

# ROS Middleware Final Project By Team Three

Legend Lee(10389043), Ken Luo(10389321), Yiran Fei(10389327)

*For Kobuki Yujin Robot*

(Dated: June 30, 2013)

## Abstract

This is the project base on package of Software for Kobuki, Yujin Robot's mobile research base(<http://www.ros.org/wiki/kobuki>). After trying many projects and papers posted at all pages of ROS, our team only found that this package can be excuted and run normaly at our computers. In order to run such a demo, we spent a lot of time in finding and verification on versions. At this runnable demo, we determine to change the code in the part of key operation ultimately. Because at first, we can't find where the controller stack really is, although it contains the stack named `controller_tutorial`. And the key operation stack to operate the robot remotely is very simple and it will lead to the problem of command dragging. Therefore, we want to modify this stack and make it better.

Our link of GIT code is [https://github.com/waldronluo/ROS\\_final](https://github.com/waldronluo/ROS_final).

Our link of Video is @todo.

Our instructions are post at the file README of git source.

## I. INTRODUCTION

Kobuki\_keyop originally was a node provide keyboard tele-operation for controlling a robot. This node can be used in both simulator and real world kobuki. But since there's no kobuki model for simulation, turtlebot model was used by the developer of kobuki in simulation instead. The change we made for keyop was to change the way of controlling. In the original version, the up/down/left/right key was to increase/decrease the speed of moving forward/backward/left/right, our version is that when i/k/j/l key was hit, the model will move forward/backward/turn left/turn right a little bit.

## II. METHODS

We use ecl package and publisher in ROS to accomplish this task. For each i/k/j/l, when the key was hit, it will publish a new speed with the variable, type of MotorPower in kobuki\_msgs, and publish it. After that, the thread will sleep for 0.03s, after that, an zero cmd will be published, which will stop the model.

## III. RESULTS

After debugging and trying for several times, the response was now sensitive, which means when the key was hit, the model will give a right move.

## IV. DISCUSSION

The hardest part in this was to process the input of keyboard. Though the original give some examples, all the actions are synchronous, which means each key input you process should wait for the publish of the msg. And it's a significant problem for controlling in our way, which the whole controlling node were designed in synchronous way. At first we didn't have the sleep method, which makes the result of moving very difficult to observe. Then we use methods provided by ctime.h, but the effect was horrible. The response was very not sensitive since it just provide second-level of sleep method. Thus we found ecl package provided in ROS which can have million-second level of sleep method. We then want to make a new thread to handle the process (one thread for receiving keyboard input and one

for processing the command), but found that normal pthread provided in linux can't be used due to lack knowledge of changing compile option in rosmake. Then we want to use spin provided in ROS. That will be our fuerte work. Besides adding a thread for processing, the collision detect was bad in the origin code in the simulation. It looks like that the collision detect can't be used in our trials. In our future work, we will try the collision detection and add collision avoidance in our code.

## **V. CONCLUSION**

After a series of trying and coding for a simple node, I think I learned much in ROS. ROS have lots of packages for us to used but we didn't know before.

### **Acknowledgments**

@todo