BLG456E
Robotics
ROS messages

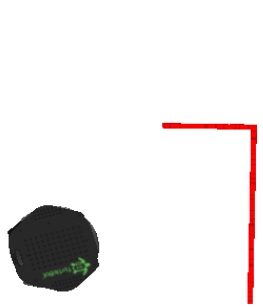
Lecture Contents:

Callback functions.
Node set-up.
Sending motor command messages.
Receiving laser scan messages.
More ROS tools.

Lecturer:	Damien Jade Duff
Email:	djduff@itu.edu.tr
Office:	EEBF 2316
Schedule:	http://djduff.net/my-schedule

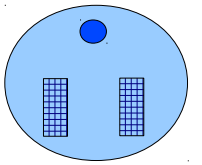
Turtlebot ROS control example

- Subscribe to laser scan messages (callback).
- Publish movement as twists (2D constrained).



Kinect depth camera abstracted as laser scanner

Two-wheel differential drive robot with non-actuated castor wheel



`Callback functions`

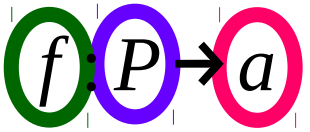
~ A function that you write that you send to other code that calls it. ~

- General concept in programming.

- Used in:
  - **GUIs**: `do-on-mouse-click`, `do-on-mouse-move`, `do-on-redraw`, etc.
  - **I/O handling**: `do-on-data-received`, `do-on-interrupt`, etc.
- …

Callback functions in ROS

- Called on input (subscribed message).
- Can provide an output (publish message).
- Can facilitate **behaviours**:



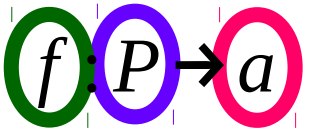
Callback **Messages**

A laser scan callback function

```
void scan_cb(const sensor_msgs::LaserScan::ConstPtr &scan){
   geometry_msgs::Twist cmd;
    //process scan to set up cmd
   pub_.publish(cmd);
}
```

Callback functions in ROS

- Called on input (subscribed message).
- Can provide an output (publish message).
- Can facilitate **behaviours**:



Callback **Messages**

A laser scan callback function

```
def scan_cb(data):
    motor_command = Twist()
     //process scan to set up cmd
    pub.publish(cmd);
}
```

Reactive Obstacle Avoidance & Wandering

- Reactive vs deliberative.
  - Reacting vs planning.
- Behavioural vs cognitive robotics.
  - Acting vs thinking.

Assignment 1

$$f : P^* \rightarrow a$$

abstraction

$$f : P \rightarrow a$$

BLG456E
Robotics
ROS messages

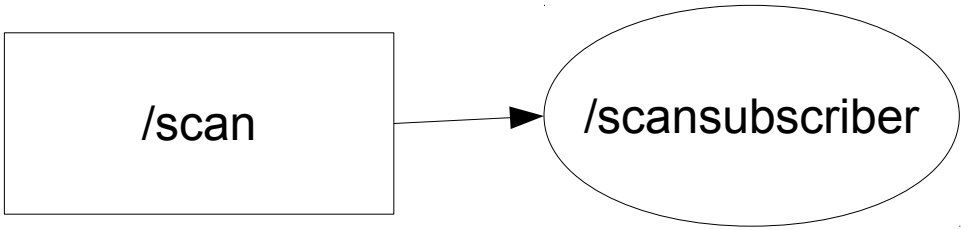
Lecture Contents:

Callback functions.
Node set-up.
Sending motor command messages.
Receiving laser scan messages.
More ROS tools.

Lecturer:	Damien Jade Duff
Email:	djduff@itu.edu.tr
Office:	EEBF 2316
Schedule:	http://djduff.net/my-schedule

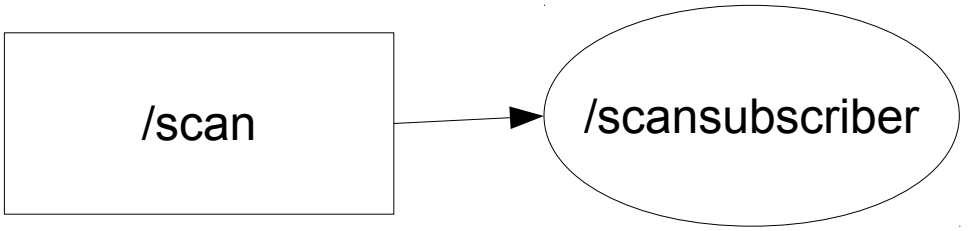
Setting up your node: subscribe to laser scan

```
#include "ros/ros.h"
#include "sensor_msgs/LaserScan.h>"
ros::Subscriber sub_;
int main(int argc,char **argv){
    ros::init(argc, argv, "scansubscriber");
    ros::NodeHandle n;
    sub_ = n.subscribe("/scan", 1000, scan_cb);
    ros::spin();
}
```



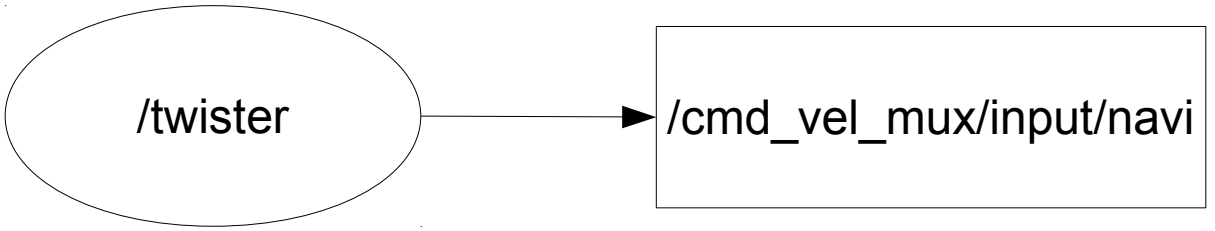
Setting up your node: subscribe to laser scan

```
import rospy
from sensor_msgs.msg import LaserScan
global sub;
if __name__ == '__main__':
    rospy.init_node('scansubscriber')
    sub = rospy.Subscriber("/scan", LaserScan, scan_cb,
          queue_size = 1000)
    rospy.spin()
}
```



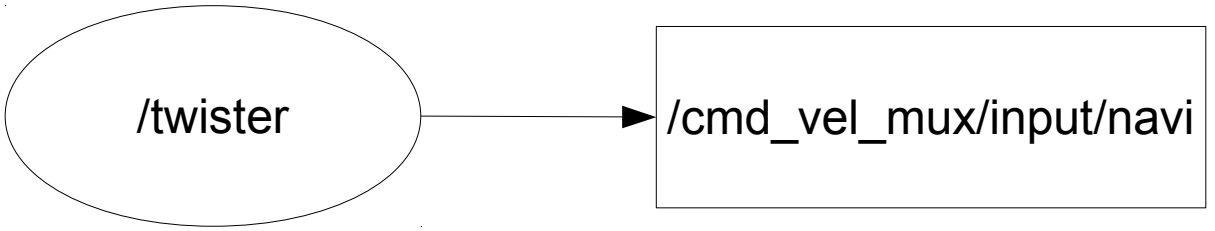
Setting up your node: publish to twists

```
#include "ros/ros.h"
#include "geometry_msgs/Twist.h"
ros::Publisher pub_;
geometry_msgs::Twist cmd;
int main(int argc, char **argv){
   ros::init(argc, argv, "twister");
   ros::NodeHandle n;
   pub_ =
n.advertise<geometry_msgs::Twist>("/cmd_vel_mux/input/navi", 100);
   ros::spin();
}
```



Setting up your node: publish to twists

```
import rospy
from geometry_msgs.msg import Twist
global pub
global cmd
if __name__ == '__main__':
   ros::init(argc, argv, "twister");
   ros::NodeHandle n;
   pub = rospy.Publisher("/cmd_vel_mux/input/navi", Twist,
         queue_size = 100);
   rospy.spin()
}
```



Twist messages & data structure

- **geometry_msgs/Twist -** 6 numbers:
  - 3D linear velocity (of robot).
  - 3D angular velocity around an axis.
- Simpler in 2D:
  - Linear velocity is in XY plane.
  - Angular velocity is around z axis.

Question: What are the 6 numbers in a twist?

Question: Which of these are held at zero in the 2D case?

Contents of a twist message in ROS

A **geometry_msgs/Twist** message in ROS:
http://docs.ros.org/api/geometry_msgs/html/msg/Twist.html

```
geometry_msgs/Vector3 linear
geometry_msgs/Vector3 angular
```

A **geometry_msgs/Vector3** message in ROS:
http://docs.ros.org/api/geometry_msgs/html/msg/Vector3.html

```
float64 x
float64 y
float64 z
```

Try running:

```
rosmsg show geometry_msgs/Twist
```

Or use an IDE with code completion (e.g. kdevelop).

Accessing a Twist message in ROS-CPP

```
geometry_msgs::Twist vel;
vel.linear.x = -2.0;
vel.angular.z = 3.0;
```

Only 2 numbers for the 2D case.

This is in robot's **local frame of reference**.

Exercise: What should the value of `vel.linear.z` be for a robot constrained to the plane?

Accessing a Twist message in ROSPy

```
vel = Twist()
vel.linear.x = -2.0
vel.angular.z = 3.0
```

Only 2 numbers for the 2D case.

This is in robot's **local frame of reference**.

Exercise: What should the value of `vel.linear.z` be for a robot constrained to the plane?

Lookahead: Coordinate transformations

- In mobile robot kinematics we need to *transform* **poses** and **velocities** between *reference frames*.
- With multi-link arms, several *transforms* need to be **chained**.

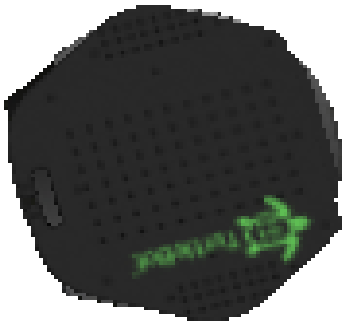
LaserScan message

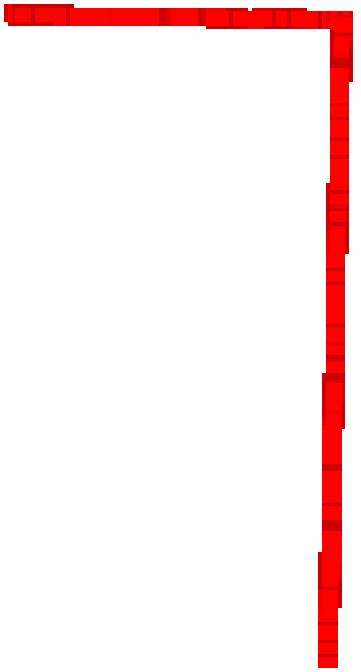
From http://docs.ros.org/api/sensor_msgs/html/msg/LaserScan.html

Or run >

```
rosmsg show sensor_msgs/LaserScan
```

```
std_msgs/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
```

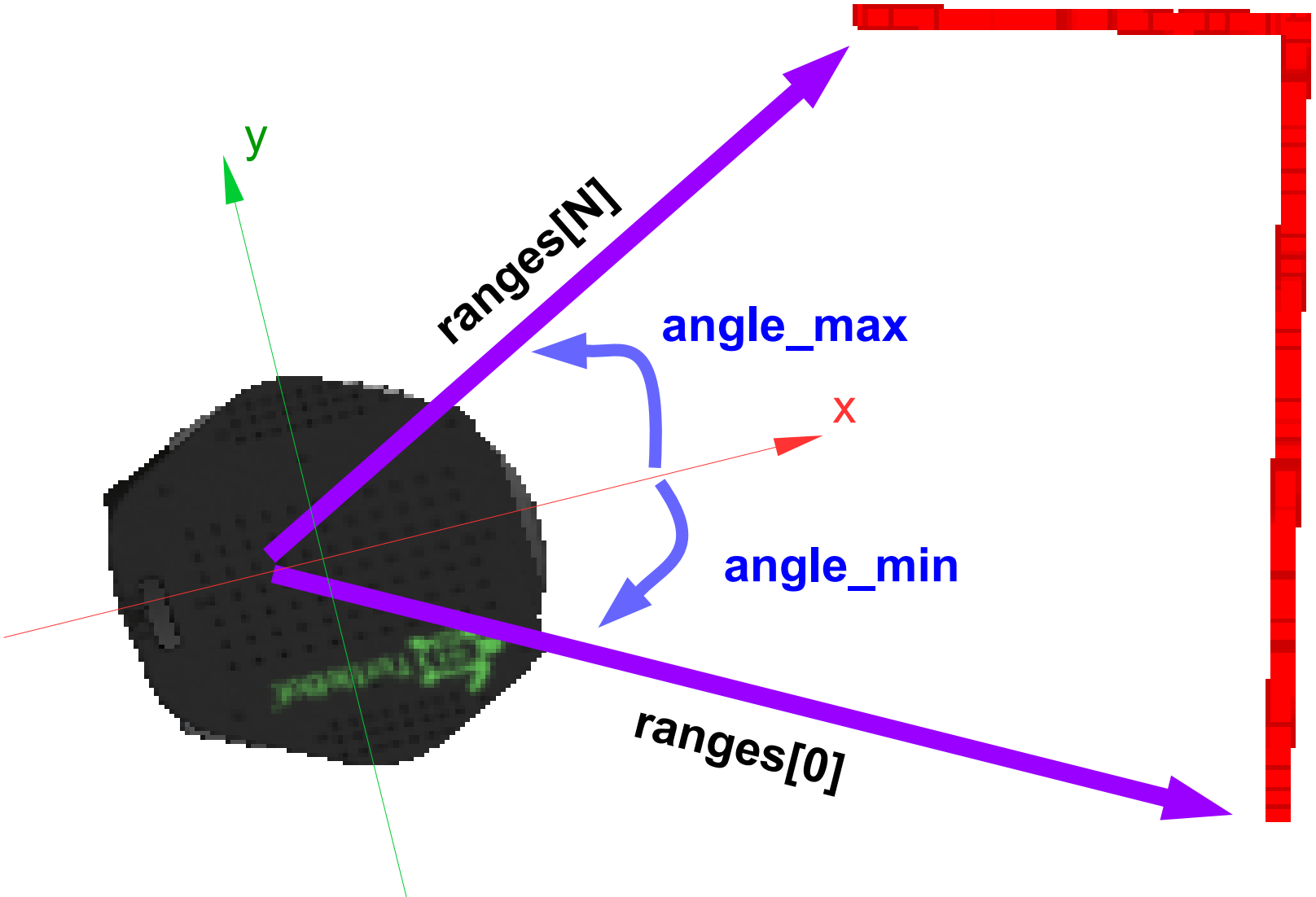




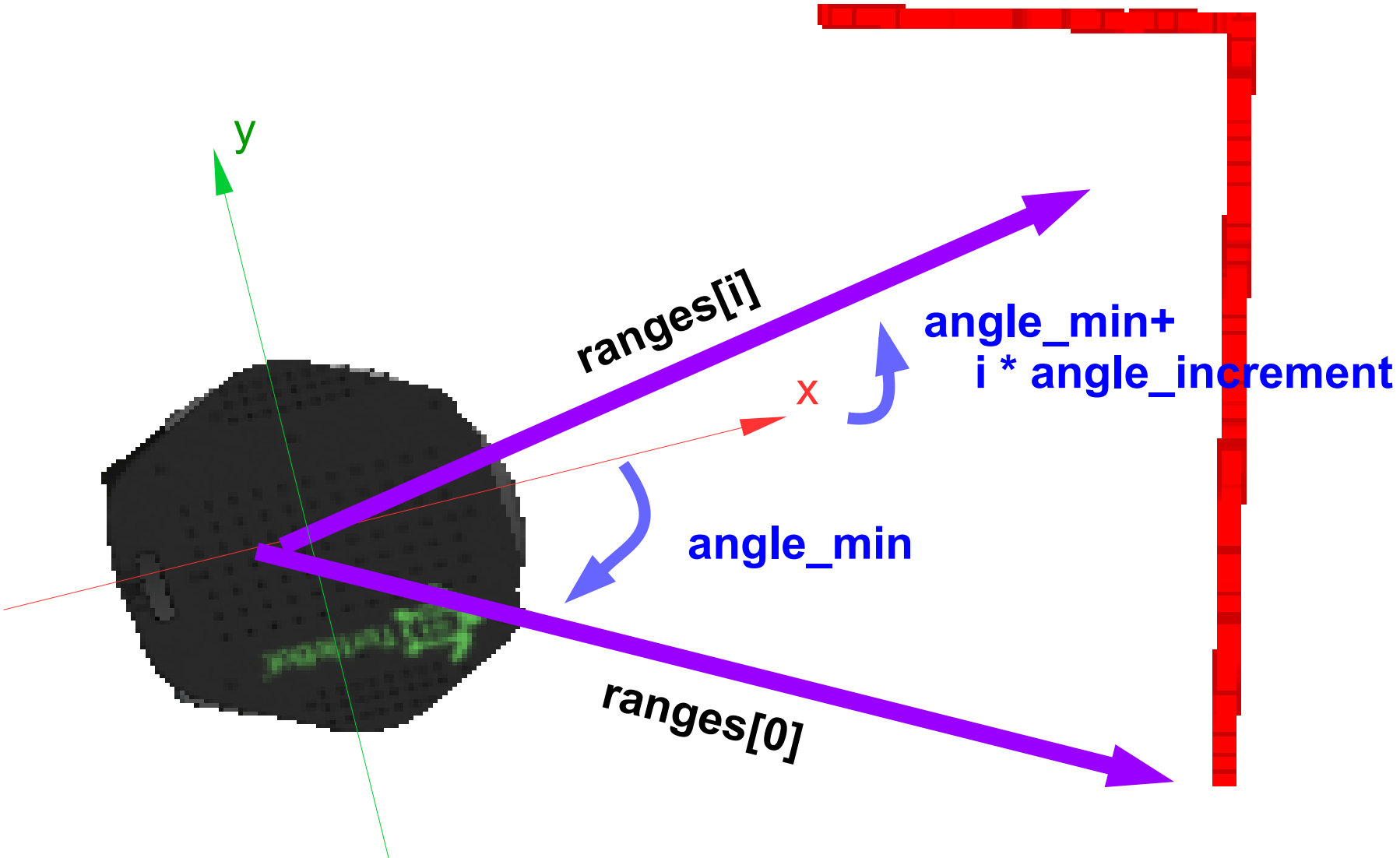
Accessing laser scan data
Approach 1: Hacker

- Determine data-type of **`ranges`**.
- Determine length of **`ranges`**.
- Determine contents of **ranges** under different situations.

Accessing laser scan data
Approach 2: Geometry



Accessing laser scan data
Approach 2: Geometry



BLG456E
Robotics
ROS messages

Lecture Contents:

Callback functions.
Node set-up.
Sending motor command messages.
Receiving laser scan messages.
More ROS tools.

Lecturer:	Damien Jade Duff
Email:	djduff@itu.edu.tr
Office:	EEBF 2316
Schedule:	http://djduff.net/my-schedule

More ROS tools: Command line tools

- Find out what nodes exist:

  ```
  rosnode list
  ```

- Find out information on a node:

  ```
  rosnode info /robot_state_publisher
  ```

- Find out what topics exist:

  ```
  rostopic list
  ```

- Find out information on a topic:

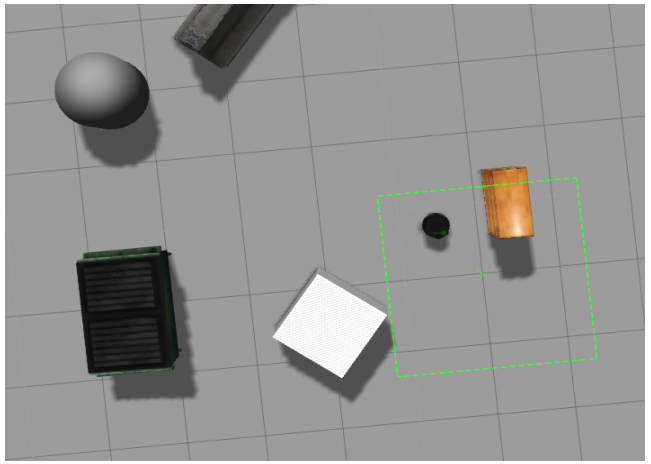
  ```
  rostopic info /odom
  ```

- Show what is being published over a topic:

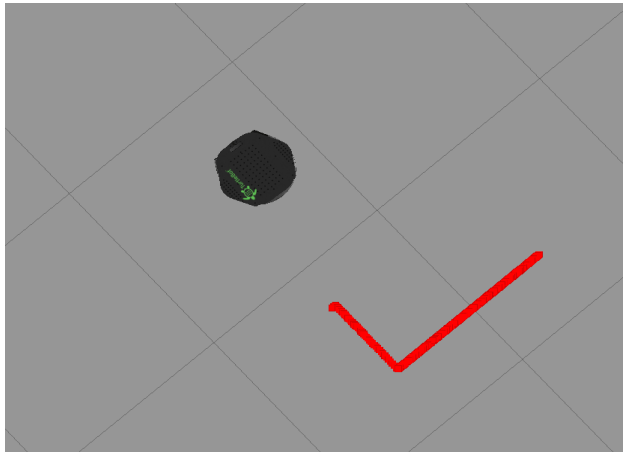
  ```
  rostopic echo /odom
  ```

More ROS tools: *rviz*: visualise the robot's world

GAZEBO



RVIZ



- From *robot*'s perspective

  (vs. Gazebo: full knowledge).
- Run *rviz*:

  `rosrun rviz rviz`
- Visualise robot:
  - Add... RobotModel
- Visualise laser scan:
  - Add... LaserScan
  - Choose topic: `/base_scan/scan`
- Visualise coordinate frames:
  - Add... TF

Readings

- Official ROS Tutorials.
  - Installing & Configurating your ROS Environment
  - Navigating the ROS Filesystem
  - Creating a ROS Package
  - Building a ROS Package
  - Understanding ROS Nodes
  - Understanding ROS Topics
  - Writing a Simple Publisher & Subscriber (C++)
  - Examining the Simple Publisher & Subscriber
  - Using rqt_console & roslaunch