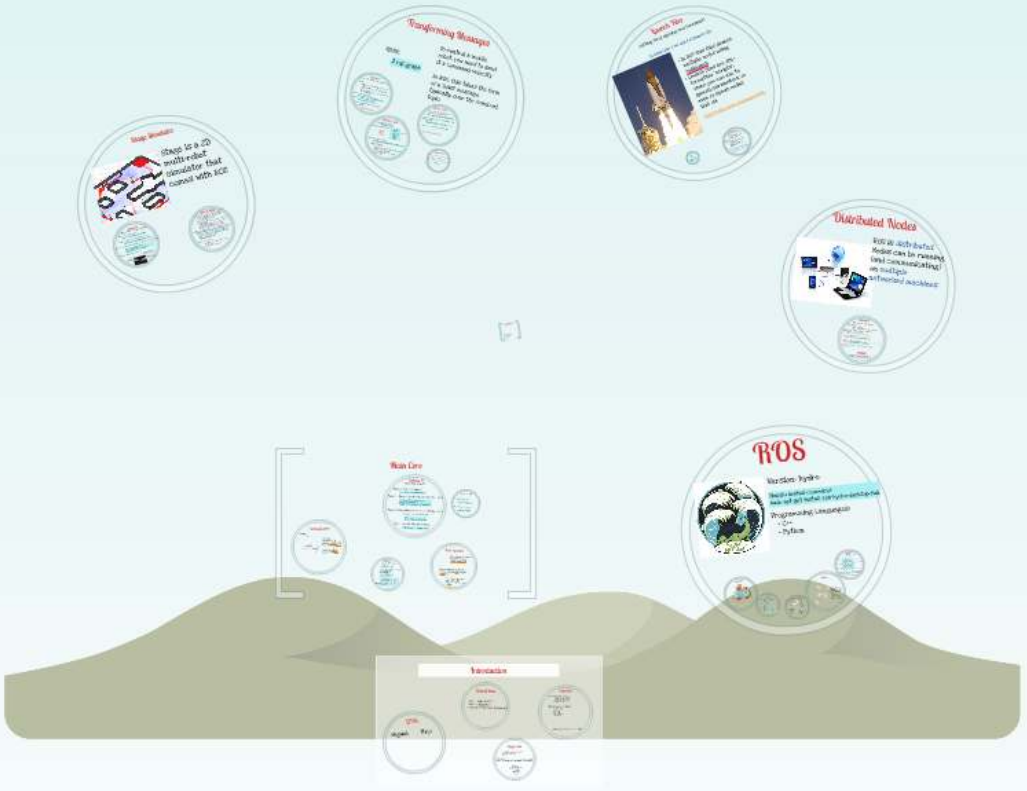
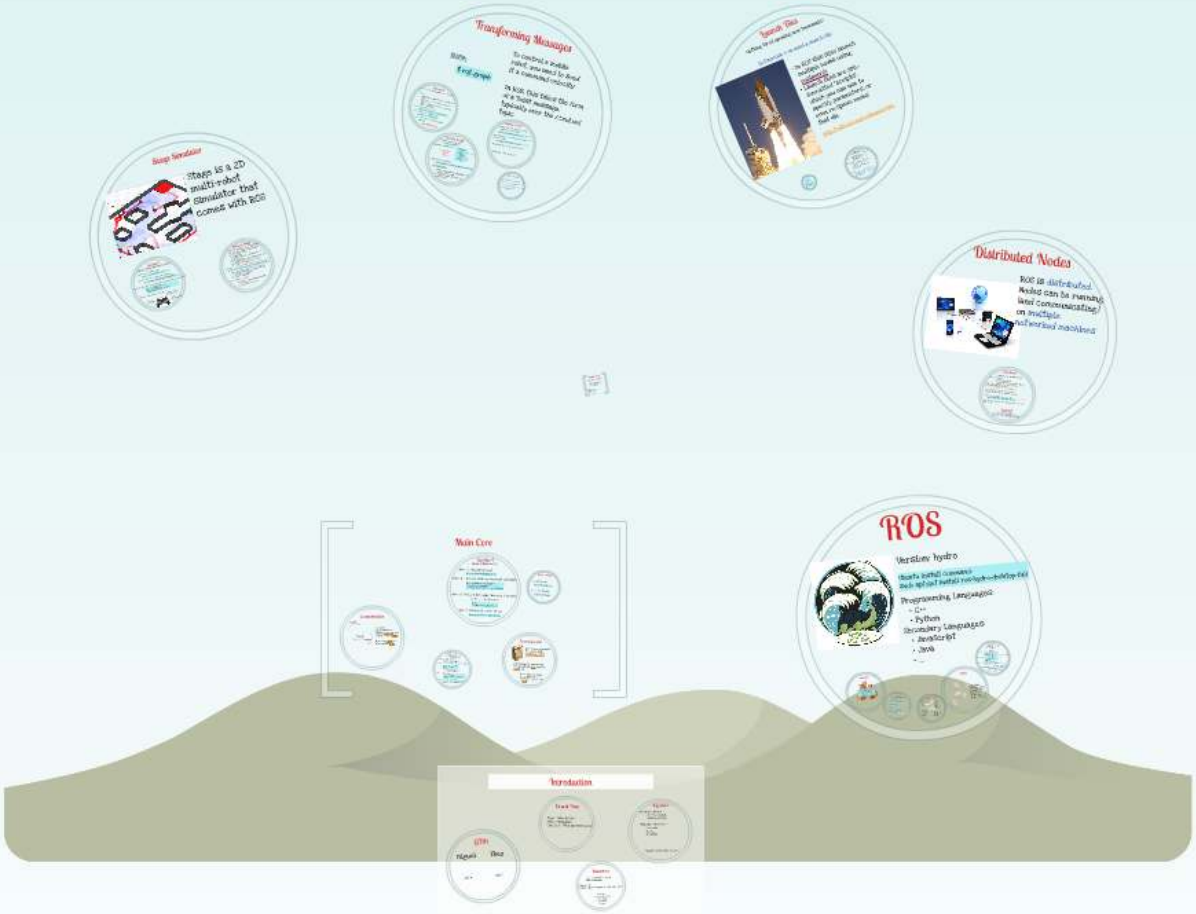
# Intro To ROS



# Intro To ROS


Transforming Messages
Stage
tf
Distributed Nodes
Main Core
ROS
Introduction

# Introduction

## Overal Plan

- Day 1 - Intro to ROS
- Day 2 - Navigation
- Day 2 & 3 - Xtion and Transforms

## Important

PC User Account
- Username: human
- Password: human

Bitbucket Repository
- ~/hcr2013
- hg pull
- hg update

Ready to start the tutorial

## GTAs

Miguel — ms606

Theo — tg108

## Resources

ROS : Robot Operating System
http://wiki.ros.org/

Bitbucket Url:
https://bitbucket.org/personal_robotics/hcr2013/

Hardware:
- P3-AT mobile robots
- 2 x Laptops
- Gamepad
- Mouse

# GTAs

Miguel

ms606

Theo

tg108

# Overal Plan

- Day 1 - Intro to ROS
- Day 2 - Navigation
- Day 2 & 3 - Xtion and TranSforms

# Resources

ROS : Robot Operating System
http://wiki.ros.org/

Bitbucket Url:
https://bitbucket.org/personal_robotics/hcr2013/

Hardware:

- P3-AT mobile robots
- 2 x Laptops
- Gamepad
- Mouse

# Important

## PC User Account

- Username: human
- Password: human

## Bitbucket Repository

- ~/hcr2013
- hg pull
- hg update

Ready to start the tutorial

# ROS


## Version: hydro

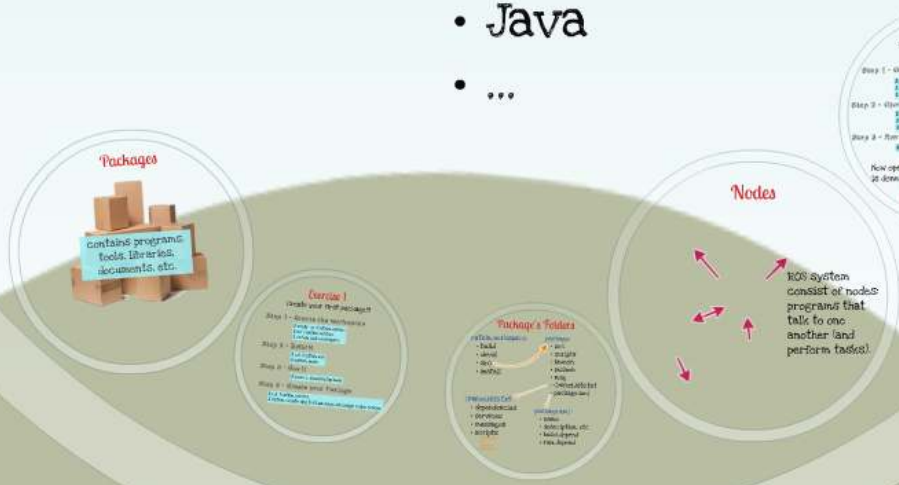
Ubuntu install command:
sudo apt-get install ros-hydro-desktop-full

## Programming Languages:

- C++
- Python

## Secondary Languages:

- JavaScript
- Java
- ...



# Packages

contains programs,
tools, libraries,
documents, etc.

Creat

Step 1 - Cre

$ mkd
$ cd ~/
$ catk

Step 2 - Build

$ cd ~/c
$ catkin

Step 3 - U

# Exercise 1

Create your first package!!!

## Step 1 - Create the workspace

```
$ mkdir -p ~/catkin_ws/src
$ cd ~/catkin_ws/src
$ catkin_init_workspace
```

## Step 2 - Build it

```
$ cd ~/catkin_ws/
$ catkin_make
```

## Step 3 - Use it

```
$ source devel/setup.bash
```

## Step 4 - Create your Package

```
$ cd ~/catkin_ws/src
$ catkin_create_pkg test_package std_msgs rospy roscpp
```

# Package's Folders

catkin_workspace:

- build
- devel
- src
- install

package:

- src
- scripts
- launch
- include
- msg
- CMakeLists.txt
- package.xml

CMakeLists.txt:

- dependencies
- services
- messages
- scripts

  Installations and Compilation instructions

package.xml :

- name
- description, etc
- build_depend
- run_depend

$ roSrun exercise2 joy

Now open the code an
is done. $ gedit src/exercis

More...? Have a

# Nodes

ROS system consist of nodes: programs that talk to one another (and perform tasks).

lders

ckage:
src
scripts
launch
include
msg
CMakeLists.txt
package.xml

ge.xml :
ne
cription, etc
d_depend
_depend

# Exercise 2

Create a node that reads from joystick

## Step 1 - Go to our directory

```
$ cd ~/hcr2013/exercises/ros_intro
$ catkin_make
$ source devel/setup.bash
```

## Step 2 - Open the ROS Master (on separate terminal)

```
$ cd ~/hcr2013/exercises/ros_intro
$ source devel/setup.bash
$ roscore
```

## Step 3 - Run the Joystick Node

```
$ rosrun exercise2 joystick_node.py
```

Now open the code and see how the magic is done. `$ gedit src/exercise2/scripts/joystick_node.py &`

More...? Have a look at the wiki

# Main Core

## Exercise 4

Record stuff on ROS

**Step 1 - Run Exercise4**

```
$ rosrun exercise4 joystick_node.py
```

**Step 2 - Record with rosbag (new Terminal)**

```
$ cd ~/hcr2013/exercise5/ros_intro
$ source devel/setup.bash
$ rosbag record joystick -O my_first_bag.bag
```

**Step 3 -Record for some time and then stop**

Use CTRL - C : to stop the recording

```
$ ls
$ rosbag info my_first_bag.bag
```

**Also- Record all topics using**

```
$rosbag record -a -O another_bag.bag
```

### Use the recording

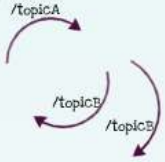
1. Replay the data

```
$ rosbag play my_first_bag.bag
```

2. Analyze the data

```
$ rqt_bag my_first_bag.bag
```

## Communication




ROS nodes communicate by passing messages over channels called topics

Nodes can publish or subscribe to topics

## Exercise 3

Create a talking node

Step 1 - Run Exercise3

Step 2 - Assess the topic(in another terminal)

'What happens?

Step 3 - Also try:

Check the code again and see the wiki:

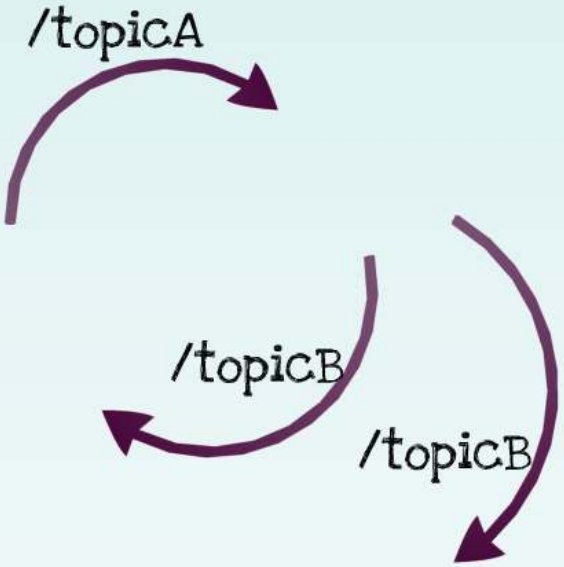
## Record you data

When doing an experiment is very important to RECORD YOUR DATA!!!

ROS has a built in function that records data into bags using the rosbag tool

Once recorded, you can playback or analyze the information. A useful GUI tool is rqt_bag

# Communication




ROS nodes communicate by passing messages over channels called topics.

Nodes can publish or subscribe to topics

# Exercise 3

## Create a talking node

### Step 1 - Run Exercise3

```
$ roSrun exercise3 joyStick_node.py
```

### Step 2 - Access the topic(On another terminal)

```
$ cd ~/hcr2013/exercises/roS_intro
$ source devel/setup.bash
$ roStopic echo /joyStick
```

## What happens?

### Step 3 - Also try:

```
$ roStopic list
$ roStopic type /joyStick
$ roStopic (then tap TAB twice to see all the
options)
```

## Check the code again and see the wiki:

```
$ gedit /exercise2/scripts/joyStick_node.py &
```

# Record you data

When doing an experiment is very important to RECORD YOUR DATA!!!

ROS has a built in function that records data into bags using the rosbag tool

Once recorded, you can play-back or analyze the information. A useful GUI tool is rqt_bag

# Exercise 4

## Record Stuff on ROS

### Step 1 - Run Exercise4

```
$ roSrun exerciSe4 joyStick_node.py
```

### Step 2 - Record with rosbag (new Terminal)

```
$ cd ~/hcr2013/exerciSeS/roS_intro
$ Source devel/Setup.baSh
$ roSbag record joyStick -O my_firSt_bag.bag
```

### Step 3 -Record for some time and then stop

USe CTRL - C : to Stop the recording

```
$ lS
$ roSbag info my_firSt_bag.bag
```

### Also- Record all topics using

```
$roSbag record -a -O another_bag.bag
```

# Use the recording

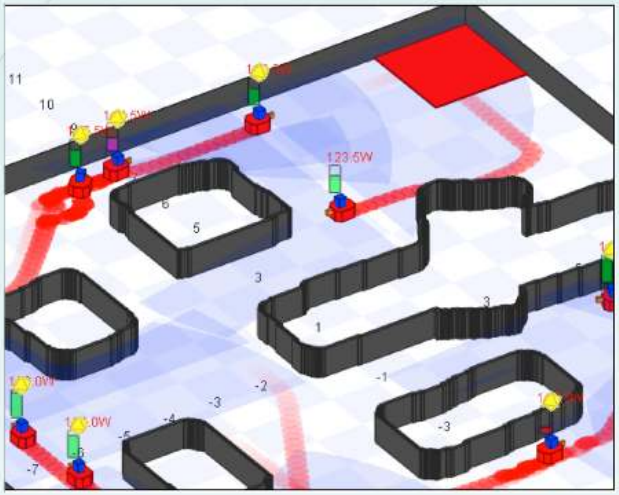
## 1. Replay the data

```
$ rosbag play my_first_bag.bag
```

## 2. Analyze the data

```
$ rqt_bag my_first_bag.bag
```

# Stage Simulator




Stage is a 2D multi-robot simulator that comes with ROS


# Exercise 5

## Getting to know the Stage Simulator

### Step 1 - cd to Exercise5

```
$ cd ~/hcr2013/exercise5/ros_intro/exercise5
```

### Step 2 - run stage with the world map

```
$ rosrun stage_ros stageros world/simple_map.world
```

TRY: View -> "Perspective Camera"

### Step 3 - Connect your joystick to the robot (new terminal)

```
$ cd ~/hcr2013/exercise5/ros_intro
$ source devel/setup.bash
$ roslaunch exercise5 exercise5.launch
```

### Step 4 - Using the Joystick

# More on Stage

You have almost complete control over the environment:

- Speed up and slow time (click Run to see options)
- Move objects around (click on an object and drag)
- you can even move the entire map
- Move freely around
- You can look around in the world by holding down CTRL and moving the mouse.
- Move around by holding down the ALT and moving the mouse
- Change what you see (click on view for options)
- Try clicking on Data (you'll see the laser scans)

## The World File: (Define models and then instantiate them to the world)

```
$gedit exercises/ros_intro/src/exercise5/world/simple_map.world &
```

- Line 58: floorplan model which defines our environment
- Line 67: we want the walls to be detectable by ranger scans
- Line 82: Create our own floor map
- See more in the Stage manual

## Change the floorplan map:

- Change line 87 to : bitmap "map_level10_ros.pgm"
- Rerun the stage!!!!
- Change line 90 to: size [20 15 0.5]

# Transforming Messages

RUN:

```
$ rqt_graph
```

To control a mobile robot, you need to send it a command velocity

In ROS, this takes the form of a Twist message, typically over the /cmd_vel topic




# Exercise 6

Convert from Joystick Messages to Command Velocities

Review:

- Nodes are ROS programs that communicate with other nodes
- Nodes can publish or subscribe to messages over topics.
- Nodes can communicate only if they sent/receive the same message (topic)

## Step 1 - run Stage Simulator

```
$ rosrun stage_ros stageros world/simple_map.world
```

## Step 2 - run Our Joystick node

```
$ rosrun exercise6 joystick_node.py
```

## Step 2 - Run the connection's graph

```
$ rqt_graph
```

Any Problem?

# Exercise 6 cont.

## Stage's topic: cmd_vel

```
$ rostopic info /cmd_vel
```

geometry_msgs/Twist:

```
Vector3 linear
  float64 x
  float64 y      linear velocities
  float64 z
Vector3 angular
  float64 x      roll
  float64 y      pitch
  float64 z      yaw
```

We only need
- linear.x
- angular.z

## Joystick's topic: JoyAxis

```
$ gedit exercises/ros_intro/src/exercise6/msg/ JoyAxis.msg &
```

## Solutions:

1. Modify our joystick node to publish Twist messages over /cmd_vel
2. Create a new node that does this transformation

# Exercise 6 cont.

Step1: Open the file and edit it so that it runs correctly

```
$ gedit /exercise6/scripts/joystick_to_cmd_vel.py &
```

Step2: Run the node along with stage and joystick nodes

```
$ rosrun exercise6 joystick_to_cmd_vel.py
```

Step3: Try to move the virtual robot

Step4: Open the rqt_graph

# Solutions Exercise 6

- Line 14: `from geometry_msgs.msg import Twist`
- 
- Line 29: `roSpy.init_node('joystick_to_cmd_vel')`
- 
- Line 32: `Self.pub = roSpy.Publisher('cmd_vel', Twist)`
- 
- Line 59: `Self.pub.publish(cmd)`

# Launch Files

Getting tired opening new terminals?

In Exercise 5 we used a launch file

- In ROS this files launch multiple nodes using roslaunch
- Launch files are XML-formatted "scripts", which you can use to specify parameters, or even re-spawn nodes that die

http://wiki.ros.org/roslaunch/XML

# Exercise 7

## Launching Multiple Nodes

### Go to folder launch in exercise7

- simulation.launch
- joystick_controller.launch

### Xml Launch:

- <node> tag specifies a node to run
  - pkg: specifies which package is this node in
  - name: specifies the name of this node (it can be anything)
  - type: specifies the executable
  - args: the arguments to pass to the node

### Task:

1. Run simulation.launch using roslaunch
2. Complete and run joystick_controller.launch
3. See if they work
4. Merge all into one launch file (exercise7.launch)

# Solution

## joystick_controller.launch:

```
<launch>
    <!-- Start Joystick Node -->
    <node pkg="exercise7" type="joystick_node.py"
name="joy_node"/>

    <!-- Start Joystick converter -->
   <node pkg="exercise7" type="joystick_to_cmd_vel.py"
name="joy_to_cmd_vel"/>
</launch>
```

## exercise7.launch :

```
<launch>
    <!-- Start Joystick Node -->
    <node pkg="exercise7" type="joystick_node.py" name="joy_node"/>

    <!-- Start Joystick converter -->
   <node pkg="exercise7" type="joystick_to_cmd_vel.py" name="joy_to_cmd_vel"/>

   <!-- Stage Simulator -->
  <node pkg="stage_ros" type="stageros" name="stage"
        args="$(find exercise7)/world/simple_map.world"/>
</launch>
```

# Distributed Nodes

ROS is distributed.
Nodes can be running (and communicating) on multiple networked machines

## Exercise 8

Running Nodes on Multiple Machines

Using 2 Laptops:
- Run the stage on one
- Run the joystick launcher from the other

**Step1:** Go to roscore terminal and check ROS_MASTER_URI variable
ROS_MASTER_URI=http://[HOSTNAME]:11311/

**Step2:** Get the name of the computer running the roscore by typing "hostname"

**Step3:** On the other computer type
export ROS_MASTER_URI=http://[HOSTNAME]:11311/

**Step4:** Run launchers on separate computers from exercise 8

**DONE**

YOU JUST USED THE POWER OF DISTRIBUTED COMPUTING IN ROS

# Exercise 8

Running Nodes on Multiple Machines

Using 2 Laptops:

- Run the stage on one
- Run the joystick launcher from the other

**Step1:** Go to roscore terminal and check ROS_MASTER_URI variable

ROS_MASTER_URI=http://[HOSTNAME]:11311/

**Step2:** Get the name of the computer running the roscore by typing "hostname"

**Step3:** On the other computer type

Also set ROS_IP=http://[IP]

```
export ROS_MASTER_URI=http://[ENTER HOSTNAME HERE]:11311
```

**Step4:** Run launchers on separate computers from exercise 8

# DONE

YOU JUST USED THE POWER OF DISTRIBUTED COMPUTING IN ROS

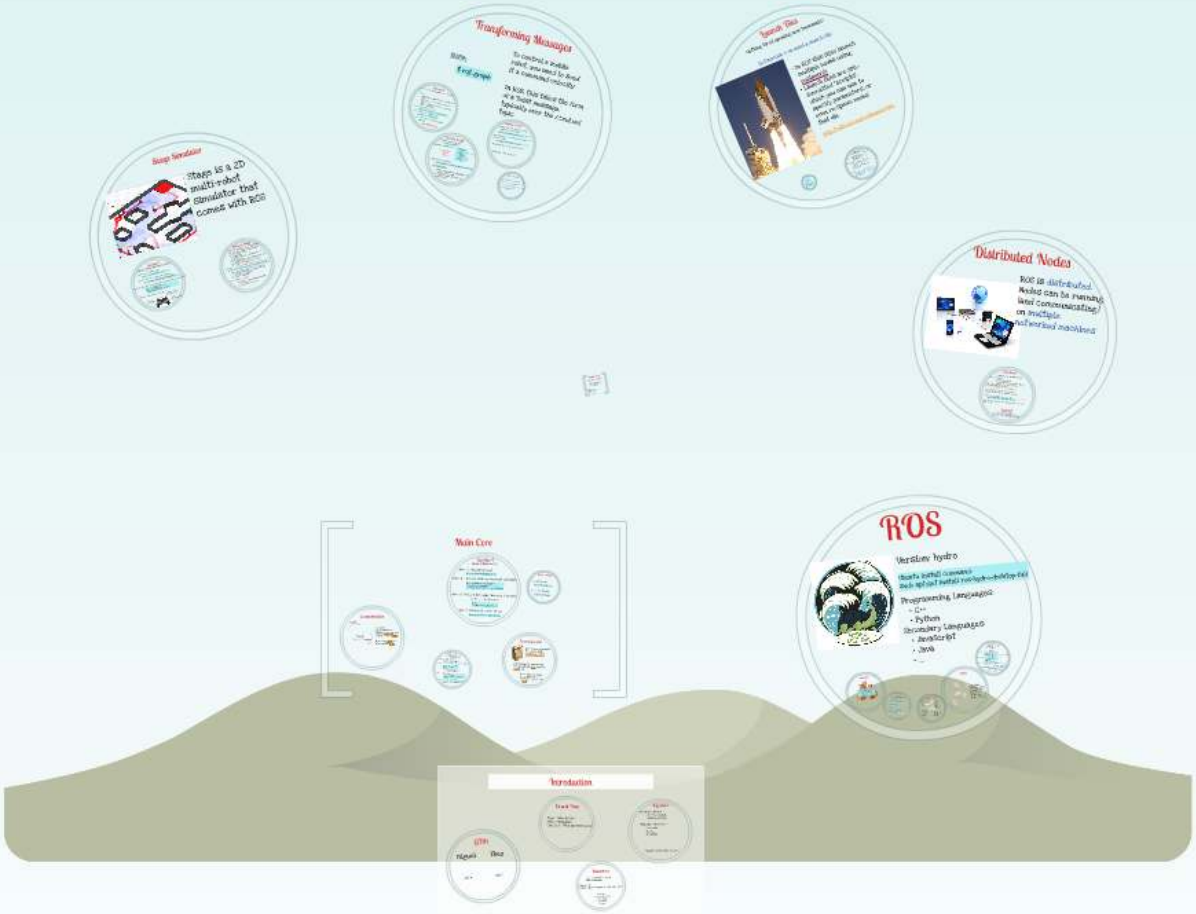
# Day 1 Done

Day 2 - Monday 18:00 - 20:00
Day 3 - Thursday 18:00 - 20:00

## PLEASE HELP US REARRANGE THE ROOM

```
def congrats:
    if robot_remotely_moved:
        print("Go Home!!! :)")
    else:
        print("Go Home :(")
```

# Intro To ROS


Transforming Messages
Stage
tf
Distributed Nodes
Main Core
ROS
Introduction