

iRobot Create Setup with ROS and Implement Odometric Motion Model

Welcome

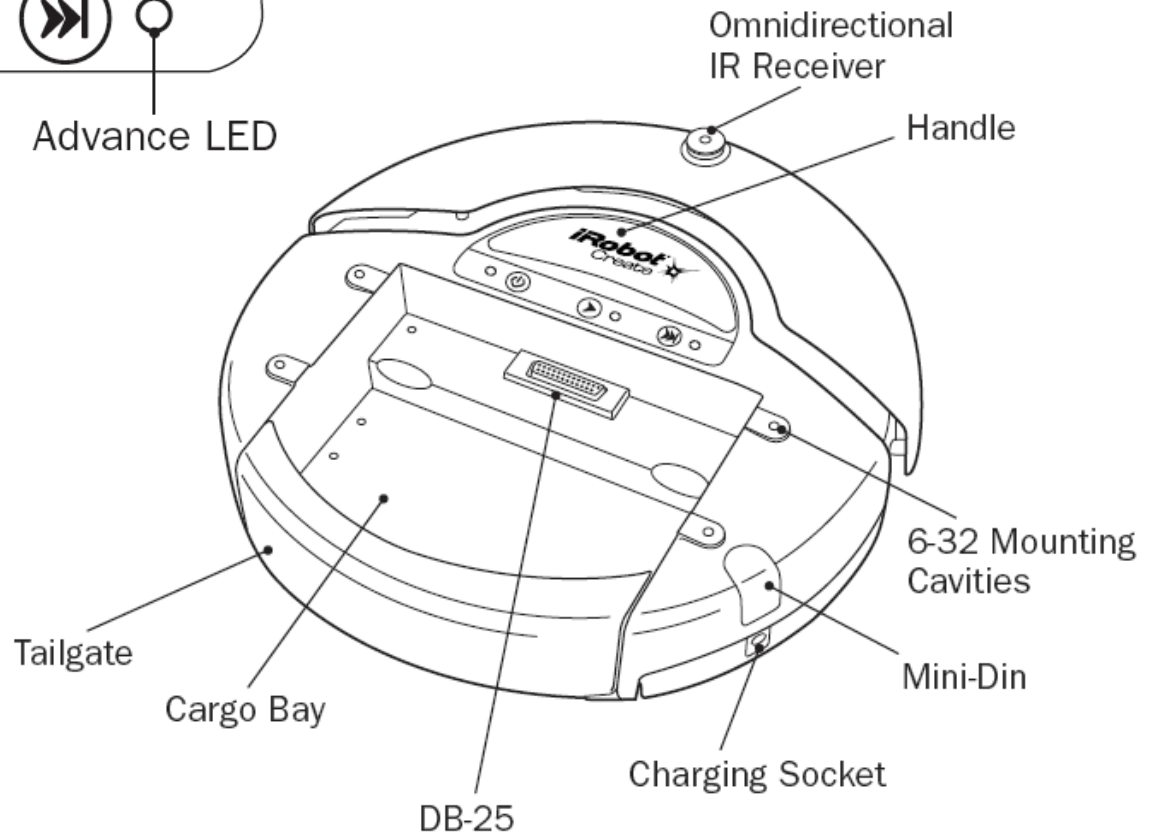
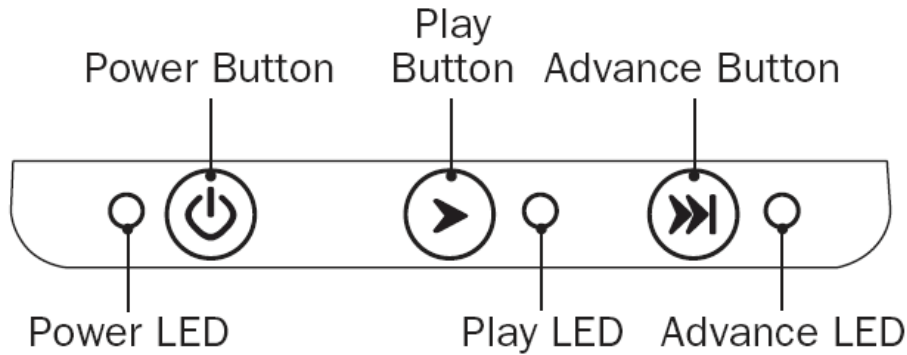
Lab 4

Dr. Ahmad Kamal Nasir

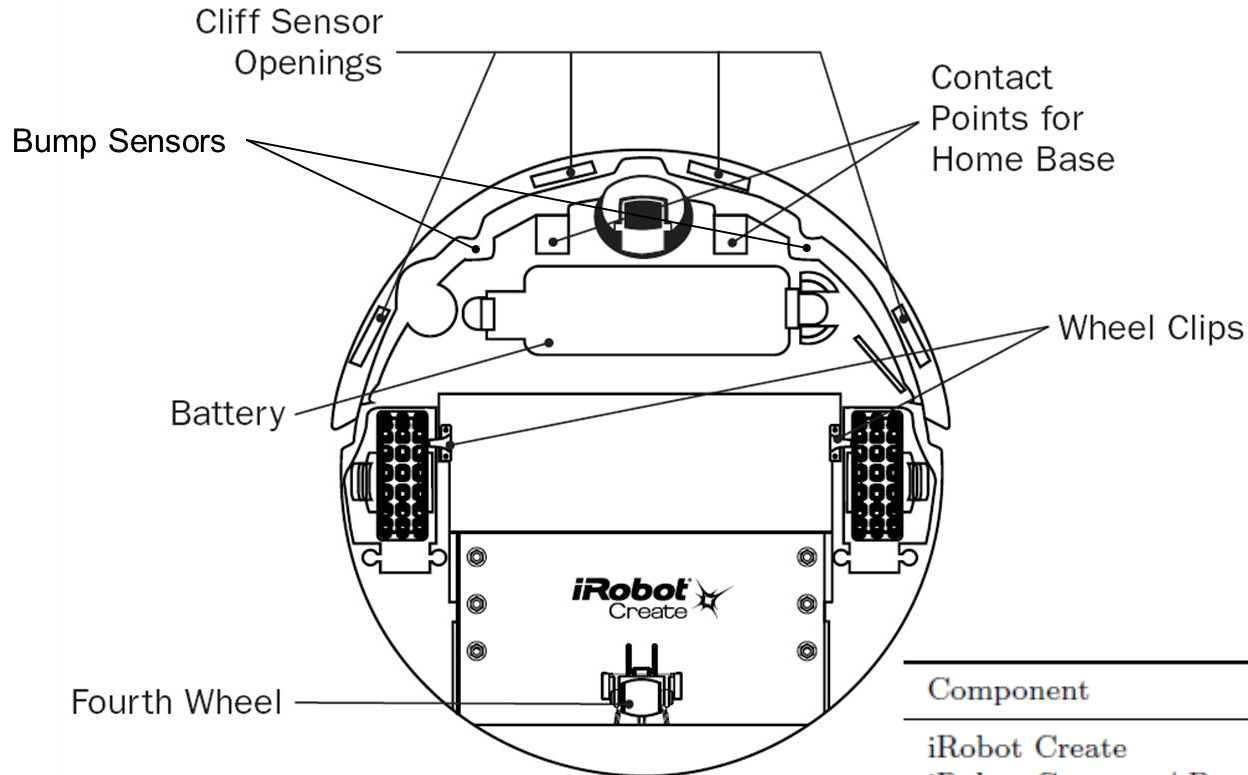
Today's Objectives

- Introduction to iRobot-Create
 - Hardware
 - Communication
- ROS with iRobot-Create Hardware
 - ROS driver nodes
 - Teleop Keyboard/Joystick
- ROS with iRobot-Create Gazebo Model
 - Odometry
 - Setting up wheel slippage and acceleration effects

iRobot Create - Hardware

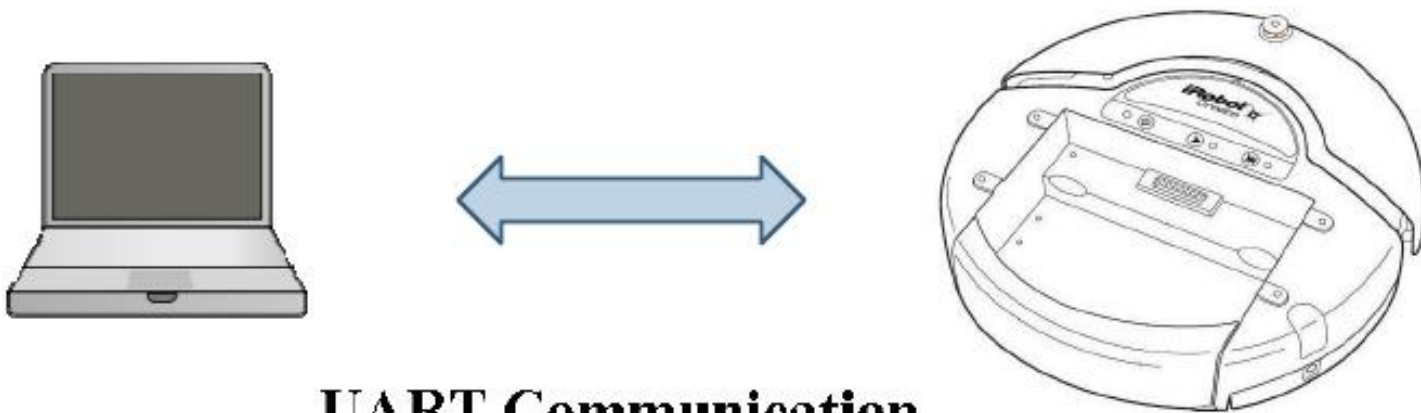


Hardware Components

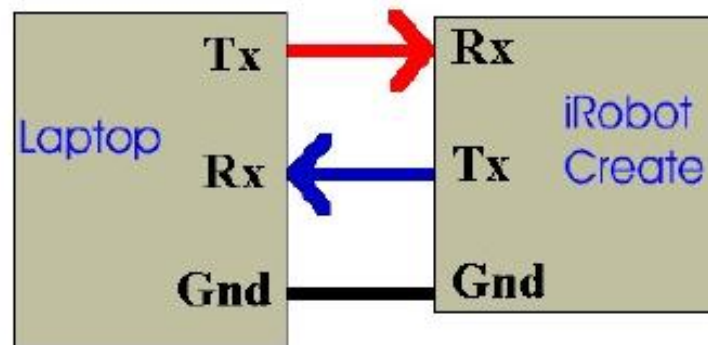


Component	Price (USD)
iRobot Create	\$129.00
iRobot Create w/ Battery	\$219.00
iRobot Create w/ Premium Development Package	\$299.00
BAM Wireless Accessory	\$59.99
USB Bluetooth Radio	\$39.95
Battery	\$69.99
Battery Charger	\$39.99
Command Module	\$59.99

Communication between ROS and iRobot-Creat

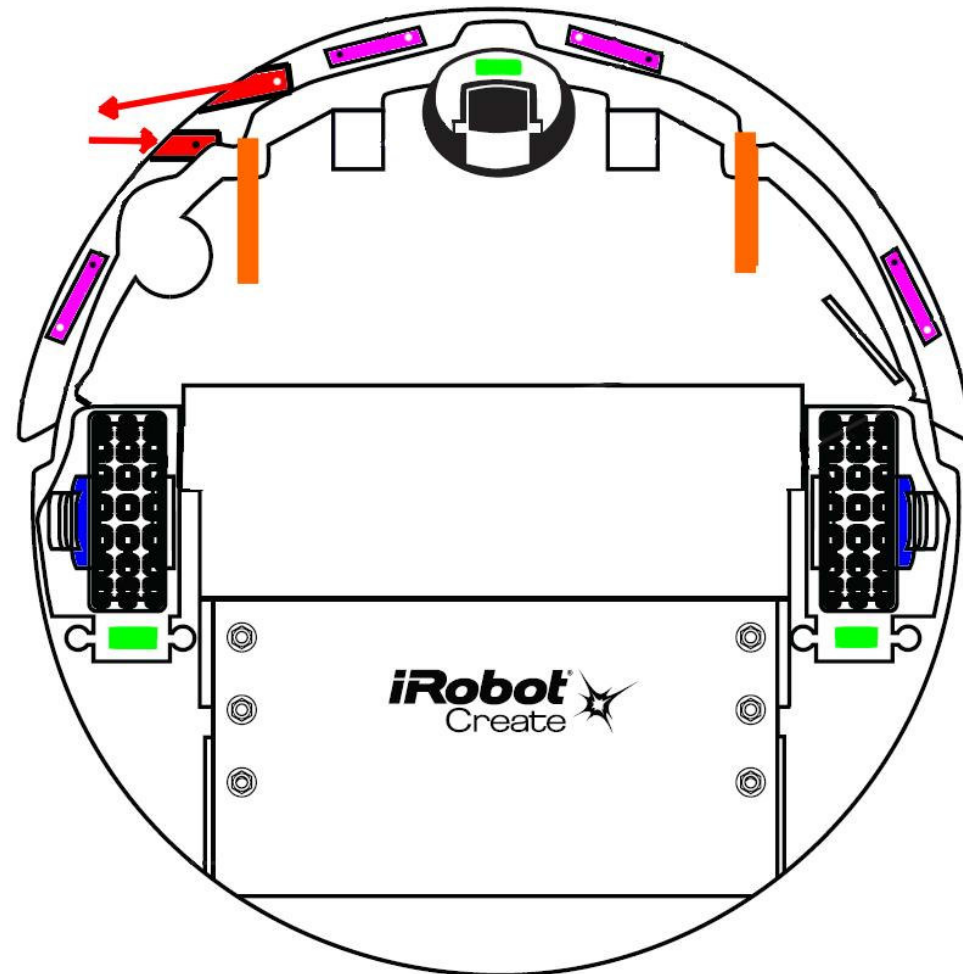


UART Communication



Sensor Locations

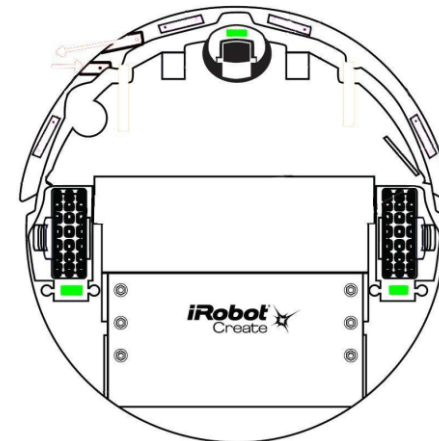
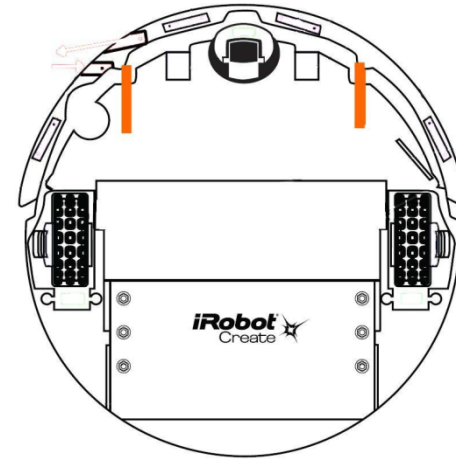
Sensor / Input	Number Available
Wheel Encoder	2
Wall Sensor IR	1
Omnidirectional IR	1
Cliff Sensor	4
Bump Sensor	2
Wheel Drop Sensors	3
Buttons	2
Digital Input	4
Analog Input	1



Bump/Wheel Drop Sensor

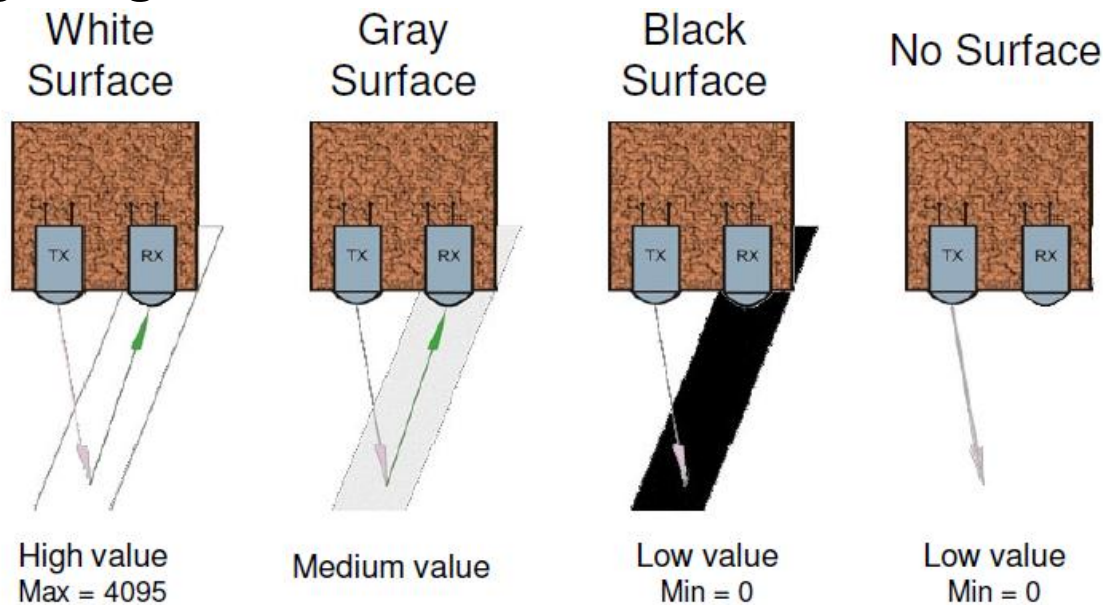
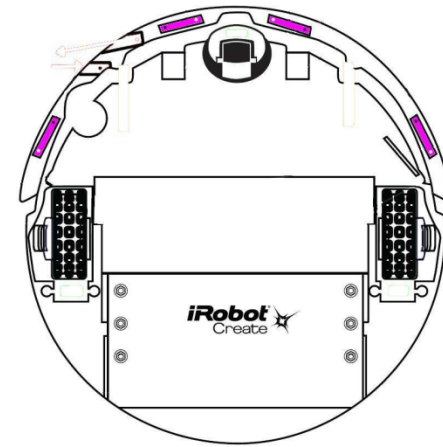
- Two digital signals
 - Left Bumper
 - Right Bumper

- Three digital inputs
 - Front Wheel Drop
 - Left Wheel Drop
 - Right Wheel Drop



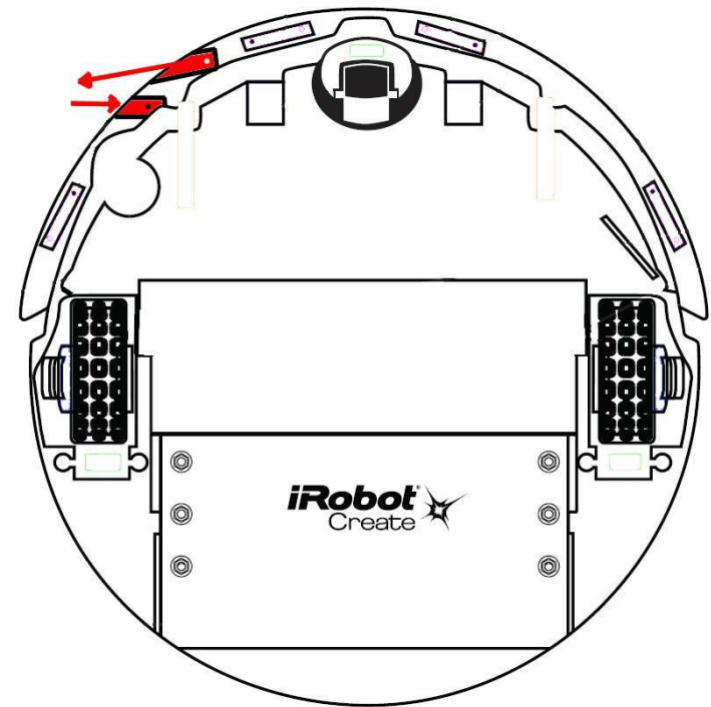
Cliff Sensor

- Four analog inputs
 - Cliff Left Signal
 - Cliff Front Left Signal
 - Cliff Front Right Signal
 - Cliff Right Signal



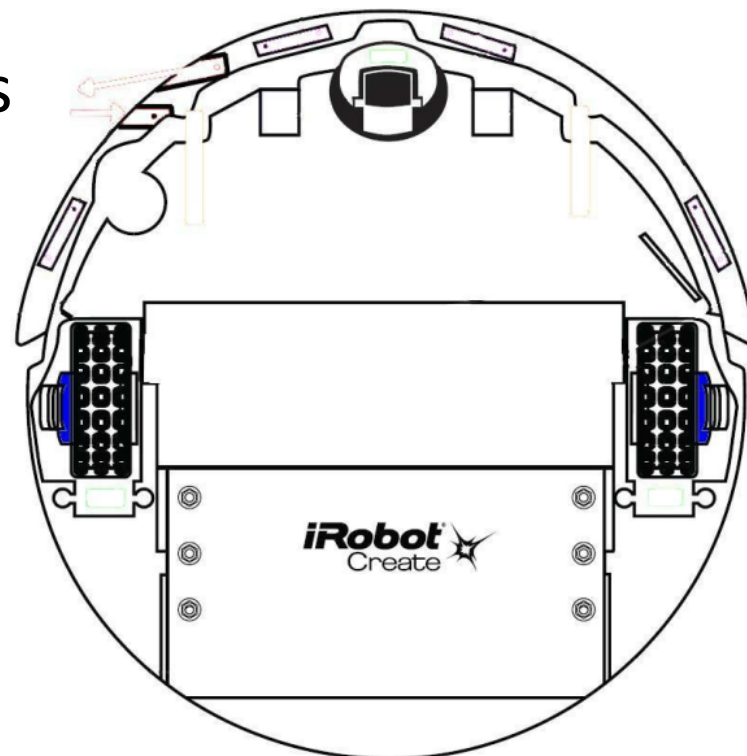
Wall Sensor

- Analog Sensor
 - Value relates to the distance between wall and Create
 - 0 = No wall seen



Wheel Encoder

- Digital Sensor
 - Distance since last request
 - Angle since last request
 - Used internally to control wheel speed

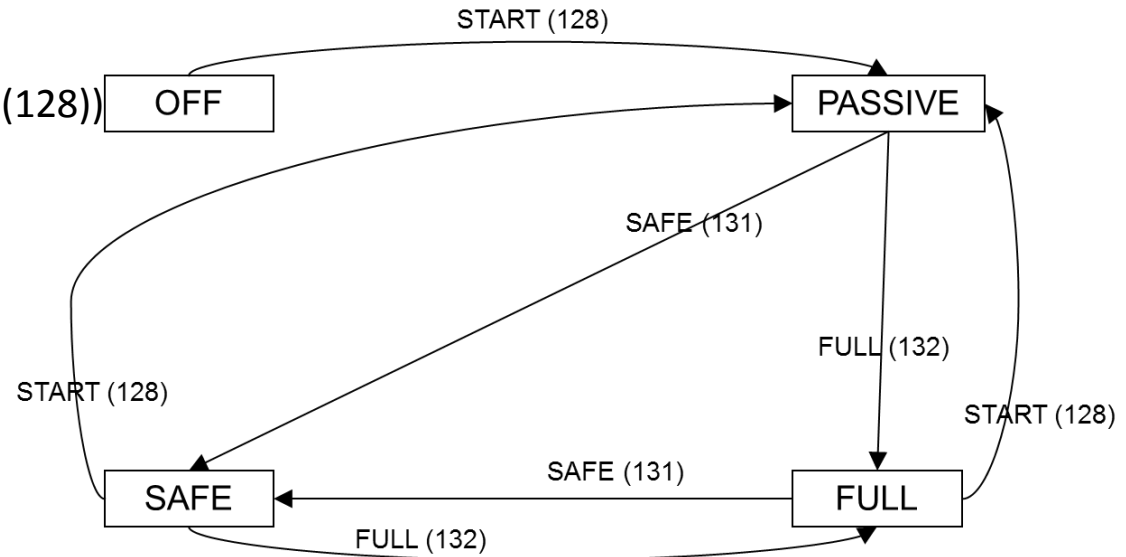


Overview

- Based on *Roomba* robotic vacuum cleaner
- Programmable with **open interface** of over 100 commands
- 32 internal and external sensors including bumpers and infrared
- Expansion port to add **microcontroller**, **Bluetooth**, and/or additional sensors

Modes

- **OFF**
 - Unresponsive (except START (128))
 - Can charge
- **PASSIVE**
 - Sensor status commands
 - **No Actuator commands**
 - Can charge
- **SAFE** ← **recommended!**
 - Sensor status commands
 - Actuator commands
 - **But reverts to PASSIVE if moving forward and any cliff sensor is activated; any wheel drop sensor is activated; or the charger is plugged in**
 - **No Charging**
- **FULL**
 - Sensor status commands
 - Actuator commands
 - **No Charging**



START

MODE

Opcode

Parameters

Programming (Microprocessor)

- *Command Module* plugs into expansion port
- 8-bit RISC microprocessor (~18 MHz)
- Upload C programs that send *Open Interface* commands and read sensor data
- Open source toolkit (Windows/Mac compatible)
- Four extra expansion ports to add custom hardware (sensors, LCD display, etc.)



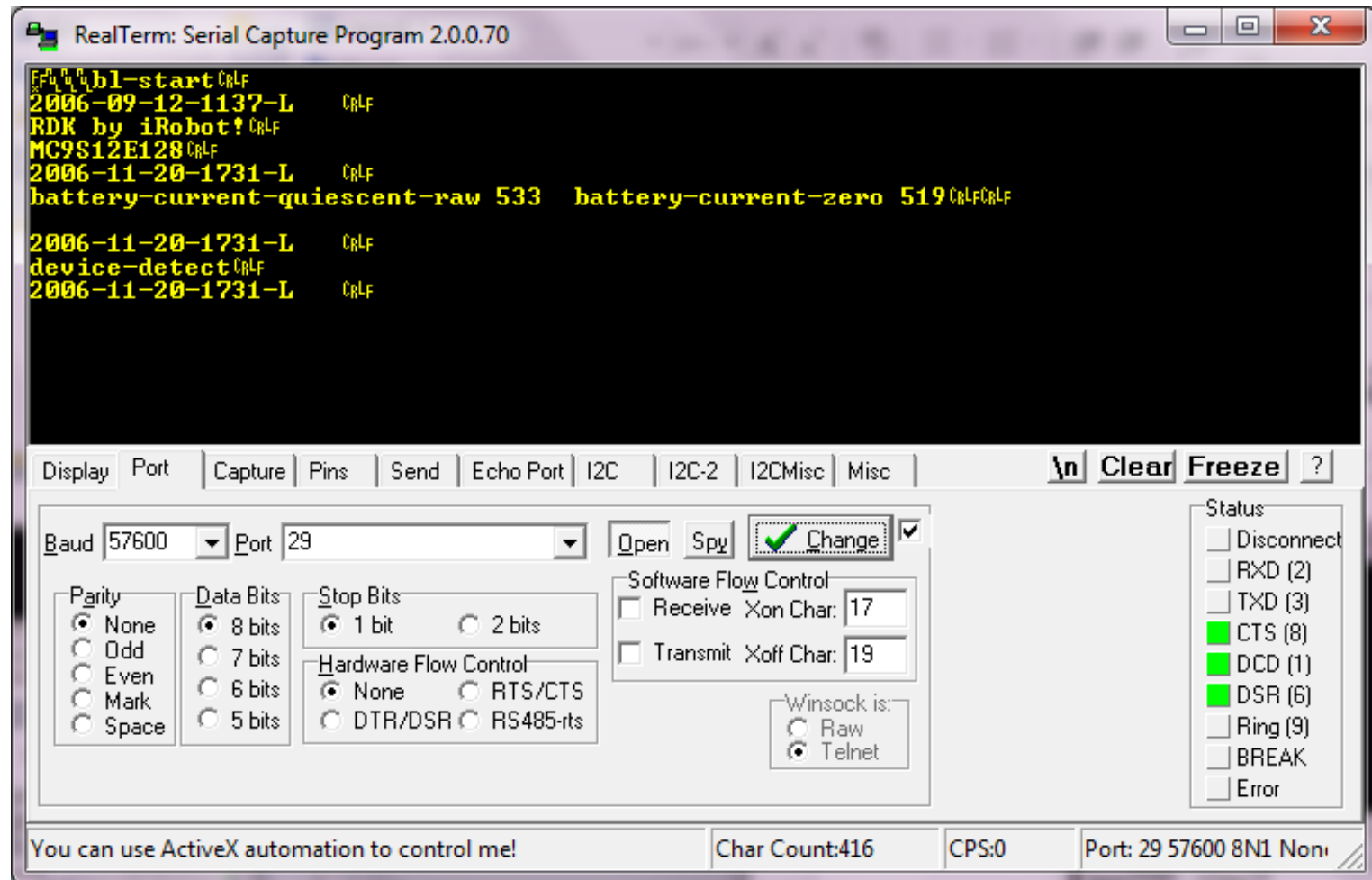
- Drive Create with specified velocity and radius (C function):

```
void drive(int16_t velocity, int16_t radius)
{
  byteTx(CmdDrive);
  byteTx((uint8_t)((velocity >> 8) & 0x00FF));
  byteTx((uint8_t)(velocity & 0x00FF));
  byteTx((uint8_t)((radius >> 8) & 0x00FF));
  byteTx((uint8_t)(radius & 0x00FF));
}
```

Terminal Communication-Linux

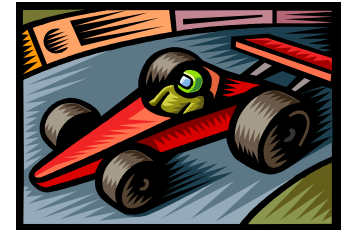
- Send *Open Interface* commands via a virtual serial port: 57600 baud, 8 data bits, 1 stop bit
 - Linux `stty -F /dev/ttyUSB0 57600 cs8 -cstopb`
- Receive sensor data back as packets
- Using any scripting language (Perl, Python, etc.)
- Drive *Create* forward (OI script):
 - 128 131 (Start in safe mode)
 - 137 0 100 128 0 (Drive forward 100 mm/s)

Terminal Communication-Windows



Actuators: Drive

- DRIVE (137 <velocity_{high}> <velocity_{low}> <radius_{high}> <radius_{low}>)
 - 2 short (16 bit) parameters
 - Each represented by 2 bytes, high byte first (so a total of 5 bytes for this command)
- Parameter 1: **velocity** in mm/sec
 - -500 to +500 (-ve means “backwards”)
- Parameter 2: **curve radius** in mm
 - -2000 to +2000 where -ve means clockwise and +ve means counterclockwise
 - 32768 means “go straight”
 - -1 and +1 mean spin in place clockwise, counterclockwise respectively



Motor Drive Command

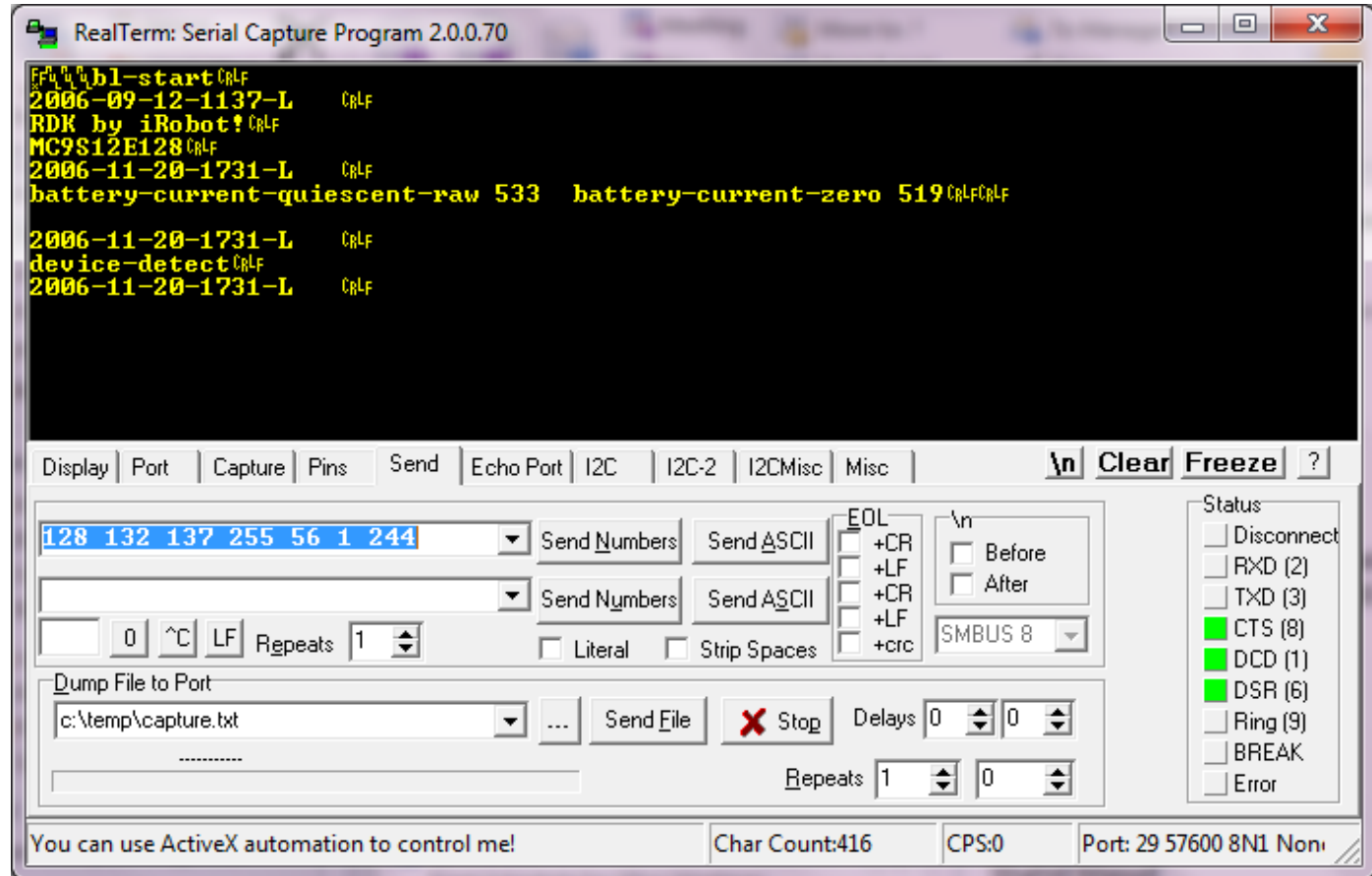
Example:

To drive in reverse at a velocity of -200 mm/s while turning at a radius of 500mm, send the following serial byte sequence:

[128] [132] [137]
[255] [56] [1] [244]

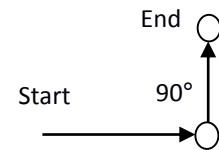
Velocity = -200 = hex FF38
= [hex FF] [hex 38] = [255] [56]

Radius = 500 = hex 01F4 =
[hex 01] [hex F4] = [1] [244]

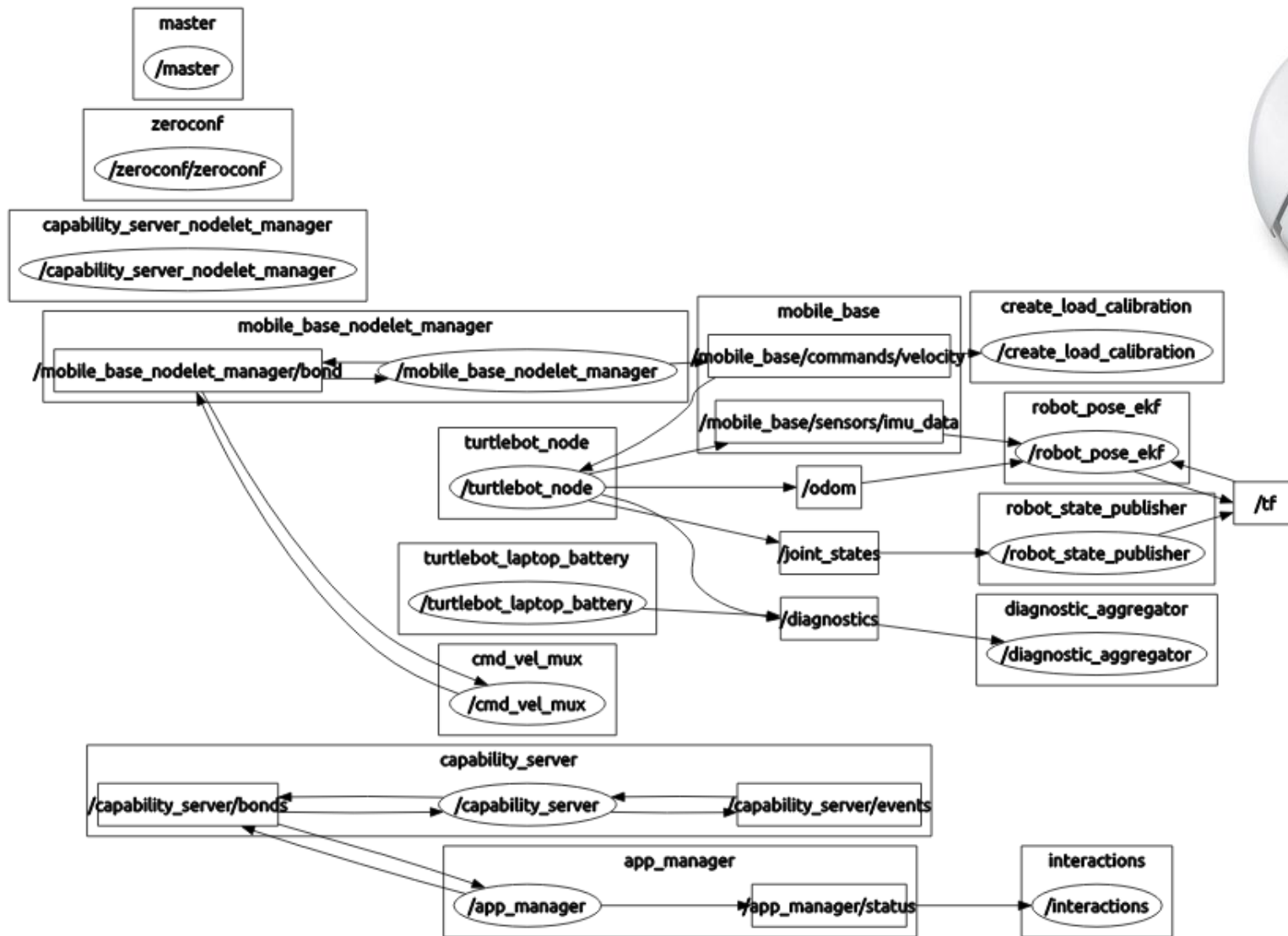


Task1: ROS with iRobot-Create Hardware

- Install turtlebot packages (Debs Installation only) found at [{url}](#).
- Follow Robot configuration robot from [{url}](#) for Create base (contrast with Kuboki).
- Configure serial port and launch turtlebot_bringup after configuring it for Create base. [{url}](#) Only run TurtleBot Bringup (not Workstation bringup).
- Navigate the robot with turtlebot_teleop package for keyboard. [{url}](#)
- Echo odometry. Get familiar with rosbag.
- Visualize the odometry in RViz.
- Now perform the following task 6 times, and record each run in rosbag for latter analysis



iRobot-Create with ROS



ROS Communication with iRobot-Create

- After turtlebot package installation
- Configure stack for iRobot-Create
 - export TURTLEBOT_BASE=create
 - export TURTLEBOT_STACKS=circles
 - export TURTLEBOT_SERIAL_PORT=/dev/ttyUSB0
- Bringup
 - roslaunch turtlebot_bringup minimal.launch
serialport:=/dev/ttyUSB0
 - Test: rostopic list
- Teleop
 - roslaunch turtlebot_teleop keyboard_teleop.launch
 - roslaunch turtlebot_teleop logitech.launch

Sensor and Odometry Topic

```

bumps_wheeldrops: 0
wall: False
cliff_left: False
cliff_front_left: False
cliff_front_right: False
cliff_right: False
virtual_wall: False
motor_overcurrents: 0
dirt_detector_left: 0
dirt_detector_right: 0
remote_opcode: 255
buttons: 0
distance: 0.0
angle: 0.0
charging_state: 0
voltage: 15224
current: -169
temperature: 21
charge: 2699
capacity: 2702
wall_signal: 0
cliff_left_signal: 1052
cliff_front_left_signal: 2108
cliff_front_right_signal: 1204
cliff_right_signal: 888
user_digital_outputs: 1
user_digital_inputs: 0
user_analog_input: 5
charging_sources_available: 0
oi_mode: 3
song_number: 0
song_playing: False
number_of_stream_packets: 0
requested_velocity: 0
requested_radius: 0
requested_right_velocity: 0
requested_left_velocity: 0
---
^Chamza@hamza-laptop:~$ rostopic echo /mobile_base/sensors/core

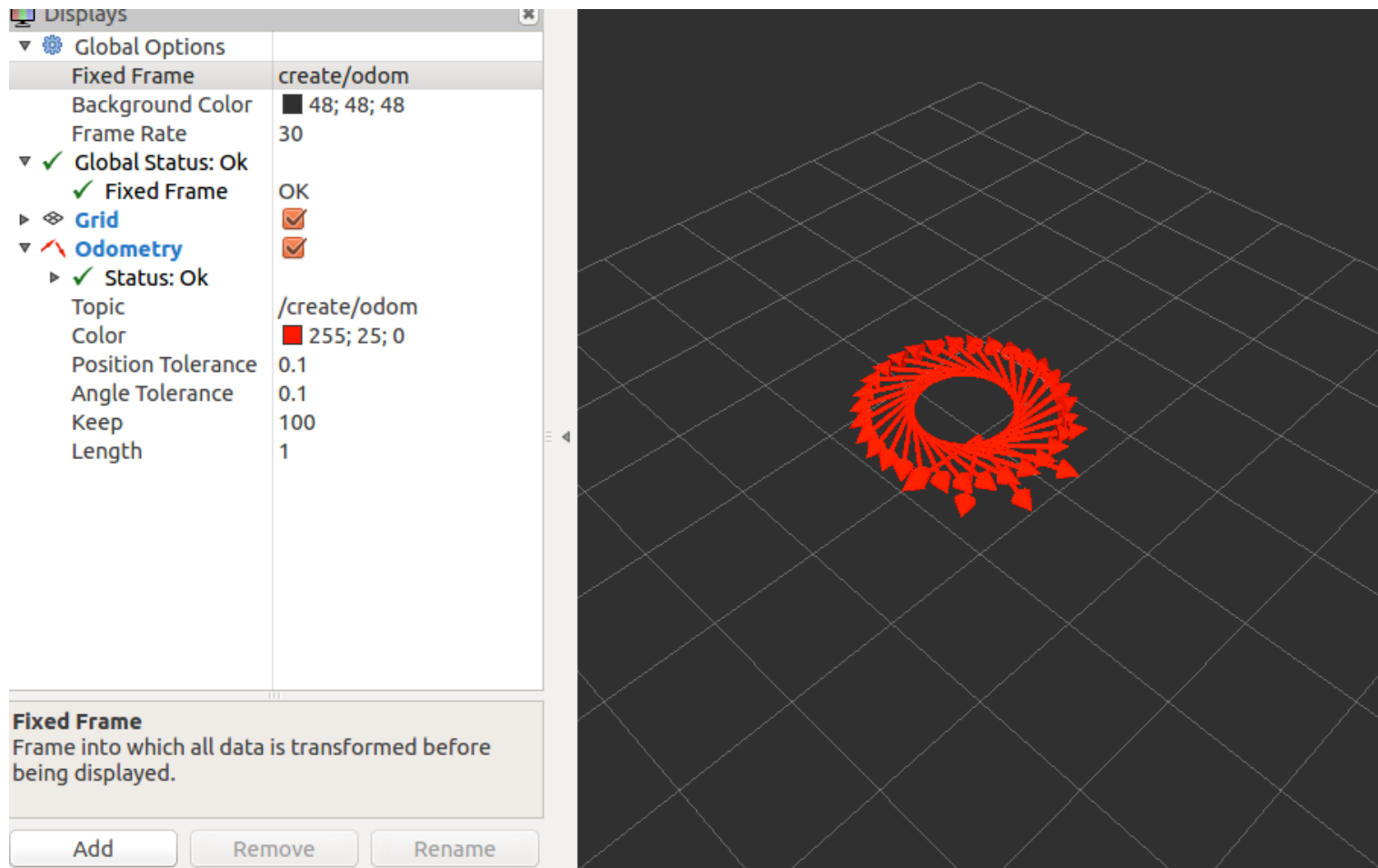
```

```

/opt/ros/hydro/share/gazebo_ros/launch/em... x ahmad@ahmad-laptop: ~ x
pose:
pose:
  position:
    x: -0.00487205904455
    y: -1.49355393522
    z: 0.000946513831449
  orientation:
    x: -3.36661727843e-05
    y: -0.000227964754744
    z: -0.000167602912964
    w: 0.999999959404
twist:
twist:
  linear:
    x: 0.000761776303189
    y: 0.00015526727852
    z: 0.0
  angular:
    x: 0.0
    y: 0.0
    z: 5.54694543396e-06
---
ahmad@ahmad-laptop:~$ rostopic echo -c --noarr /create/odom

```

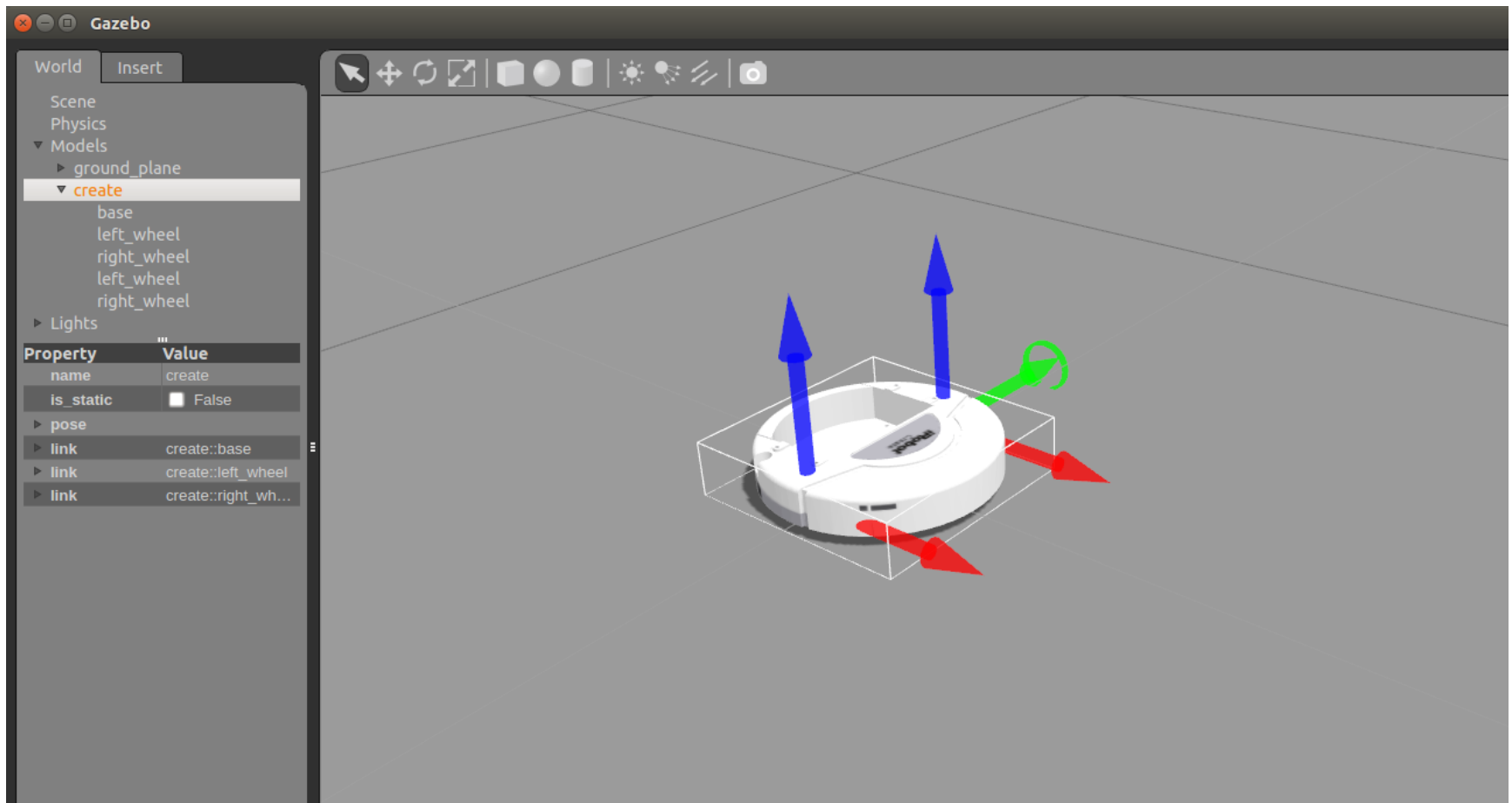
RViz



Task2: ROS with iRobot-Create Gazebo Model

- Spawning iRobot Create in Gazebo. [Launch gazebo with gazebo_ros package]
- Incorporate differential-drive plugin to get the iRobot odometry data published as ROS topic (like done in Lab 2). Also add the following tags within the odometry plugin tag:
 - `<wheelAcceleration>0</wheelAcceleration>`
 - `<odometrySource>encoder</odometrySource>`
- Modify the left and right wheel mu and mu2 inside the <ode> tag. These two parameters represent the coulombs friction coefficient of friction. The value can be between 0 and Infinity, where zero means a frictionless surface (Maximum slippage).
- Using turtlebot_teleop, navigate the robot along a L-shaped path. [Translate, Rotate]
 - `roslaunch turtlebot_teleop turtlebot_teleop_key`
- Record and replay the simulated experiment dataset (odometry) in RViz to get graded.

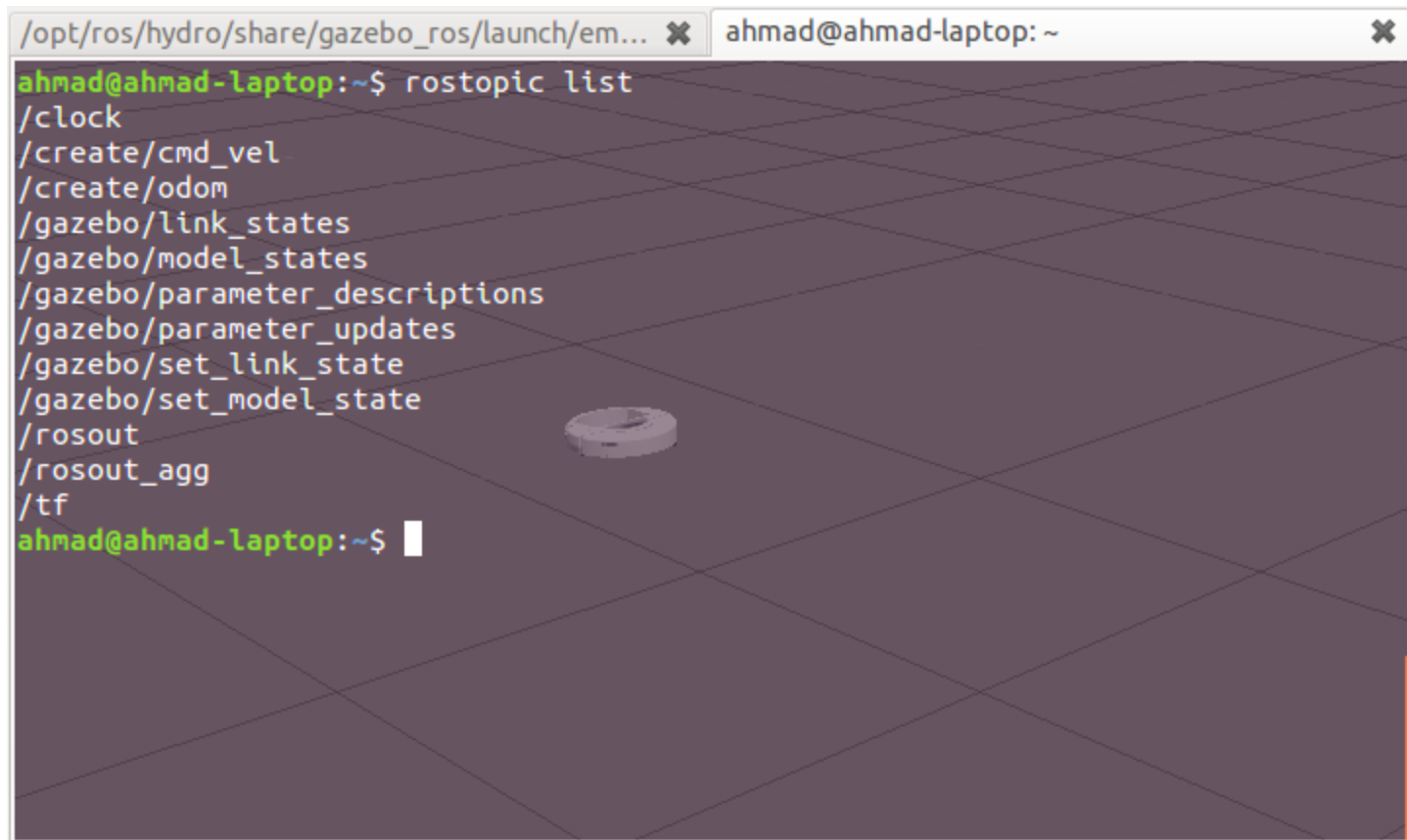
Gazebo Model of iRobot-Create



Odometry Plugin

```
<plugin name="differential_drive_controller" filename="libgazebo_ros_diff_drive.so">
<robotNamespace>/irobot</robotNamespace>
<publishWheelTF>false</publishWheelTF>
<publishWheelJointState>true</publishWheelJointState>
<alwaysOn>true</alwaysOn>
<updateRate>10</updateRate>
<leftJoint>left_wheel</leftJoint>
<rightJoint>right_wheel</rightJoint>
<wheelSeparation>0.283</wheelSeparation>
<wheelDiameter>0.066</wheelDiameter>
  <wheelTorque>5.0</wheelTorque>
  <wheelAcceleration>0</wheelAcceleration>
<commandTopic>cmd_vel</commandTopic>
  <odometryTopic>odom</odometryTopic>
  <odometryFrame>odom</odometryFrame>
  <odometrySource>encoder</odometrySource>
<robotBaseFrame>/base_footprint</robotBaseFrame>
</plugin>
```

Odometry/Cmd_vel Topics Published by Simulated model



```
/opt/ros/hydro/share/gazebo_ros/launch/em... x ahmad@ahmad-laptop: ~ x
ahmad@ahmad-laptop:~$ rostopic list
/clock
/create/cmd_vel
/create/odom
/gazebo/link_states
/gazebo/model_states
/gazebo/parameter_descriptions
/gazebo/parameter_updates
/gazebo/set_link_state
/gazebo/set_model_state
/rosout
/rosout_agg
/tf
ahmad@ahmad-laptop:~$
```

Wheel Slippage in Gazebo

```
<link name="left_wheel">
```

```
...
```

```
<surface>  
  <friction>  
    <ode>  
      <mu>10</mu>  
      <mu2>10</mu2>  
      <fdir1>0 0 0</fdir1>  
      <slip1>0</slip1>  
      <slip2>0</slip2>  
    </ode>  
  </friction>  
</surface>
```

```
...
```

```
</link>
```

```
<link name="right_wheel">
```

```
...
```

```
<surface>  
  <friction>  
    <ode>  
      <mu>10</mu>  
      <mu2>10</mu2>  
      <fdir1>0 0 0</fdir1>  
      <slip1>0</slip1>  
      <slip2>0</slip2>  
    </ode>  
  </friction>  
</surface>
```

```
...
```

```
</link>
```

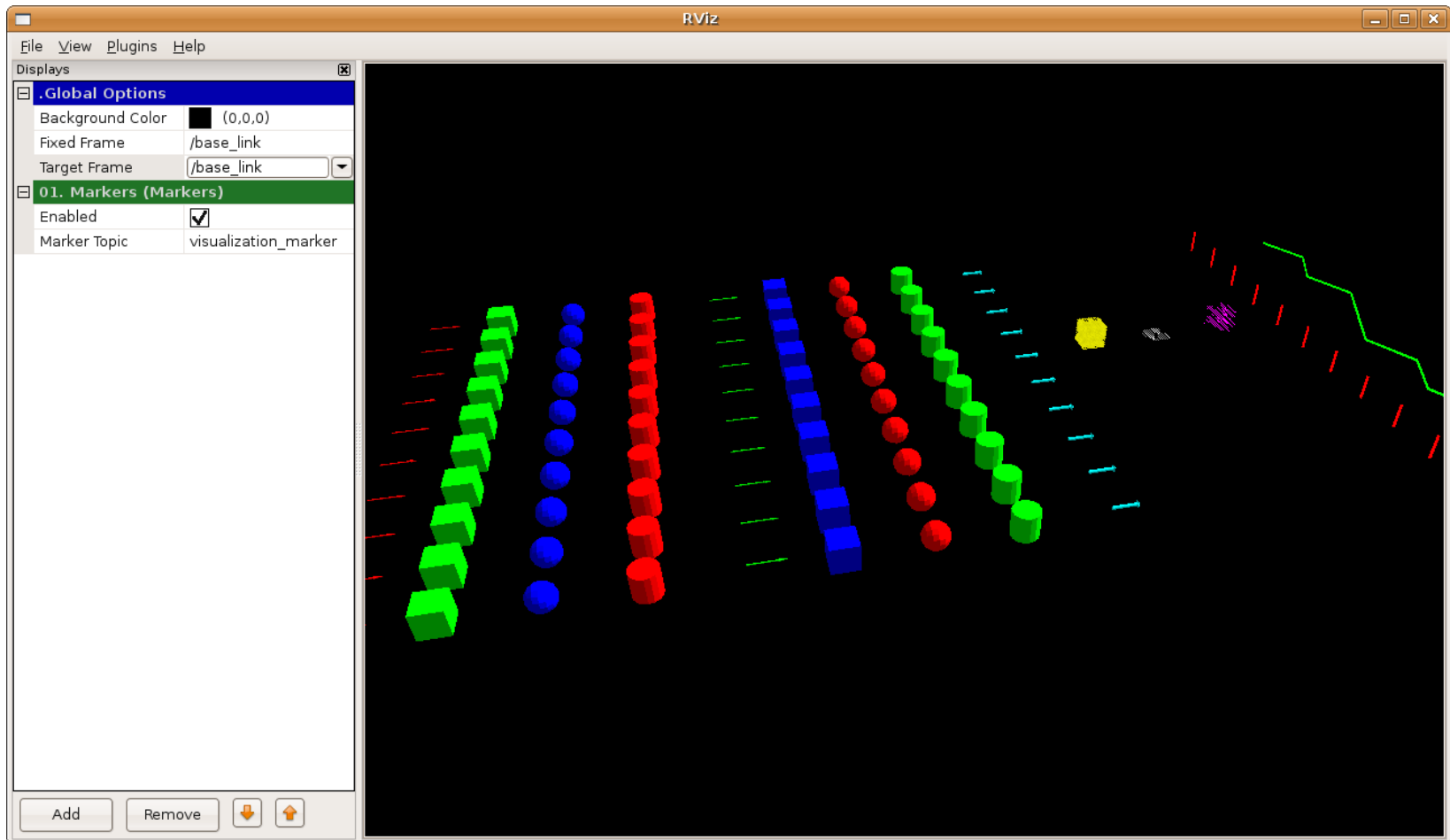
Utility Code – ROS Timers

```
#include "ros/ros.h"
void callback1(const ros::TimerEvent&)
{
    ROS_INFO("Callback 1 triggered");
}

void callback2(const ros::TimerEvent&)
{
    ROS_INFO("Callback 2 triggered");
}

int main(int argc, char **argv)
{
    ros::init(argc, argv, "talker");
    ros::NodeHandle n;
    ros::Timer timer1 = n.createTimer(ros::Duration(0.1), callback1);
    ros::Timer timer2 = n.createTimer(ros::Duration(1.0), callback2);
    ros::spin();
    return 0;
}
```

Custom Objects Visualization in RViz



Utility Code – Marker

```
#include <ros/ros.h>
#include <visualization_msgs/Marker.h>
int main( int argc, char** argv )
{
    ...
    ros::Publisher marker_pub =
n.advertise<visualization_msgs::Marker>("visualization_marker", 1);
    visualization_msgs::Marker marker;
    marker.header.stamp = ros::Time::now();
    marker.type = visualization_msgs::Marker::CUBE;
    marker.action = visualization_msgs::Marker::ADD;
    marker.pose.position.x = 0;
    marker.pose.position.y = 0;
    marker.pose.position.z = 0;
    ...
    marker.lifetime = ros::Duration();
    marker_pub.publish(marker);
    ...
}
```

Lab Assignment

- Implement a ROS node to timely publish `/cmd_vel` topic for trajectory following (L-shaped) 1m x 1m. Move the robot using this node and visualize in Gazebo. [You will need to implement `ros::Timer` so that you can keep track of duration of publishing data]
- Read published (noisy) odometry and convert the data into $(\delta_{rot1}, \delta_{trans}, \delta_{rot2})$ convention, in the same node.
- Also read create model's position and orientation states from `/gazebo/model_states` as ground truth. At this point, you will have (x, x', u) as learned in Lecture 3.
 - x : Initial pose of robot [from model states]
 - x' : Final pose of robot [from model states]
 - u : Odometry (rot, trans, rot) [from step 2]
- Publish the final pose of robot (u) as a Points type marker in `visualization_msgs::Marker`. [{url}](#)
- Write code (inside the same node) for repeating steps 1-4 (run L-shaped trajectory 100 times, collect data and put final pose as points in the same `visualization_msgs::Marker` message).
 - In each iteration, there will be some error between ground truth pose (obtained through `model_states`) and odometric data (obtained from noisy `/odom`)
- Plot the 2D points where the robot arrives at end of each trajectory in RViz. [Use `MarkerArray`] [You will get a scatter of points, as well as the desired 2D point]
- Find the mean and variance of $(\delta_{rot1}, \delta_{trans}, \delta_{rot2})$ from sampled data.
- **BONUS:** Graphically plot an oval gray region on the 2D plot of scattered points, that shows mean point and two-standard-deviations confidence region around the mean.