# Design and Implementation of Self Learning Autonomous Robot using Neural Networks under ROS (Robot Operating System) Platform

**Salam Al-Khammasi, Adnan Qayym, Prof. Tarek Sobh**
**Computer Science and Engineering Department, University of Bridgeport, Bridgeport, CT 06604, USA, (salkhamm, aqayyum)@mybridgeport.edu, sobh@Bridgeport.edu**

## Abstract

In this poster, we are designed and implemented an avoidance obstacle robot by using ROS (Robot operating system) as main platform. Neural Network algorithm has been used to program the robot. The algorithm has been written in Python programming language. In the hardware part, Arduino Uno Board with Ultra sonic sensor have been used to to detect the obstacles. Our contribution will be how to make the Robot detects the obstacle using neural network by learning himself from the environment and save that data which is getting by the ultra-sonic sensor to the base station so when it comes back to the same environment, the robot will not need to do the same procedure because the data already saved to the Base Station. All the related variables like Velocity, Acceleration and distance, etc. will get from ROS platform. The ROS will minimize the coding and gives us relative results. The communication between the robot and the base station will be wireless.

## Introduction

ROS (Robot Operating System) is the most modern framework which is used to improve the software programming in the robotics field.

ROS has been involved to enhance and develop the robotics in many fields. For instance, industrial robots, biomedical robots, underwater robots, space robots, military robots, etc... ROS is open source software supported on Linux and can support many different programing languages like C++ and Python.

In our project, we have used the latest version which is released in July 2014, named as “Indigo Igloo”.

ROS simplified the function of creating complex robot behavior by using many built in tools and libraries. It’s growing up very fast among different Robotics platforms. As a result of using ROS, our team has reduced the consuming time of the programming to the half. In this project, our team has designed and implemented an obstacle avoidance autonomous robot by using neural network techniques by using Python programming language under ROS framework.

**An example robot application (known-map navigation):**

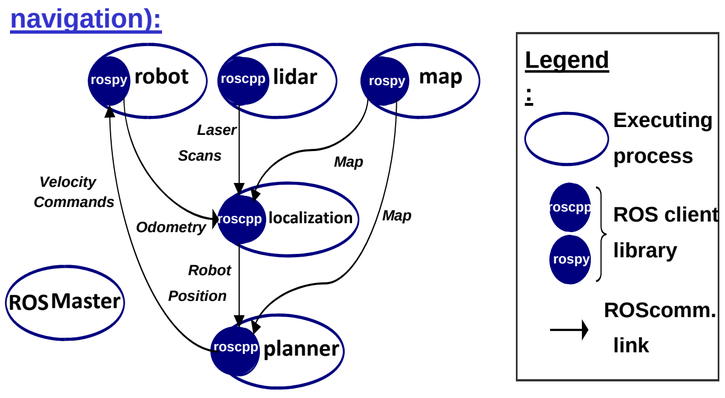



Figure 1: Robot application sing ROS

In this work we have used offline path planning. I.e. the path planning can be done before the robot starts to move. We implemented the proposed algorithm in python language under ROS framework. ROS have many built in packages for robotics that reduce our consumption time of programming to half. We used the simple neural network technique, i.e. ‘Learning your environment’ means robot will collect the data (offline) from its environment and saved it for the situation when robot counters the same situation again.

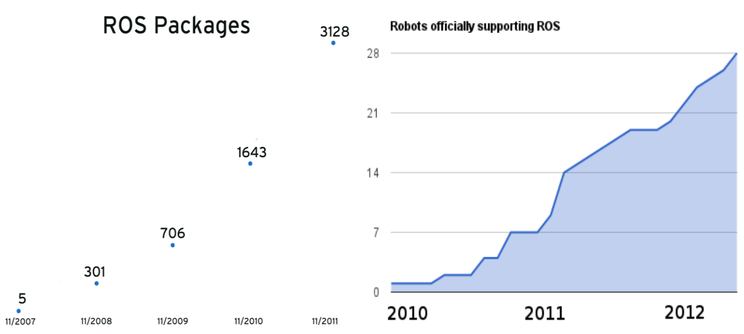



Figure 2: The increment in the number of packages and the number of robots supporting by ROS

### Goals of Using ROS :

- **Peer-to-peer**
  – Heterogeneous networks and scaling
- **Flexible:** do not impose methodology
  – Does not wrap “main()”
- **Thin**
  – Encourages development of ROS-independent libraries
- **Tools-based**
  – Microkernel vs. monolithic
- **Multi-lingual**
  – C++, Python, Java, Lisp, MATLAB, Lua
- **Free and open source**
- **Scalable**
  – Small to large runtime systems, development processes

## Hardware Implementation

For the hardware part, as shown in the figure 3 the robot has been built with Arduino UNO board with ultra-sonic sensor. The ROS will minimize the coding and gives us relative results. The communication between the robot and the base station will be wireless.. For distance sensing, SRF-08 ultrasonic sensor were used. Additionally, the platforms are also equipped with an Xbee Shield from Maxstream, consisting on a ZigBee communication module with an antenna attached on top of the Arduino Uno board as an expansion module. This Xbee Series1 module is powered at 2mW having a range between 40m to 120m, for indoor and outdoor operation. The robot will communicate with the base station using Wireless Xbee modules which provide communication via Wireless Wi-Fi 802.11 b/g/. One Xbee module is attached to the base computer through USB port.

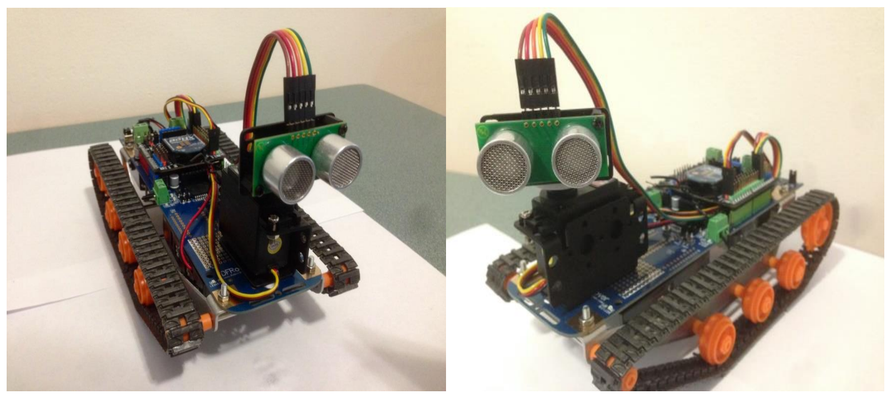


Figure 3 : Self-Learning autonomous robot with Ultra-Sonic Sensor

## Software Architecture

### Nodes, Messages, & Topics

- **Nodes : processes performing computation in the ROS system**
  - **Vertices in the ROS computational graph**
  - **Created by including and initializing a ROS client library in the program’s source code**
- **i.e. roscpp for C++, rospy for Python (also rosjava, roslisp, roslua)**
- **Each instance must have a unique *name* ( text string )**
- **Also have a *type* – filesystem location of the executable**
- **Messages : packets of data sent between ROS nodes**

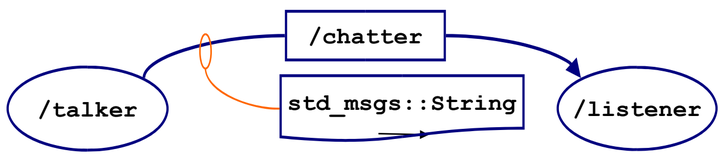
– **Named data structure comprised of strictly typed fields, defined in *.msg* files**

- **Somewhat similar to ‘C’ structs**

– **Converted to client language data structures during build / compile**

- **Topics: named *unidirectional* communication links between ROS nodes**
- **Form edges in the ROS computation graph**
- **Topics are named (text string)**
- **One or more nodes may publish messages to a topic**
- **One or more nodes may subscribe to messages on a topic**
- **Each topic is linked to a single message type**
- **Created dynamically at runtime by ROS nodes (using the ROS client library)**

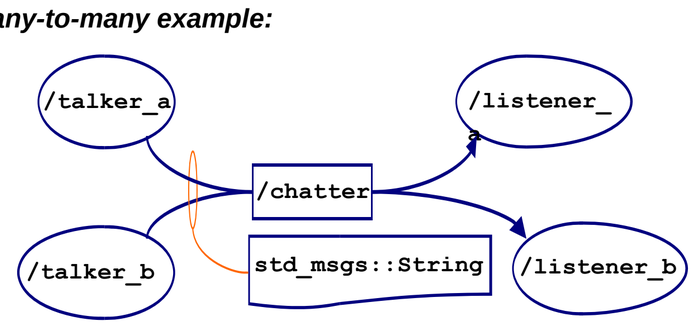
**A simple example:**




**The /talker node *publishes* messages with type The /listener node *subscribes* to messages with**

**std_msgs::String on the /chatter topic type std_msgs::String the /chatter topic**

***Many-to-many example:***




**Both the /talker_a and /talker_b nodes *publish* messages with type std_msgs::String on the /chatter topic**

**Both the /listener_a and /listener_b node *subscribe* to messages with type std_msgs::String the /chatter topic**

### Inspecting the Computational Graph :

- **ROS inspection tools help debug the computational graph**
  **Also useful for understanding how ROS & nodes work**
- **Command line tools:**
  **rosnode – information on ROS Nodes (publications, subscriptions) rosnode list; rosnode info /map_server; rosnode ping --all**
- **rostopic – information on ROS Topics rostopic list; rostopic echo /odom; rostopic hz /odom; rostopic bw /odom; rostopic pub /chatter std_msgs/String “hello”**
- **Graphical user interface - rqt**

**ROS Graph Plugin** **Topic Introspection Plugin** **Plot Plugin**

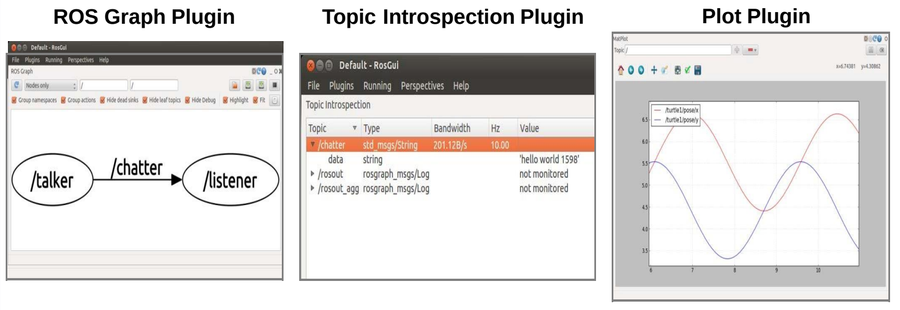



Figure 4 : ROS Graph, Topic, and Plot Plugins

## Proposed Work and Algorithm

We have implemented the algorithm that the robot learns internal model of the environment by recurrent neural network, it predicts succession of sensors inputs and on the base of the model it generates navigation steps as a motor commands. Robot will collect the data from its environment and saved all the instances of collected data for the situation if it ever comes again. Therefore we use the sensor data from the environment and the classical find object problem in our strategy was transform to the procedure ‘learning your environment’. The robot has in any position in workspace information about its distances to the all objects in this workspace. We use this information in neural network that learns these situations and in any position gives the free segment of space for safe and fastest path as output.

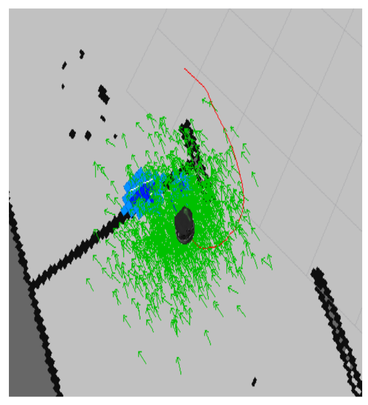


Figure 5: Global Planning Screenshot

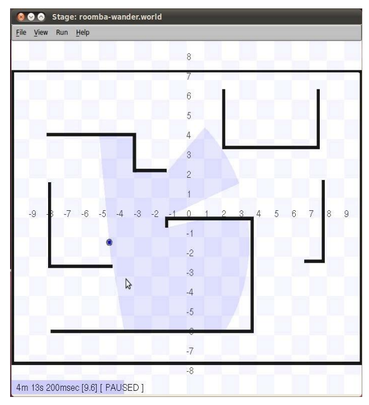



Figure 6 : 2D Planning Screenshot

Our motion planning algorithm can be summarized as follows:

- Turn on the robot.
- Initiate and upload the code to it.
- Robot will move fast and slow its motion once it detects an obstacle.
- Once it detects the obstacle, it will collect data through the Ultra-Sonic Sensor and send it to the base station to be saved.
- After collecting the data, the robot training the network (offline).
- The final step will be deploying and testing the environment.

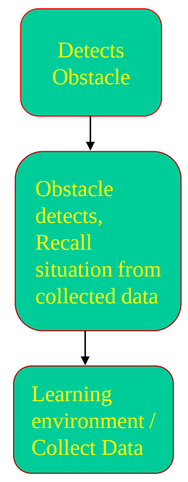



Figure 7: Motion Planning Block Diagram

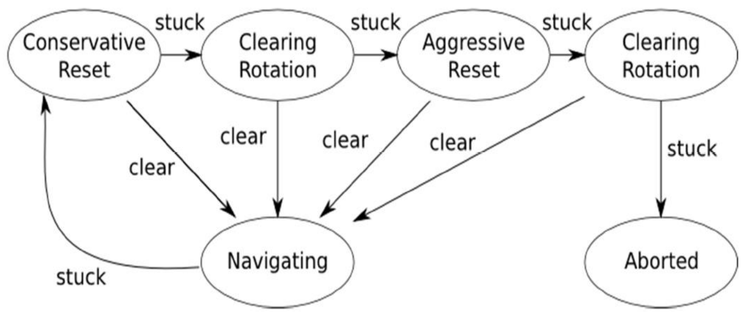



Figure 8: Move_Base Recovery procedure

## Conclusion

In conclusion, our team came up with a new modern robot to avoid the obstacles avoidance by using the new robotic platform (ROS) and our team enhances the time consumption to program the robot itself. In addition, our contribution will be how the robot will learn the environment with the neural network algorithm. Also, in this project, we reduced the time consumption for the programmer.

## Future work

As future work for this project. Any programmers can apply his own code or algorithm on this robot by just adding it to the platform like, mapping, navigation, surveillance and made small modification on the hardware part .

## Acknowledgment

Some slide content is adapted from the ROS.org Wiki under the Creative Commons Attribution 3.0 license. Also, some of the figure is adapted from the Lincoln laboratory at MIT http://ll.mit.edu

http://www.ros.org/wiki/ROS/Introduction
http://www.ros.org/wiki/stage http://playerstage.sourceforge.net/doc/Stage-3.2.1