Najslajše v mestu

15-25
minut

Olimpija Burek Rimská

Najboljši burek v mestu in še več

15-25
minut

Trappa Pizzeria

Pice iz kislega testa, nežno vzhajane in hit...

20-30
minut

Hong Xiang

Kitajska hrana za dušo

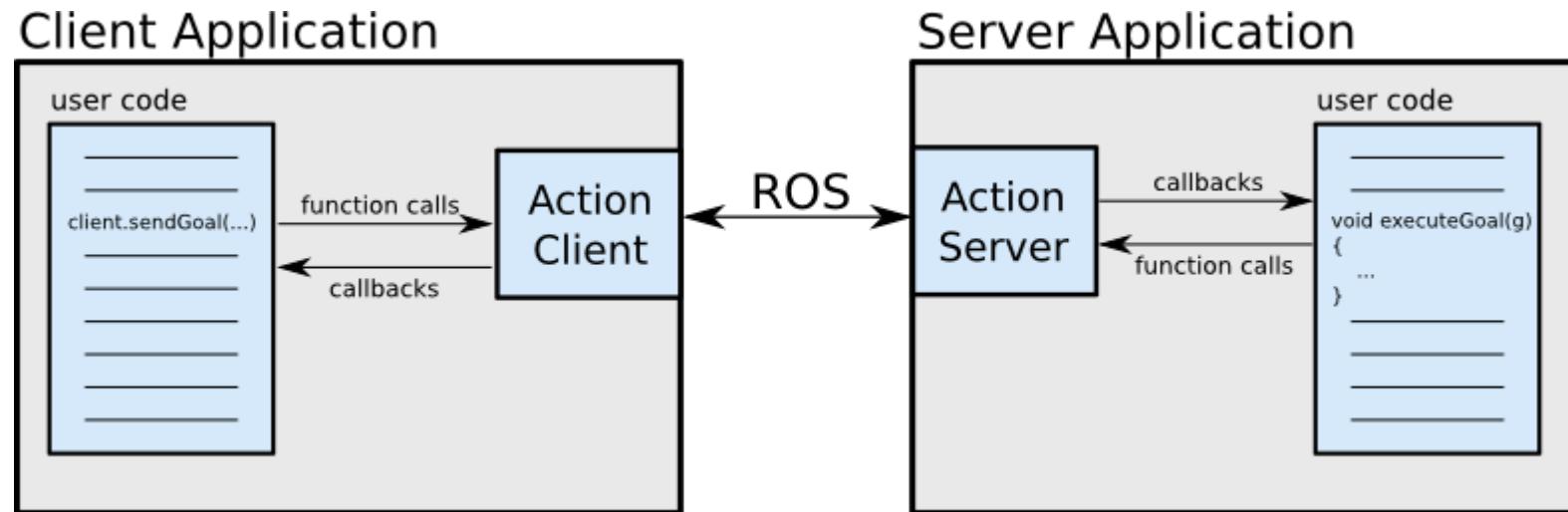
20-30
minut

Pinsa Rustika

Slastne pinsse s hrustljivim, a obenem me...

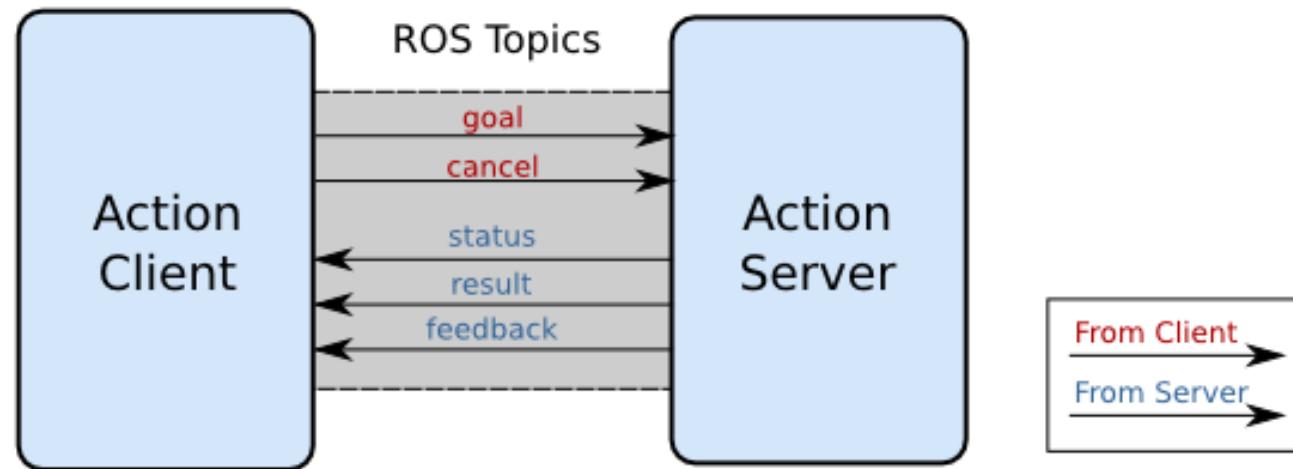
20-30
minut

Actions



Actions

Action Interface



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

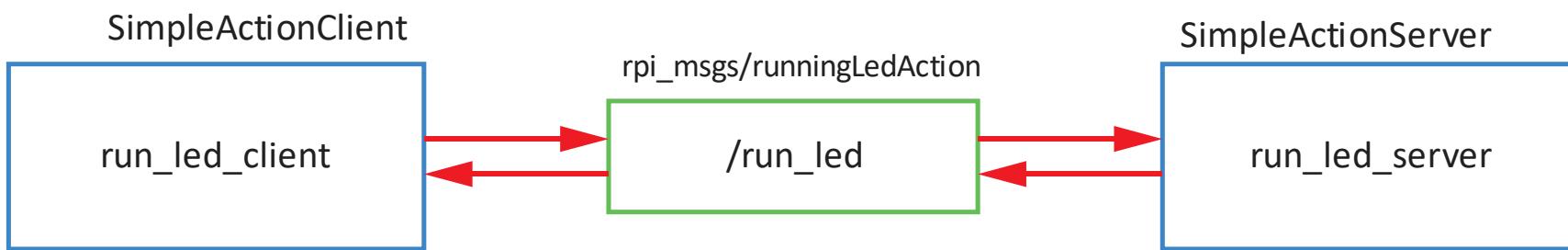
as_name/goal

as_name/result

as_name/status

Primer

- n-kratno izvajanje sekvenčnega prižiganja LED
- `ledrun_client.py`
- `ledrun_server.py`



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()

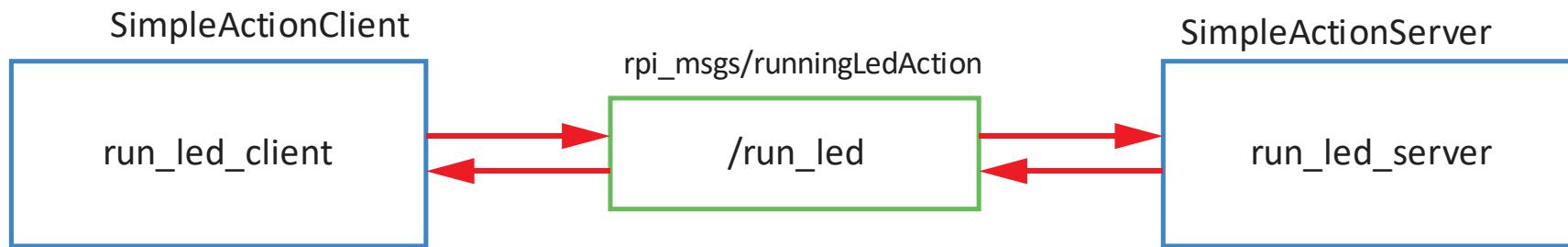
    if current_state == WARN:
        rospy.logwarn("[Warn] Warning on the action server side.")

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

    return client.get_result()
```

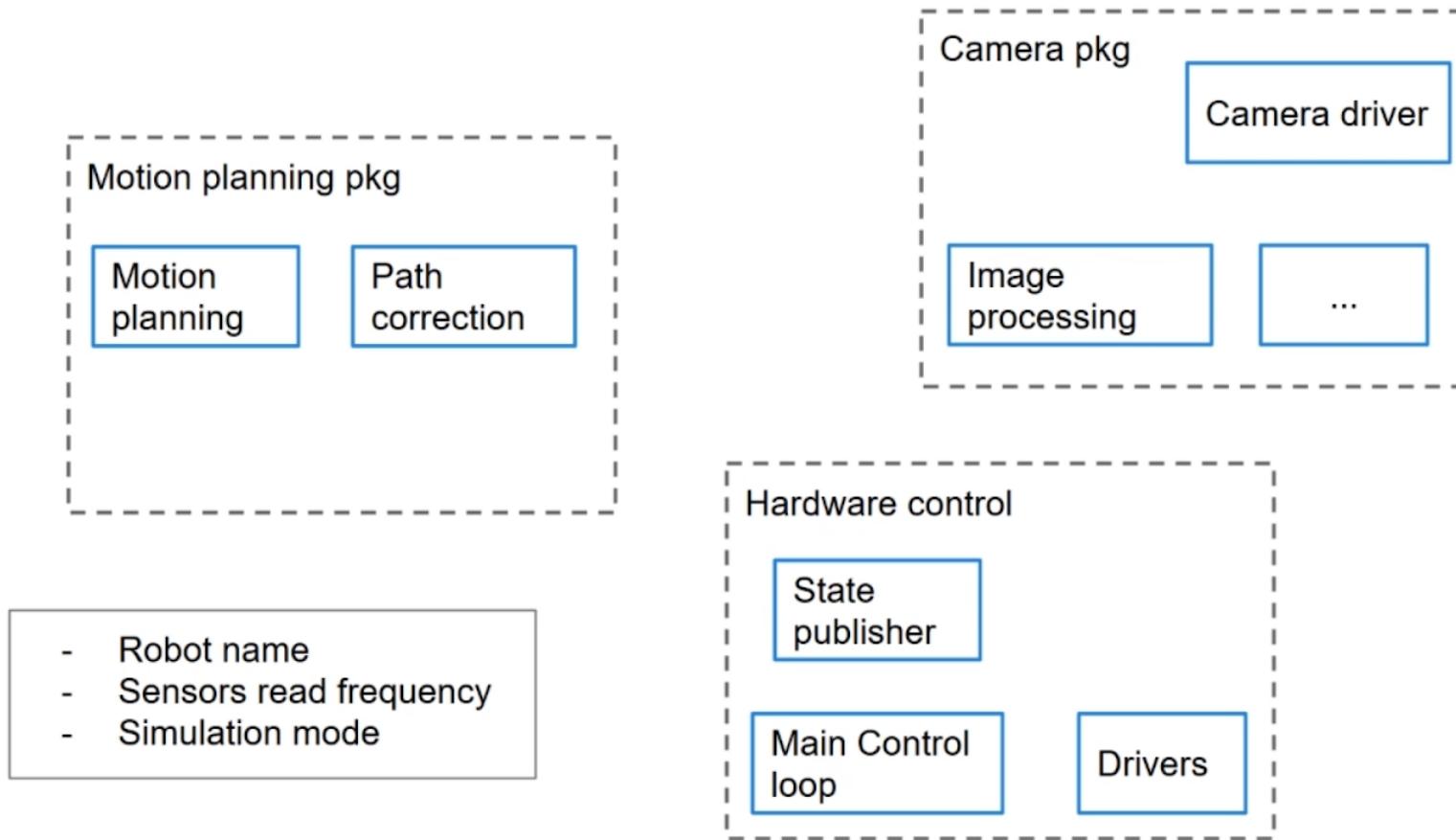
Naloga

- n-kratno izvajanje sekvenčnega prižiganja LED
- ustavi prižiganje LED, če je objekt bližje kot 20 cm.

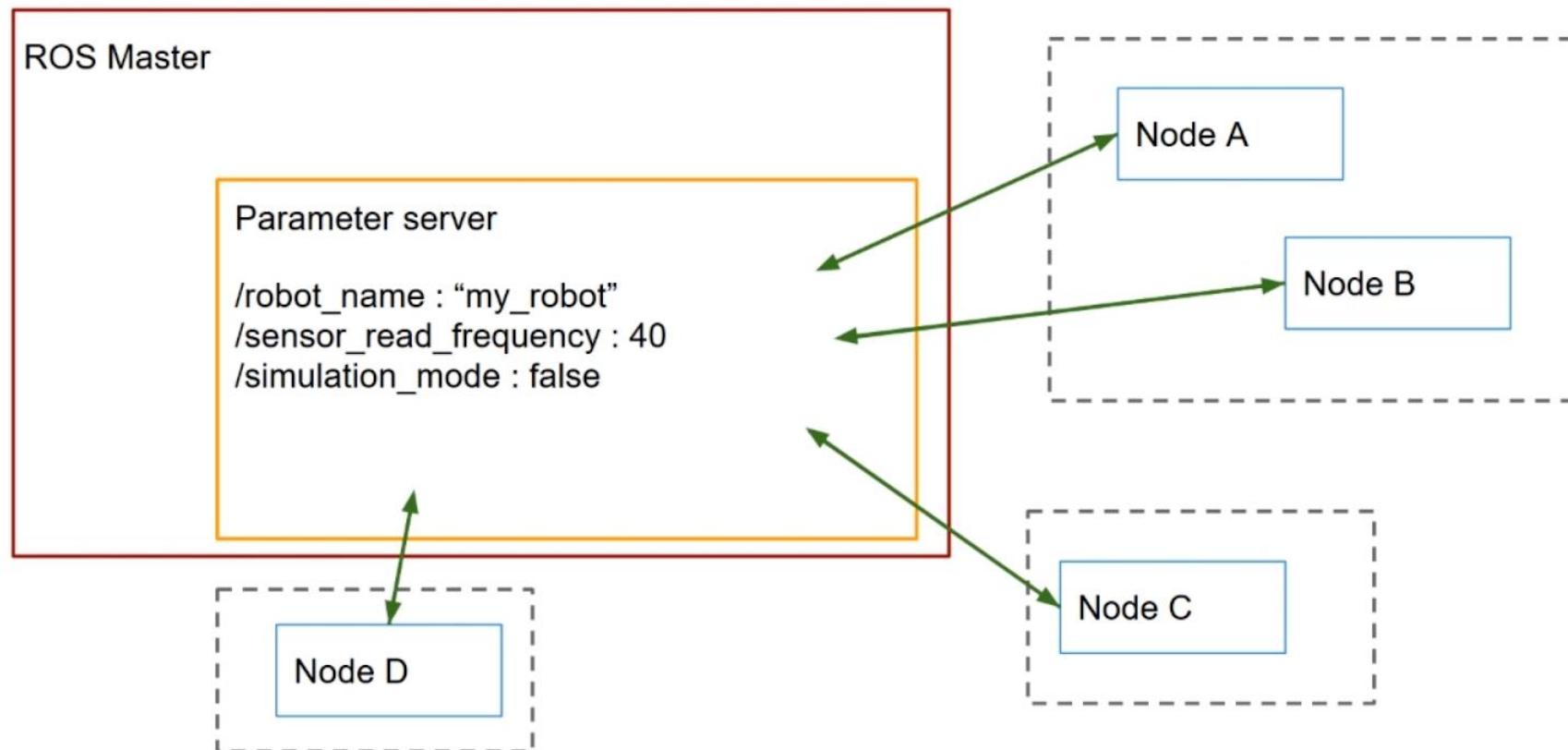


PARAMETERS

Parameters



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>      ... tako ga tudi ustvariš  
$ rosparam get <param name>  
$ rosparam list
```

- Primer:

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

```
$ rosparam set /number_publish_freq 2
```

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