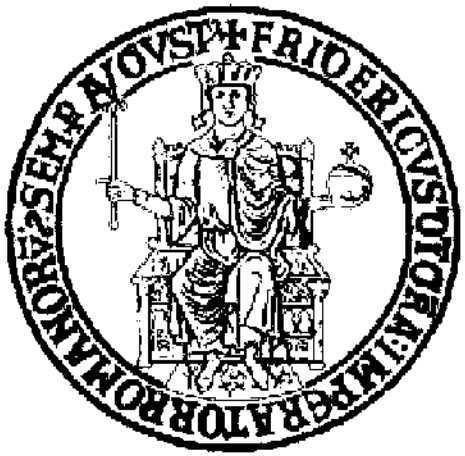


ROS.org

Robot **O**perating **S**ystem

Lezione 2

A cura di:
Jonathan Cacace

Outline

Lezione 1:

- Introduzione;
- …
- Primi passi.

Lezione 2:

- Messaggi in Ros;
- ROS Stage;
- Esempio di applicazione;
- RosAria.

Messaggi ROS

In ROS i messaggi sono utilizzati sia per richiedere servizi che per permettere la comunicazione tra i vari nodi (pubblicando su topic).

Un messaggio non è altro che una struttura dati. In ROS sono presenti i tipi primitivi standard (int, double, float…), gli array dei tipi primitivi, ed altre strutture più complesse.

ROS mette già a disposizione una serie di messaggi predefiniti, che permettono di gestire la maggior parte informazioni in ambito robotico (informazioni riguardo la posa del robot, gli output dei sensori, il movimento dei giunti di un robot,…). E' importante scegliere il messaggio giusto durante lo sviluppo dell'applicazione.

Messaggi ROS

In ROS i messaggi sono utilizzati sia per richiedere servizi che per permettere la comunicazione tra i vari nodi (pubblicando su topic).

Es.
Creare un messaggio contenente un dato di tipo float:

std_msgs/Float32 Message;
Message.data = 10;

Pubblicare il messaggio creato:
Publisher.publish(Message)

std_msgs/Float32 Message

File: std_msgs/Float32.msg

```
float32 data
```

Expanded Definition

```
float32 data
```

Messaggi ROS

In ROS i messaggi sono utilizzati sia per richiedere servizi che per permettere la comunicazione tra i vari nodi (pubblicando su topic).

Lista dei messaggi standard di ros:
http://ros.org/wiki/std_msgs

Messaggi ROS

Alcuni messaggi comuni in ROS sono:

geometry_msgs/PoseStamped: descrive la posizione di un oggetto nelle componenti (x,y,z) con quattro gradi di orientazione.

geometry_msgs/PoseStamped Message

File: geometry_msgs/PoseStamped.msg

```
# A Pose with reference coordinate frame and timestamp
Header header
Pose pose
```

Expanded Definition

```
Header header
   uint32 seq
   time stamp
   string frame_id
Pose pose
   geometry_msgs/Point position
      float64 x
      float64 y
      float64 z
   geometry_msgs/Quaternion orientation
      float64 x
      float64 y
      float64 z
      float64 w
```

Messaggi ROS

Alcuni messaggi comuni in ROS sono:

geometry_msgs/Twist: descrive le velocità che è possibile fornire ad un robot nelle componenti lineari (x,y,z) ed angolari (x,y,z)

geometry_msgs/Twist Message

File: geometry_msgs/Twist.msg

```
# This expresses velocity in free space broken into it's linear and
Vector3  linear
Vector3  angular
```

Expanded Definition

```
Vector3 linear
    float64 x
    float64 y
    float64 z
Vector3 angular
    float64 x
    float64 y
    float64 z
```

Messaggi ROS

Alcuni messaggi comuni in ROS sono:

sensor_msgs/Image: contiene i dati immagine. Il campo data del messaggio trasmette la matrice che contiene i valori dei pixel di un immagine.

sensor_msgs/Image Message

File: sensor_msgs/Image.msg

Expanded Definition

```
Header header
    uint32 seq
    time stamp
    string frame_id
uint32 height
uint32 width
string encoding
uint8 is_bigendian
uint32 step
uint8[] data
```

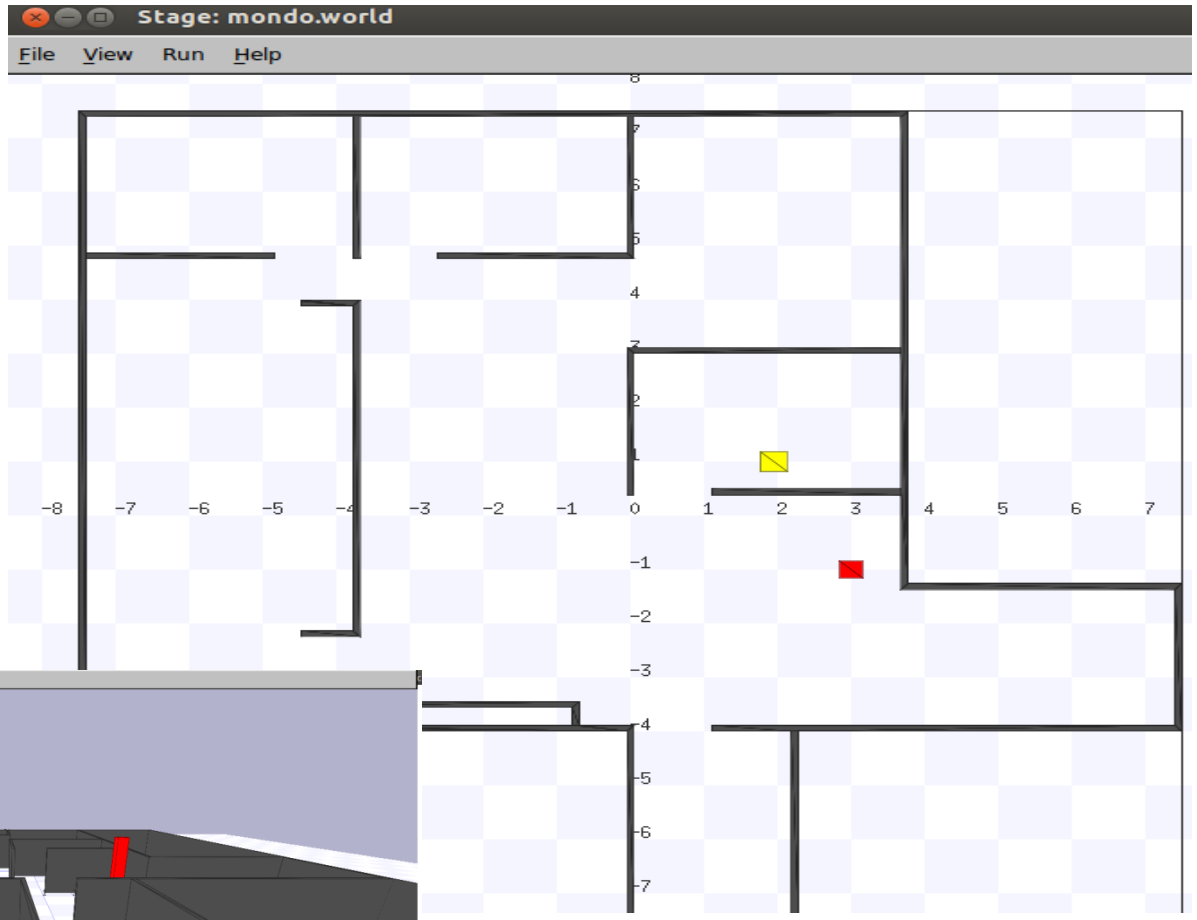
Stage

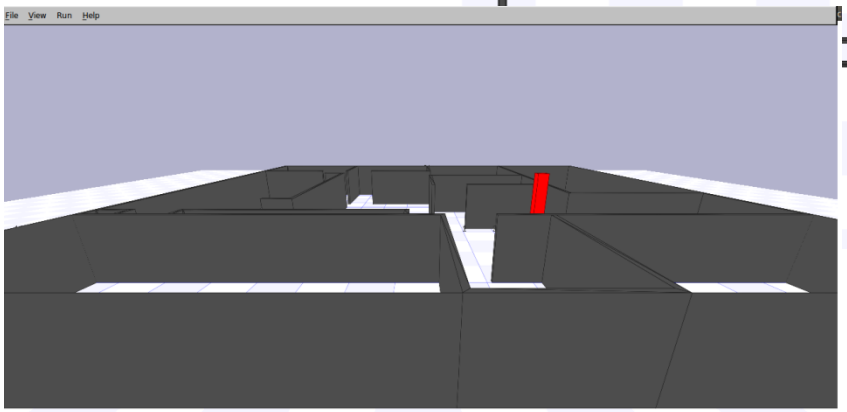
Nello sviluppo di un'applicazione robotica è fondamentale avere uno strumento che aiuta lo sviluppatore a testare il comportamento che avrebbe il robot immerso nel mondo.

Stage è un simulatore di robotica che fornisce un modello *virtuale* del mondo, popolato da robot, ostacoli e sensori.

Il mondo è composto di oggetti e, ogni oggetto è modellato attraverso un file di tipo "world", in cui il programmatore specifica la forma e la posizione degli oggetti.

Stage


Stage: mondo.world
File View Run Help
8
7
6
5
4
3
2
1
-8 -7 -6 -5 -4 -3 -2 -1 0 1 2 3 4 5 6 7
-1
-2
-3
-4
-5
-6
-7


File View Run Help

Configurazione del mondo: ambiente

```
# Configurazione della finestra utente
window (
        size [700.000 700.000]
        scale 41
        center [0 0]
)

# Configurazione del pianta dell'ambiente
define floorplan model (
        color "gray30"
        boundary 1
        gui_nose 0
        gui_grid 1
        gui_move 0
        gui_outline 0
)

# Creazione dell'ambiente
floorplan (
        bitmap "autolab1.png"
        size [15 15 1.5]
)
```

Configurazione della finestra con cui interagisce l'utente.
Size: in pixel definisce le dimensioni della finestra.

Configurazione del mondo: ambiente

```
# Configurazione della finestra utente
window (
        size [700.000 700.000]
        scale 41
        center [0 0]
)

# Configurazione del pianta dell'ambiente
define floorplan model (
        color "gray30"
        boundary 1
        gui_nose 0
        gui_grid 1
        gui_move 0
        gui_outline 0
)

# Creazione dell'ambiente
floorplan (
        bitmap "autolab1.png"
        size [15 15 1.5]
)
```

Parametri della mappa in cui operano i robot.

Configurazione del mondo: ambiente

```
# Configurazione della finestra utente
window (
        size [700.000 700.000]
        scale 41
        center [0 0]
)

# Configurazione del pianta dell'ambiente
define floorplan model (
        color "gray30"
        boundary 1
        gui_nose 0
        gui_grid 1
        gui_move 0
        gui_outline 0
)

# Creazione dell'ambiente
floorplan (
        bitmap "autolab1.png"
        size [15 15 1.5]
)
```

Creazione della mappa.
La Mappa viene importata attraverso un file immagine :“autolab1.png”

Configurazione del mondo: ambiente

```
# Configurazione di altri oggetti
# (ostacoli)
define puck1 model
(
        size [ 0.5 0.5 0.500]
        ranger_return 1
        obstacle_return 1
)
```

Configurazione di altri oggetti del mondo. E' possibile specificarne dimensione e comportamento

Ranger_return = 1 -> l'oggetto risponde al sensore laser
Obstacle_return = 1 -> l'oggetto viene riconosciuto fisicamente come un ostacolo

Configurazione del mondo: laser

```
# Sensore - Ranger (laser)
define topurg ranger
(
        sensor(
                # Min Max range per rispondere
                # agli ostacoli
                range [ 0.0 3.0 ]
                # Cono di visione del laser
                fov 30.25
                samples 1081
        )
        # Corpo del laser
        color "black"
        size [ 0.05 0.05 0.1 ]
)
```

Laser:
E' possibile impostare il range minimo e massimo di risposta agli oggetto, l'angolo di visuale (30° in questo caso) e la frequenza di campionamento del sensore.

Configurazione del mondo: laser

```
# Sensore - Ranger (laser)
define topurg ranger
(
        sensor(
                # Min Max range per rispondere
                # agli ostacoli
                range [ 0.0 3.0 ]
                # Cono di visione del laser
                fov 30.25
                samples 1081
        )
        # Corpo del laser
        color "black"
        size [ 0.05 0.05 0.1 ]
)
```

Laser:
Forma fisica del sensore da applicare al robot

Configurazione del mondo: robot

```
# Configurazione del robot
define erratic position
(
        # Dimensioni
        size [0.35 0.35 0.25]
        origin [-0.05 0 0 0]
        gui_nose 1
        # Controllo differenziale
        drive "diff"
        # Posizione del laser
        topurg(pose [ 0.050 0.000 0 0.000 ])

        # Configurazione della webcam
        camera
        (
                # Risoluzione
                resolution [ 160 120 ]
                range [ 0.2 8.0 ]
                fov [ 70.0 40.0 ]
                size [ 0.1 0.07 0.05 ]
                color "black"
        )
)
```

Dimensione del robot e posizionamento del suo centro.

Configurazione del mondo: robot

```
# Configurazione del robot
define erratic position
(
        # Dimensioni
        size [0.35 0.35 0.25]
        origin [-0.05 0 0 0]
        gui_nose 1
        # Controllo differenziale
        drive "diff"
        # Posizione del laser
        topurg(pose [ 0.050 0.000 0 0.000 ])

        # Configurazione della webcam
        camera
        (
                # Risoluzione
                resolution [ 160 120 ]
                range [ 0.2 8.0 ]
                fov [ 70.0 40.0 ]
                size [ 0.1 0.07 0.05 ]
                color "black"
        )
)
```

Inclusione del laser (definito nella slide precedente)

Configurazione del mondo: robot

```
# Configurazione del robot
define erratic position
(
        # Dimensioni
        size [0.35 0.35 0.25]
        origin [-0.05 0 0 0]
        gui_nose 1
        # Controllo differenziale
        drive "diff"
        # Posizione del laser
        topurg(pose [ 0.050 0.000 0 0.000 ])

        # Configurazione della webcam
        camera
        (
                # Risoluzione
                resolution [ 160 120 ]
                range [ 0.2 8.0 ]
                fov [ 70.0 40.0 ]
                size [ 0.1 0.07 0.05 ]
                color "black"
        )
)
```

Configurazione della webcam: Risoluzione (in bitmap)

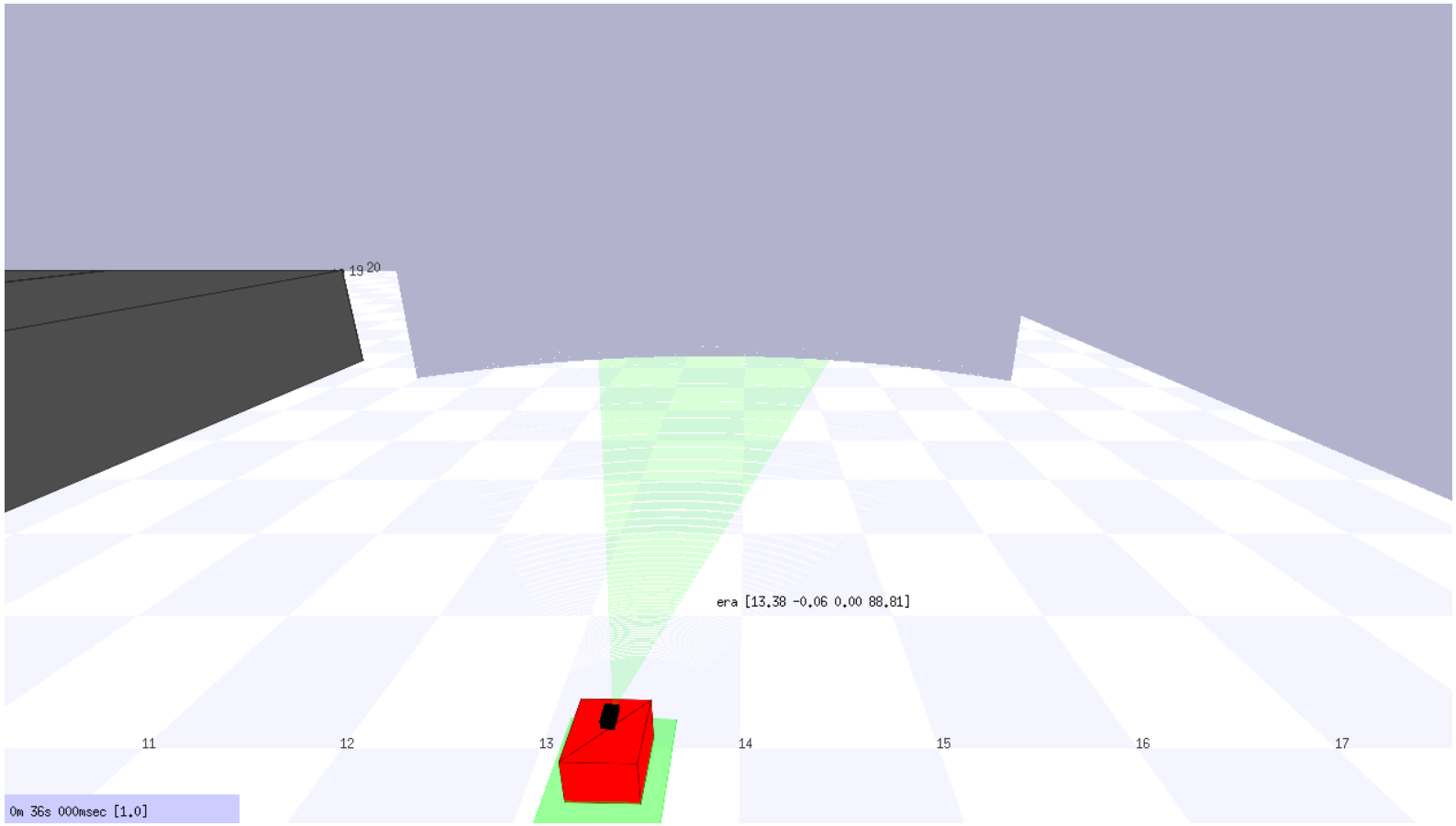
Come per il larser, anche in questo caso è possibile impostare il cono di visione del sensore ed il range di risposta.

Configurazione del mondo: mondo

```
# Inclusione degli oggetti nel mondo
puck1( pose [ 3 -1 0 0 ] color "yellow" )
erratic( pose [ 2 1 0 0 ] name "era" color "red")
erratic( pose [ 1 1 0 0 ] name "era2" color "red")
erratic( pose [ 2 2 0 0 ] name "era3" color "red")
```

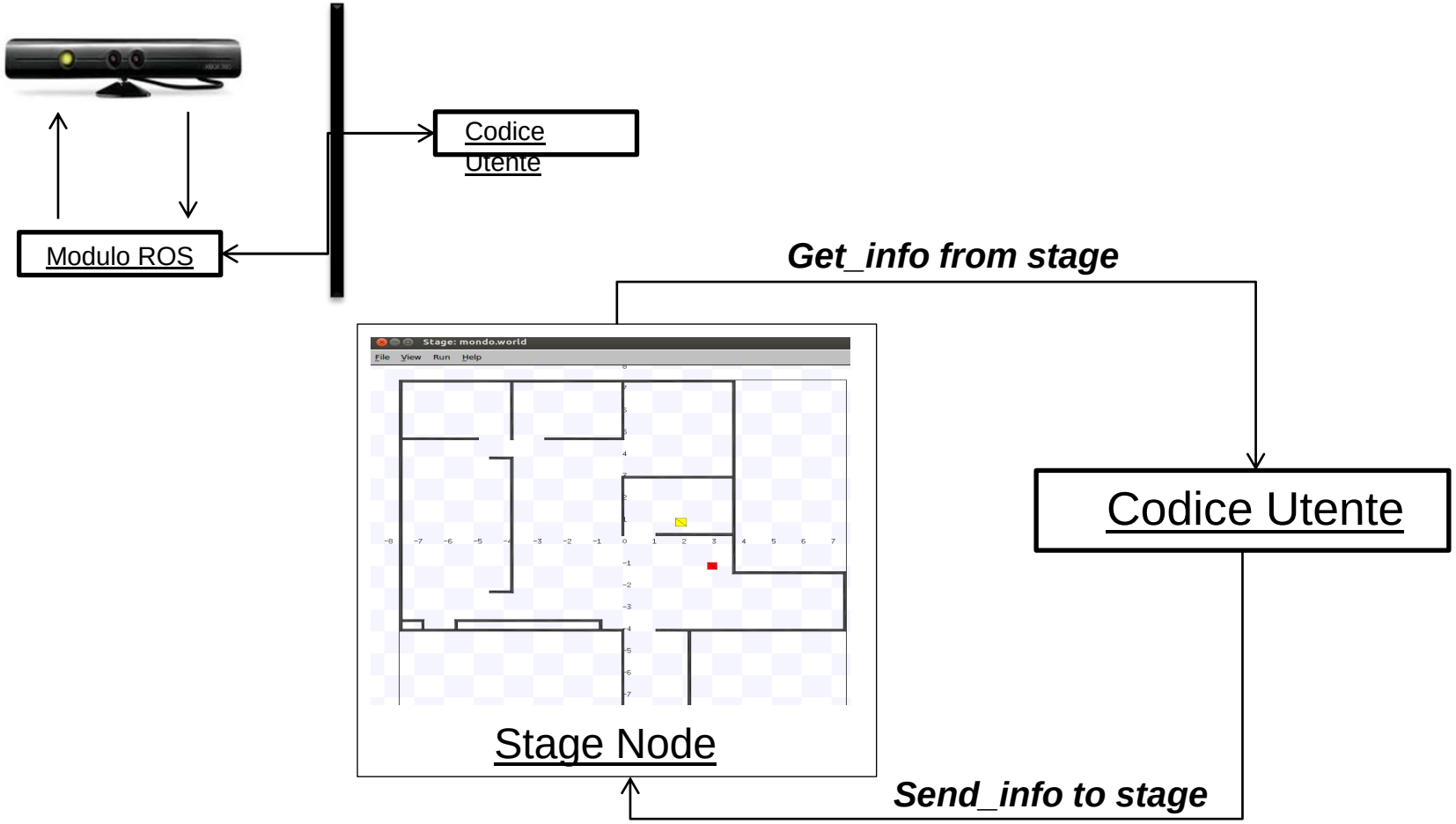
Nome_robot(posizione nel mondo, “nome nel mondo”, colore)

Configurazione del mondo: robot



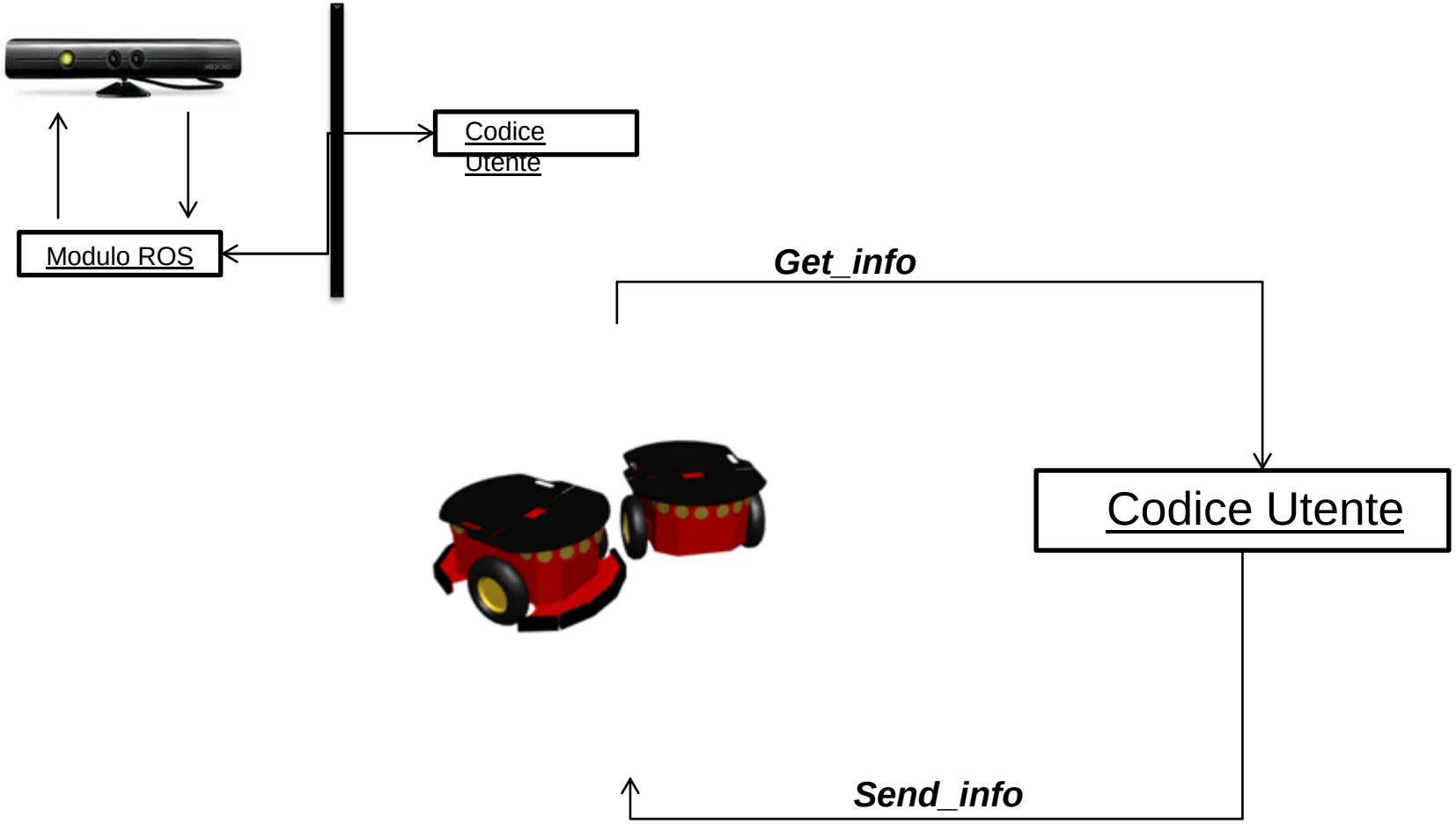
Stage

Ricordando il concetto di modularità introdotto nella lezione precedente:



Stage

Ricordando il concetto di modularità introdotto nella lezione precedente:



Stage

Quindi, per utilizzare Stage con ROS è necessario:

1. Installare la versione di Stage per ROS (già presente nell'ultima release di ros);
2. Creare il mondo e il robot, attraverso il file .world;
3. Avviare il nodo ROS per Stage, indicando il file .world;
4. Utilizzare i topic messi a disposizione dal nodo ROS per interagire con il mondo simulato.

Comunicare con stage

Per comunicare con Stage e ricevere informazioni sullo stato del robot e del mondo è necessario utilizzare i topic. Per accedere alla lista dei topic attivi, dopo aver avviato il nodo stage è possibile utilizzare il comando rostopic list:

```
pacifica@pacifica-Dell-System-XPS-L502X:~/fuerte_workspace/tutorial/Stage$ rostopic list
/base_pose_ground_truth
/base_scan
/clock
/cmd_vel
/image
/odom
/rosout
/rosout_agg
/tf
```

Comunicare con stage

Topic: /base_pose_gound_truth

```
pacifica@pacifica-Dell-System-XPS-L502X:~/fuerte_workspace/tutorial/Stage$ rostopic info /base_pose_ground_truth
Type: nav_msgs/Odometry

Publishers:
 * /stageros (http://pacifica-Dell-System-XPS-L502X:45890/)

Subscribers: None
```

Comunicare con stage

```
Header header
   uint32 seq
   time stamp
   string frame_id
string child_frame_id
geometry_msgs/PoseWithCovariance pose
   geometry_msgs/Pose pose
      geometry_msgs/Point position
         float64 x
         float64 y
         float64 z
      geometry_msgs/Quaternion orientation
         float64 x
         float64 y
         float64 z
         float64 w
   float64[36] covariance
geometry_msgs/TwistWithCovariance twist
   geometry_msgs/Twist twist
      geometry_msgs/Vector3 linear
         float64 x
         float64 y
         float64 z
      geometry_msgs/Vector3 angular
         float64 x
         float64 y
         float64 z
   float64[36] covariance
```

Topic: /base_pose_gound_truth

Comunicare con stage

Topic: /base_scan

```
pacifica@pacifica-Dell-System-XPS-L502X:~/fuerte_workspace/tutorial/Stage$ rostopic info /base_scan
Type: sensor_msgs/LaserScan

Publishers:
 * /stageros (http://pacifica-Dell-System-XPS-L502X:45890/)

Subscribers: None
```

Comunicare con stage

Topic: /base_scan

sensor_msgs/LaserScan Message

```
Header header
    uint32 seq
    time stamp
    string frame_id
float32 angle_min
float32 angle_max
float32 angle_increment
float32 time_increment
float32 scan_time
float32 range_min
float32 range_max
float32[] ranges
float32[] intensities
```

Comunicare con stage

Topic: /odom

```
pacifica@pacifica-Dell-System-XPS-L502X:~/fuerte_workspace/tutorial/Stage$ rostopic info /odom
Type: nav_msgs/Odometry

Publishers:
 * /stageros (http://pacifica-Dell-System-XPS-L502X:45890/)

Subscribers: None
```

Comunicare con stage

Topic: /cmd_vel

geometry_msgs/Twist.msg

```
Vector3 linear
    float64 x
    float64 y
    float64 z
Vector3 angular
    float64 x
    float64 y
    float64 z
```

NB.

A differenza degli altri topic mostrati, questo non serve al programmatore per decifrare lo stato del mondo e del robot, ma per controllare il robot a basso livello!

Comunicare con stage

Topic: /image

```
pacifica@pacifica-Dell-System-XPS-L502X:~/fuerte_workspace/tutorial/Stage$ rostopic info /image
Type: sensor_msgs/Image

Publishers:
 * /stageros (http://pacifica-Dell-System-XPS-L502X:45890/)

Subscribers: None
```

Multirobot??

Nel caso di un sistema multirobot, verranno creati più topic a seconda del robot.

```
pacifica@pacifica-Dell-System-XPS-L502X:~/fuerte_workspace/tutorial/Stage$ rostopic list
/clock
/robot_0/base_pose_ground_truth
/robot_0/base_scan
/robot_0/cmd_vel
/robot_0/image
/robot_0/odom
/robot_1/base_pose_ground_truth
/robot_1/base_scan
/robot_1/cmd_vel
/robot_1/image
/robot_1/odom
/robot_2/base_pose_ground_truth
/robot_2/base_scan
/robot_2/cmd_vel
/robot_2/image
/robot_2/odom
/rosout
/rosout_agg
/tf
```

Controllo del robot

Dopo aver preparato l'ambiente e aver avviato il nodo ROS che gestisce Stage è possibile controllare i robot presenti.

Per controllo del robot si intende, in questo caso, la gestione delle velocità degli attuatori che ne permettono il movimento. Per far spostare uno dei robot creati è quindi necessario pubblicare queste velocità sul relativo topic.

E' possibile utilizzare uno dei comandi messo a disposizione da ROS per testare il comportamento di un robot dopo l'invio delle velocità:

$ rostopic pub "nome topic" "tipo di messaggio " "valore da pubblicare"

Controllo del robot

E’ possibile utilizzare uno dei comandi messo a disposizione da ROS per testare il comportamento di un robot dopo l’invio delle velocità:

$ rostopic pub “nome topic” “tipo di messaggio “ “valore da pubblicare”

```
rostopic pub /robot_1/cmd_vel geometry_msgs/Twist -r 10000 '[0.1,0,0]' '[0, 0.1, 0.4]'
```

Ovviamente, nel caso il robot voglia essere controllato da codice, è necessario creare un oggetto publisher che scriva sul topic relativo.

Webcam

Per utilizzare le immagini messa a disposizione dalla webcam necessario registrarsi al topic “/image” del robot desiderato.

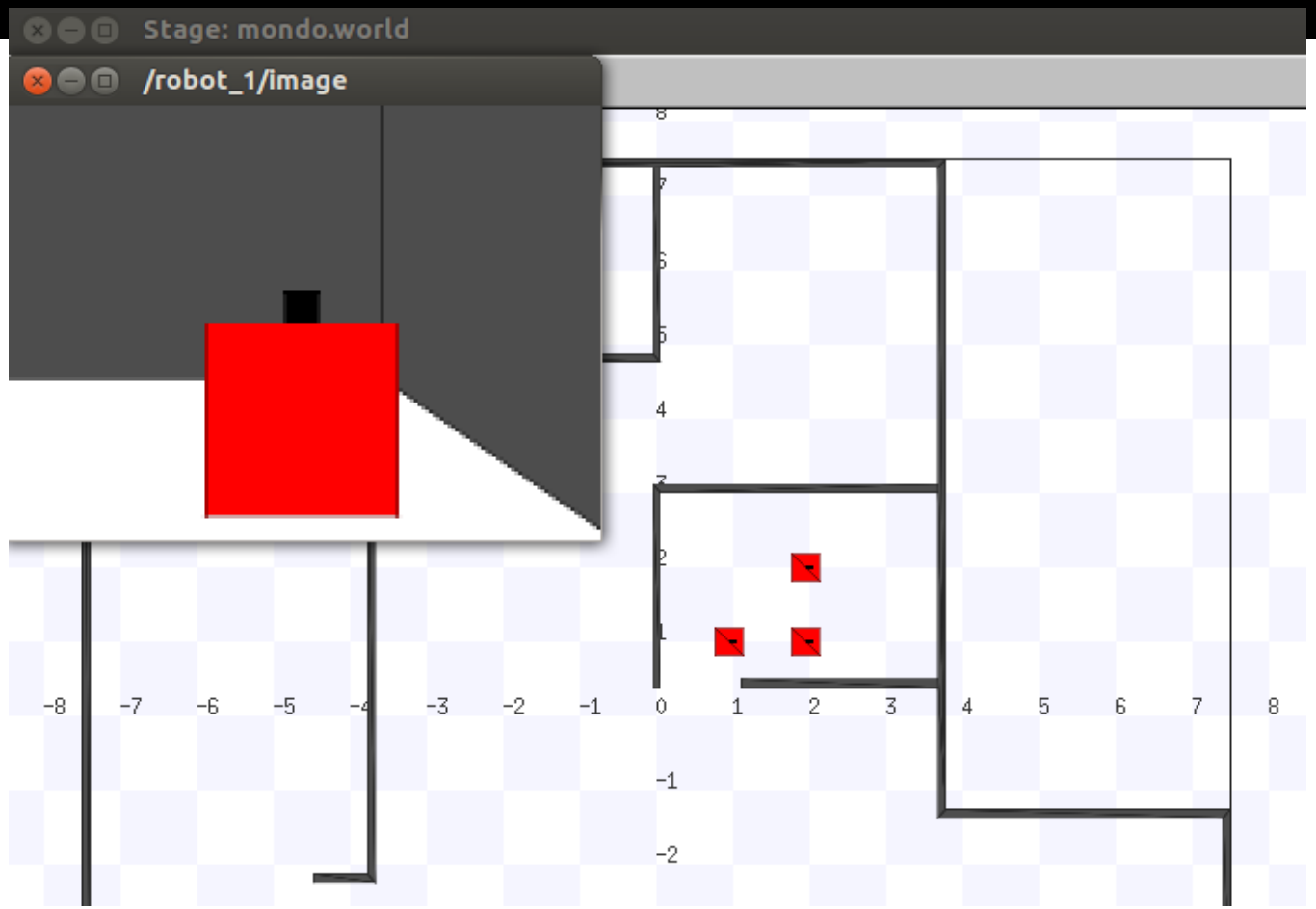
Attraverso questo topic l’utente riceve un flusso di immagini che devono essere elaborate per reperire le informazioni necessarie.

E’ possibile visualizzare l’immagine pubblicata dal robot attraverso il nodo *image_view* del pacchetto *image_view* di ROS:

```
Stage$ rosrun image_view image_view image:=robot_1/image
```

Webcam

```
Stage$ rosrun image_view image_view image:=robot_1/image
```



Webcam

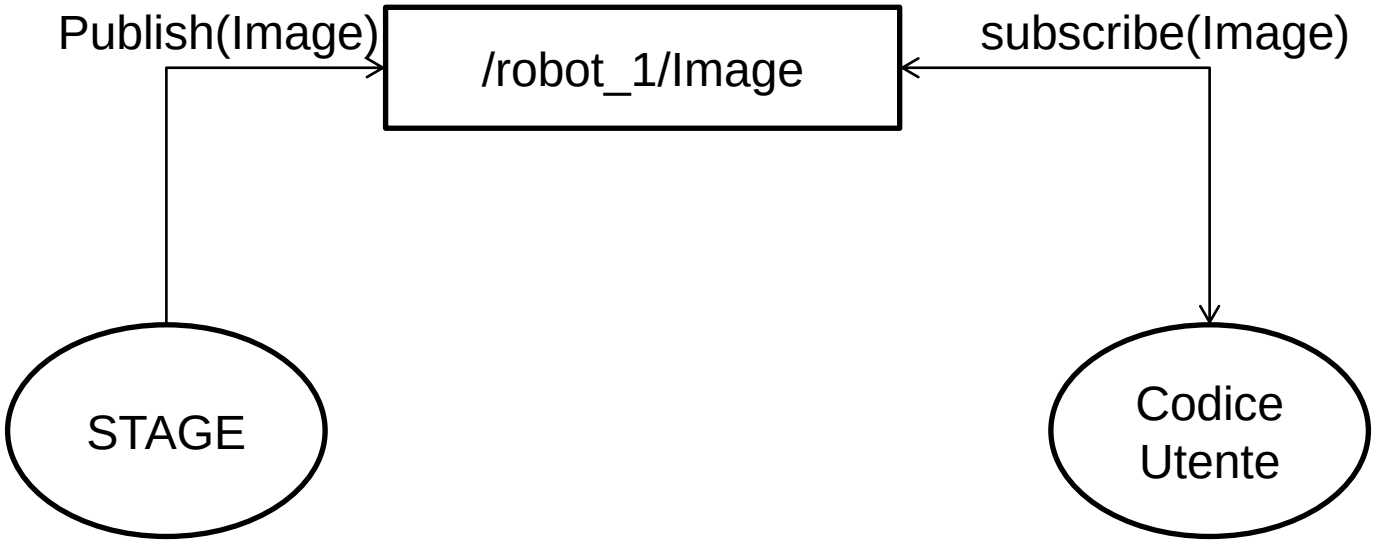
La webcam messa a disposizione non fornisce informazioni sulla distanza degli oggetti catturati, ed inoltre, a causa della mancanza di un’illuminazione reale può essere difficile estrarre molte informazioni dalle immagini.

Demo

In questa demo, le immagini della webcam vengono utilizzate il riconoscimento di *blob* di colore rosso. Ogni blob viene cerchiato e numerato.

Demo

In questa demo le immagini della webcam, presenti sul topic /robot_1/image, sono state utilizzate per effettuare un'elaborazione delle immagini.



Pioneer 3Dx

Piattaforma robotica personalizzabile.

Modello di base:

- pose
- sonar

Pioneer 3Dx

La porta seriale presente sul corpo del robot mette a disposizione la possibilità di controllare il robot in velocità e di ricevere informazioni sull'odometria.

RosAria

RosAria è un nodo ros che offre la possibilità di controllare le piattaforme robotiche più diffuse.

Avviando il nodo RosAria, è possibile utilizzare i topic per l'accesso ai sensori.

Es.

$ rosrun ROSARIA rosaria [port]
$ rorsun ROSARIA rosaria /dev/ttyUSB0

RosAria

Avviando RosAria saranno disponibili i seguenti topic:

In lettura:

- pose (nav_msgs/Odometry);
- bumper_state (ROSARIA/BumperState);
- sonar (sensor_msgs/PointCloud).

In scrittura:

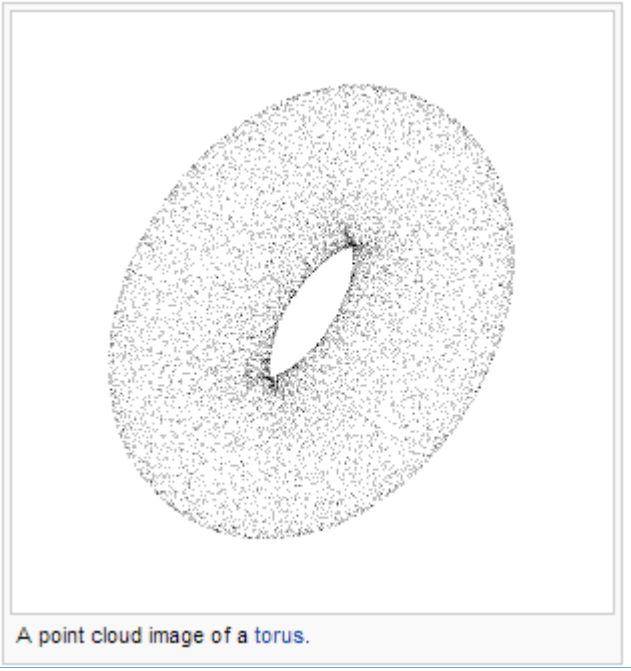
cmd_vel (geometry_msgs/Twist)

RosAria

sonar (sensor_msgs/PointCloud).

PCL: Point Cloud Library

E’ un framework per il processing di “nuvole di punti”.



A point cloud image of a torus.

Bag File

Un bag file è un formato in ROS per conservare il contenuto di messaggi. Per esempio, nel caso il programmatore stia lavorando con dati video, può registrare una sessione di dati, e salvarli nel bag file. A questo punto per sviluppare o testare il proprio codice non deve necessariamente avviare la webcam, ma semplicemente avviare il contenuto del bag file.

Es.

$ rosbag record [nome_topic] [Tipo di messaggio]

$ rosbag record /image sensor_msgs/Image

Bag File

Un bag file è un formato in ROS per conservare il contenuto di messaggi. Per esempio, nel caso il programmatore stia lavorando con dati video, può registrare una sessione di dati, e salvarli nel bag file. A questo punto per sviluppare o testare il proprio codice non deve necessariamente avviare la webcam, ma semplicemente avviare il contenuto del bag file.

Es.

$ rosbag play bagfile.bag

Wiki

http://www.ros.org/wiki/

http://www.ros.org/wiki/stage

http://playerstage.sourceforge.net/

http://www.ros.org/wiki/rosbag

http://pointclouds.org/