Introduction to Robot Operating System (ROS 1)

Playing with ROS nodes, topics and messages- turtlesim example

discusses the use of: roscore, rosnode, and rosrun commandline tools

Dr. Essa Alghannam

roscore command

- roscore is the first thing you should run when using ROS, before starting with anything,

```
$ roscore
```

1-Master + rosout node
2-It's a crucial component that acts as a central server or the central hub of a ROS network (the heart of the entire system). It's essential for ROS nodes to communicate smoothly and exchange data effectively (air traffic control tower for your ROS network).

Ros Master

- Nodes Management: The ROS Master maintains a registry of all nodes on the network, allowing them to discover each other and their capabilities.
- Topic Management: It handles the creation and management of topics – the channels through which nodes publish and subscribe to messages.
- Parameter Server Management: It provides a central location for storing configuration parameters that can be accessed by all nodes.
- Service Management: It manages ROS services, which allow nodes to request specific actions or data from other nodes.
- Time Synchronization: Ensures all nodes in the system have a common understanding of time.

Notes - Ros Master

- A single ROS Master should run for each network.
- If you have multiple computers, you'll usually have one designated as the primary ROS Master.
- ROS Nodes: When you run ROS nodes (like your publisher or subscriber nodes), they automatically connect to the running ROS Master.

Example ROS network on same computer

In Terminal 1:

roscore

In Terminal 2:

```
rosrun my_package my_publisher_node.py
```

In Terminal 3:

```
rosrun my_package my_subscriber_node.py
```

- In this example, the `roscore` in Terminal 1 starts the ROS Master. The
- `my_publisher_node` and `my_subscriber_node` scripts in Terminals 2 and 3 will automatically connect to the ROS Master to communicate with each other.

Example ROS network between two computers

laptop

- Start roscore: Open a terminal on your laptop and run: bash roscore This starts the ROS Master, which acts as the central hub for communication between nodes.

Raspberry Pi

- Install ROS Noetic on Raspberry Pi: Follow the instructions for setting up ROS Noetic on the Raspberry Pi (https://wiki.ros.org/noetic/Installation/RaspberryPi).
- Connect Raspberry Pi to Network: Ensure your Raspberry Pi is connected to the same network as your laptop (wired or wireless).
- Set ROS Master Address: You need to tell the Raspberry Pi to connect to your laptop's ROS Master. Edit the ~/.bashrc file on the Raspberry Pi and add the following: bash export ROS_MASTER_URI=http://your_laptop_ip_address:11311 Replace your_laptop_ip_address with the actual IP address of your laptop.
- Test ROS Connection: After the Raspberry Pi reboots, open a terminal on it and run: bash rosnode list You should see a list of nodes running on your laptop's ROS Master.

Firewall: Ensure that the ROS ports (like 11311) are not blocked by firewalls on any of the computers.

running `roscore` on each computer in a lab setting for teaching purposes

1. Isolated Networks:

Create Separate Subnets: The key is to isolate the student workstations into separate subnets. This means each computer has a different IP address range, preventing them from directly communicating with each other.

Network Configuration: Ensure that each subnet is properly configured with its own gateway and DNS server.

2. Dedicated `roscore` per Workstation:

Run `roscore` on each computer: Each student's workstation will have its own `roscore` running.

Adjust `ROS_MASTER_URI`: Each `roscore` instance will have a unique ROS master URI. You'll need to configure the `ROS_MASTER_URI` environment variable on each workstation to point to its respective `roscore`.

running `roscore` on each computer in a lab setting for teaching purposes

- Example Configuration (for demonstration):

Workstation 1:

* IP: 192.168.1.10

* ROS_MASTER_URI: `http://192.168.1.10:11311`

Workstation 2:

* IP: 192.168.2.10

* ROS_MASTER_URI: `http://192.168.2.10:11311`

roscore command

If roscore does not initialize and sends a message about lack of permissions, probably the ~/.ros folder is owned by root, change recursively the ownership of that folder with:

```
$ sudo chown -R <your_username>  ~/.ros
$ essa@essa:~$ id -un
essa
$ sudo chown -R essa  ~/.ros
```

The command `sudo chown -R essa ~/.ros` does the following:

* Uses `sudo` to elevate your permissions to the root user.
* Uses `chown` to change the ownership of the `~/.ros` directory.
* Uses the `-R` flag to change the ownership recursively, affecting all files and subdirectories within `~/.ros`.
* Sets the new owner to the user `essa`.

roscore command output

```
essa@essa:~$ roscore
... logging to /home/essa/.ros/log/fbb5336a-790f-11ef-bd9f-af7604c9ebae/roslaunch-essa-2665.log
Checking log directory for disk usage. This may take a while.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://essa:43633/
ros_comm version 1.16.0


SUMMARY
========

PARAMETERS
 * /rosdistro: noetic
 * /rosversion: 1.16.0
```

Logging to `/home/essa/.ros/log/ fbb5336a-790f-11ef-bd9f-af7604c9ebae / roslaunch-essa-2665..log`

* `logging to ...`: This tells you where ROS is storing its logs, which are useful for debugging problems.
* `/home/essa/.ros/log`: This is the location of your ROS log directory, typically found in your home directory.
* `a951c51e-7821-11ef-a4fd-4d1c265cc579`: This is a unique ID assigned to this particular ROS session (or "run"). It helps keep logs from different sessions organized.
* `roslaunch-essa-15297.log`: This is the specific log file being used for this `roscore` instance.

`started roslaunch server http://essa:42663/`

* `started roslaunch server`: ROS is running a server (likely using a web-based interface for managing your ROS environment)
* `http://essa:42663/`: This is the address (IP address and port) where you can access the roslaunch server.

* `PARAMETERS`: This lists parameters set for your ROS environment.
  * `/rosdistro: noetic`: This is the ROS distribution you're using (in this case, Noetic).
  * `/rosversion: 1.16.0`: This indicates the ROS version.

roscore command output

```
NODES

auto-starting new master
process[master]: started with pid [2694]
ROS_MASTER_URI=http://essa:11311/

setting /run_id to fbb5336a-790f-11ef-bd9f-af7604c9ebae
process[rosout-1]: started with pid [2707]
started core service [/rosout]
```

PID: (Process ID) A unique number assigned to each running process by the operating system.

`NODES`: This section shows the nodes that are running.

1- `auto-starting new master`: The ROS Master is starting automatically.

2- `process[master]: started with pid [2694]`: The master process has been started with a process ID (PID) of 2694.

3- `ROS_MASTER_URI=http://essa:11311/`: This is the URL you can use to connect to the ROS Master from other nodes.

4- `setting /run_id to fbb5336a-790f-11ef-bd9f-af7604c9ebae`: A unique ID is set for this ROS session, as mentioned earlier.

5- `process[rosout-1]: started with pid [2707]`: The `/rosout` logging node has also started with a PID of 2707.

6- `started core service [/rosout]`: This confirms that the `/rosout` core service is running.

جَامعة
المَنارة
MANARA UNIVERSITY

Understanding ROS Nodes

The ROS Computation Graph level- NODES

- Nodes: The basic unit of ROS. an executable file within a ROS package.
- Nodes are written with an ROS client library, for example, roscpp (c++ client library) or rospy (python client library).
- Each ROS node is a program that performs a specific task.
- A node must have a unique name in the system. This name is used to permit the node to communicate with another node using its name without ambiguity.
- Nodes can publish or subscribe to a Topic. Nodes can also provide or use a Service.

Example:

Think of a robot navigating a room.

Nodes:

1 `Camera Node`: Captures images from the camera.

2 `Motion Planning Node`: Generates a path for the robot.

3 `Motor Control Node`: Controls the robot's movements.

ROS Master: The ROS Master helps the nodes find each other and communicate.

rosnode command-line tool

The tool rosnode is a command-line tool for displaying information about nodes that are currently running.

The commands supported are as follows:

- rosnode list: This lists the active nodes. to get information about the nodes that are running.
- rosnode info node: This prints information about the node.
- rosnode ping node: This tests the connectivity to the node.

rosnode list command

Open up a new terminal, and let's use rosnode to see what running roscore did...

$ rosnode list: lists these active nodes

You will see only node running is :

/rosout

It is normal because this node runs whenever roscore is run.

it is the default ROS logger.

1- Error Reporting: If any ROS node encounters an error, it will typically send a message to `/rosout`, which then logs it.

2-Always Active: `/rosout` is always running in the background, ready to collect these messages.

rosnode info command

$ rosnode info /rosout

- `rosnode info`: This is the ROS command used to obtain information about a specific ROS node.
- `/rosout`: This is the name of the ROS node you want to get information about.
- `/rosout` is the core ROS logging node.

$ rosnode info /rosout

```
essa@essa:~$ rosnode info /rosout
--------------------------------------------------------------------------------
Node [/rosout]
Publications:
 * /rosout_agg [rosgraph_msgs/Log]

Subscriptions:
 * /rosout [unknown type]

Services:
 * /rosout/get_loggers
 * /rosout/set_logger_level

contacting node http://essa:33053/ ...
Pid: 2707
```

Nodes topics messages

Type of messages

Topics names

services

Using rosrun

open a new terminal

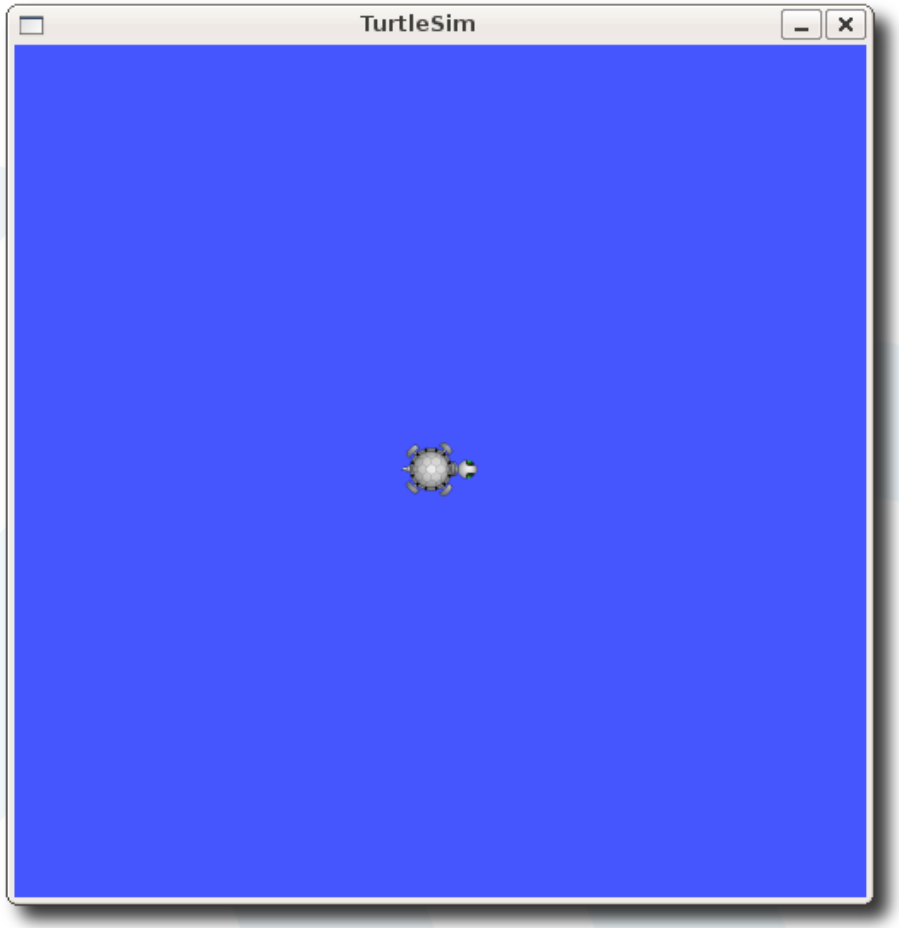
https://wiki.ros.org/turtlesim

rosrun allows you to use the package name to directly run a node within a package (without having to know the package path).

$ rosrun [package_name] [node_name]

$ rosrun turtlesim turtlesim_node // existed package and node

$ rosrun turtlesim turtlesim_node_name:=???



$ rosnode list

- /rosout
- /turtlesim

a new node with the name /turtlesim

This is a timestamp, representing the time (in seconds since the Unix epoch) when the message was generated.

Default name

[INFO] [1730457000.440218556]: Starting turtlesim with node name /turtlesim

[INFO] [1730457000.447463127]: Spawning turtle [turtle1] at x=[5.544445], y=[5.544445], theta=[0.000000]

Position in pixels

Orientation In radians

the origin (the bottom-left corner) of the `turtlesim` window

$ rosnode list

- /rosout
- /turtlesim

a new node with the name /turtlesim

call this command:

rosnode ping /turtlesim

it shows

rosnode: node is [/turtlesim]

pinging /turtlesim with a timeout of 3.0s

xmlrpc reply from http://essa:35091/ time=2.931118ms

xmlrpc reply from http://essa:35091/ time=2.041340ms

to send a "ping" message to the `/turtlesim` node. It sets a timeout of 3 seconds. If the node doesn't respond within that time, the ping will fail.

This means the `/turtlesim` node successfully responded to the ping request.

http://essa: 35091 /` is the URI (Uniform Resource Identifier) of the `/turtlesim` node.
"Essa“: hostname of your computer,
35091 is a port number used for ROS communication.

how long it took for the node to respond to the ping request.

XML-RPC (eXtensible Markup Language - Remote Procedure Call)

- XML (eXtensible Markup Language): A standardized way to structure data in a text-based format.
- XML is like a set of tags that define elements and attributes, making data easily readable and transferable.
- RPC (Remote Procedure Call): A mechanism for one computer to call a function or procedure on another computer.
- It's like making a request to another computer to do something.
- In ROS, XML-RPC is used for communication between different ROS nodes (programs).

Close the previous turtlesim window and recall:

```
rosrun turtlesim turtlesim_node __name:=my_turtle
```

```
$ rosnode list
/my_turtle
/rosout
```

```
rosnode ping my_turtle
```

The command `rosnode ping my_turtle` in ROS is used to check if a specific ROS node "my_turtle" is alive and responsive.

```
rosnode: node is [/my_turtle]
pinging /my_turtle with a timeout of 3.0s
xmlrpc reply from http://essa:38571/  time=0.844955ms
xmlrpc reply from http://essa:38571/  time=1.895905ms
```

Review

What was covered:

roscore = ros+core : master (provides name service for ROS) + rosout (stdout/stderr) + parameter server (parameter server will be introduced later)

rosnode = ros+node : ROS tool to get information about a node.

rosrun = ros+run : runs a node from a given package.

جَامعة المَنارة
MANARA UNIVERSITY

Understanding ROS Topics

3. Topics

- Topics: Channels for data exchange.
- Nodes publish messages to topics and other nodes subscribe to those topics to receive data.
- Topics: when a node is sending data, we say that the node is publishing a topic.
- Nodes can receive topics from other nodes simply by subscribing to the topic.
- It's important that the name of the topic must be unique.
- A node can subscribe to a topic only if it has the same message type.
- Nodes can publish messages to a topic as well as subscribe to a topic to receive messages.

Example:

- Think of a robot navigating a room.

Nodes:

- `Camera Node`: Captures images from the camera.
- `Motion Planning Node`: Generates a path for the robot.
- `Motor Control Node`: Controls the robot's movements.

Topics:

- `/camera/image`: The camera node publishes images to this topic.
- `/navigation/goal`: The motion planning node subscribes to this topic to receive the desired destination.
- `/motor/commands`: The motion planning node publishes motor commands to this topic.

ROS Master: The ROS Master helps the nodes find each other and communicate.

Understanding ROS Topics

roscore

rosrun turtlesim turtlesim_node

rosrun turtlesim turtle_teleop_key

turtle keyboard teleoperation

- The turtlesim_node and the turtle_teleop_key node are communicating with each other over a ROS Topic.
- turtle_teleop_key is publishing the key strokes on a topic, while turtlesim subscribes to the same topic to receive the key strokes. move the turtle using the arrow keys
- rqt_graph shows the nodes and topics currently running.

If you have another turtlesim window by calling this command:

rosrun turtlesim turtlesim_node __name:=my_turtle

the two windows will response to turtle_teleop_key

rosnode list

/rosout

/teleop_turtle

/turtlesim

rosnode info /turtlesim

Node [/turtlesim]
Publications:
 * /rosout [rosgraph_msgs/Log]
 * /turtle1/color_sensor [turtlesim/Color]
 * /turtle1/pose [turtlesim/Pose]

Subscriptions:
 * /turtle1/cmd_vel [geometry_msgs/Twist]

Services:
 * /clear
 * /kill
 * /reset
 * /spawn
 * /turtle1/set_pen
 * /turtle1/teleport_absolute
 * /turtle1/teleport_relative
 * /turtlesim/get_loggers
 * /turtlesim/set_logger_level

contacting node http://essa:35091/ ...
Pid: 5826
Connections:
 * topic: /rosout
   * to: /rosout
   * direction: outbound (50707 - 127.0.0.1:40716) [26]
   * transport: TCPROS
 * topic: /turtle1/cmd_vel
   * to: /teleop_turtle (http://essa:36819/)
   * direction: inbound (36804 - essa:57261) [28]
   * transport: TCPROS

rosnode kill /turtlesim

This command specifically tells the ROS master to terminate the node named `/turtlesim`.

kill -9 5826

$ rosnode info /teleop_turtle

$ rosnode ping /teleop_turtle

$ rosnode kill / teleop_turtle

Try after that to call:

$ rosnode ping /teleop_turtle

$ rosnode list

- a common issue when working with ROS.
- a central "master" node that keeps track of all other nodes in the network. Even if you kill the node, the ROS master might still think the node is alive.
- Process Still Running: Sometimes, the node process itself might not have completely shut down, even if the window is closed. This can happen due to background processes or lingering connections.

Using rqt_graph

- rqt_graph creates a dynamic graph of what's going on in the system.
- rqt_graph is part of the rqt package.
- Unless you already have it installed, run:

```
$ sudo apt-get install ros-noetic-rqt
$ sudo apt-get install ros-noetic-rqt-common-plugins
```

In a new terminal: `rosrun rqt_graph rqt_graph`

- the turtlesim_node and the turtle_teleop_key nodes are communicating on the topic named: /turtle1/command_velocity.

Introducing rostopic

The rostopic tool allows you to get information about ROS topics.

You can use the help option to get the available sub-commands for rostopic

```
$ rostopic -h
```

rostopic bw /topic: display bandwidth used by topic

rostopic echo /topic: print messages to screen

rostopic hz /topic: display publishing rate of topic

rostopic list print information about active topics

rostopic pub publish data to topic

rostopic type /topic: This prints the topic type (the type of message it publishes).

rostopic find message_type: This finds topics by their type.

rostopic info /topic: This prints information about the active topic.

Or pressing tab key after rostopic prints the possible sub-commands:

```
$ rostopic
```

bw echo find hz info list pub type

Using rostopic echo

rostopic echo shows the data published on a topic.

rostopic echo [topic]

Let's look at the command velocity data published by the turtle_teleop_key node.

Run the following command line , use the arrow keys to see what data is being sent:

In a new terminal, run:

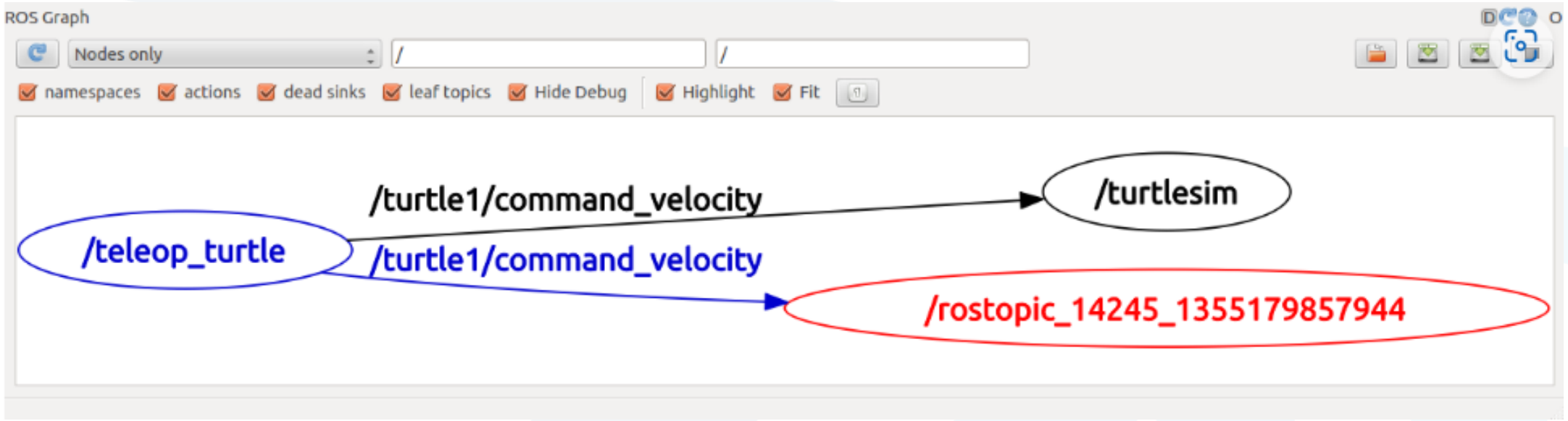
$ rostopic echo /turtle1/cmd_vel

we can see the type of message sent by the topic using the following command lines:

$ rostopic type /turtle1/cmd_vel

geometry_msgs/Twist

```
linear:
  x: 2.0
  y: 0.0
  z: 0.0
angular:
  x: 0.0
  y: 0.0
  z: 0.0
---
linear:
  x: 2.0
  y: 0.0
  z: 0.0
angular:
  x: 0.0
  y: 0.0
  z: 0.0
---
```



rostopic echo, shown here in red, is now also subscribed to the turtle1/command_velocity topic.

Using rostopic list

$ rostopic list -h

Options:

-h, --help	show this help message and exit
-b BAGFILE, --bag=BAGFILE	(list topics in .bag file)
-v, --verbose	list full details about each topic
-p	list only publishers
-s	list only subscribers

Using rostopic list -v

rostopic list -v

This displays a verbose list of topics to publish to and subscribe to and their type.

Published topics:

* /turtle1/color_sensor [turtlesim/Color] 1 publisher
* /turtle1/cmd_vel [geometry_msgs/Twist] 1 publisher
* /rosout [rosgraph_msgs/Log] 2 publishers
* /rosout_agg [rosgraph_msgs/Log] 1 publisher
* /turtle1/pose [turtlesim/Pose] 1 publisher

This tells you that one node you queried is publishing message about velocity commands (of type `geometry_msgs/Twist`) to the `/turtle1/cmd_vel` topic. This is likely the command topic used to control the turtle's movement in `turtlesim`.

Subscribed topics:

* /turtle1/cmd_vel [geometry_msgs/Twist] 1 subscriber
* /rosout [rosgraph_msgs/Log] 1 subscriber

جَامعة المَنارة
MANARA UNIVERSITY

شكرا لحسن الاصغاء