Workshop on Towards Fully Decentralized Multi-Robot Systems: Hardware, Software and Integration
2013 IEEE International Conference on Robotics and Automation, ICRA 2013
Karlsruhe, Germany, May 6 - 10, 2013, **http://icra2013mrs.tuebingen.mpg.de/**

A Software Framework for Multi-Robot Human Interaction

Junaed Sattar, Marc Grimson and James Little

Abstract— **This paper presents a software architecture for distributed human-robot interaction. This software framework encompasses mobile robots, "smart" devices, and general class of sensors. Built as an extension to the RobotOperating System (ROS) middleware, our framework extends the capabilities of ROS in two fundamental ways. Firstly, it enables multiple ROS software supervisors (*i.e.*, masters) to coexist efficiently and dynamically under the same network, and exchange data as required by commands given by non-expert human users. Secondly, because of the distributed nature of ROS running in different devices, individual device failure does not place the entire network at jeopardy. Also, in the case that network communication collapses entirely, each device falls back to operating as a standalone ROS entity with its own data and services, reverting back to a single ROS-enabled robotic entity. We describe the architecture and implementation of this framework across a number of ROS-enabled devices, implemented using ROS-Java and ROS-CPP application programming interfaces, running on Android devices, PR2 and Turtlebot robots, and RGB-D camera sensor nodes.**

I. Introduction

Mobile devices such as smart phones and tablet computers provide convenient and efficient methods of multimodal human interaction. Equipped with a variety of sensors, along with the ability to wirelessly connect to other devices make such devices especially attractive for use in a distributed wireless sensor network. Coupled with mobile robots, the entire network can be considered as a distributed human-machine interface. The integration and coherence of such a network depends heavily on the underlying software system for passing sensory data and human-interface commands. Thus, the focus of this paper is to investigate and recommend an approach towards seamless integration of diverse systems, in a robust, reliable and minimally fault-tolerant manner. Our particular interest in distributed software systems arises from the need to integrate multiple heterogeneous systems into a unified network of smart machines, mobile robots and human users.

This paper presents a software architecture for distributed human-interface systems. Specifically, our framework provides interfaces for human-machine dialog, and provides necessary means for communication, perception, task allocation and eventually, risk assessment in commands dictated from a human user. Our broader goal is to reduce uncertainty in human-robot dialog by providing a robust assessment of risk by evaluating the command through the relevant network components. To that goal, we rely on a network of devices running the Robot Operating System (ROS) middleware. The use of the ROS framework enables us to leverage available functionalities and also to build our own extension, creating a uniform interface across the devices, from both perspectives of user interfaces and application programming. One major concern is to have multiple ROS-enabled entities operate coherently on the same IP network, avoiding conflict and also providing access to required data to other entities, while maintaining modularity. Our framework achieves this by extending the ROS *"Multi-master"* system, which allows independent ROS-enabled entities (*i.e.*, modular collection of ROS nodes) to exchange data via ROS "topics", all the while maintaining data and operational integrity of the individual entities. Our approach has the added benefit of being inherently fault tolerant, as the failure of the network or any number of entities in the network will not affect the operation of the other entities, as they are merely communicating via a publisher-consumer model. At the worst case of a complete network failure, each entity will revert to operating stand-alone, which is no worse than the default operational mode. Our framework is also asynchronous, thereby allowing computation and user input commands at arbitrary rates. This paper presents our framework, with a human-interaction network comprised of multiple mobile robots, stand-alone RGB-D cameras, and mobile telephones and tablet computers.

Department of Computer Science, University of British Columbia, Vancouver, B.C., Canada. `junaed@cs.ubc.ca`, `grimsonreaper90@gmail.com`, `little@cs.ubc.ca`

II. Related work

Many mobile robots are programmed directly using hardware libraries and low-level Application Programming Interfaces (API). Unfortunately, programmers working with these libraries directly must first write a slew of basic functionalities (such as object detection and avoidance) before they can develop more sophisticated behaviors. Our proposed methodology leverages the ROS middleware, reusing already-existing algorithms whenever possible, and relying on provided mechanisms to develop high-level and complex behaviors.

The Player/Stage/Gazebo robot development suite [1] also provides commonly-used basic behavioral functions. The Player server establishes a standardized interface to a wide range of robot sensors and actuators over a network socket. This server is often accompanied by the Stage and Gazebo applications, which are visual environments for 2-D and 3-D robot simulators, respectively. As an established successor to the Player library, the Robot Operating System (ROS) software suite [2] provides a framework for message passing, interaction scheduling and code interconnection. This development environment is targeted at building mechanisms and abstraction tools to facilitate the task of robot programming, but with a much larger scope and more amorphous goal. We exploit broad-ranging facilities provided by ROS, in particular device transparency, a uniform application programming interface, and uniform data interface for robotic applications.

The Human-Robotics Interaction (HRI) literature features many different non-conventional interaction mechanisms. The RoboChat framework [3] maps robot actions and programming constructs onto various fiducial markers. By showing these visual symbols in a particular sequence, one can program robots to accomplish complex tasks without writing a single line of code. While RoboChat has been proven useful where verbal or other more "natural" means of interaction are not possible, we do not face such restrictions and have instead used the Google Speech Recognition API [4].

III. The Enhanced-MultiMaster System

The Enhanced-MultiMaster system lies at the core of our framework, and enables reliable communication between multiple ROS entities, and a human user. Currently (under ROS Groovy at the time of this writing), ROS provides an experimental ROS MultiMaster software stack [5] to handle multiple ROS masters under the same IP network. This stack allows ROS nodes to advertise or subscribe to topics from other nodes under different masters. The topic names and the URLs for their masters need to be known *a priori*. A specific ROS node called the "foreign relay" is used to advertise a ROS topic from one master to a different master. Once advertised, the remote master can subscribe to that topic as a local topic. Our work builds on the ROS MultiMaster stack, and provides two major enhancements:

1) **Dynamic Behavior**: The ROS MultiMaster stack requires either all the messages and the remote master URL have to be known at the beginning, and all the necessary foreign relays must be started at launch, or every foreign relay has to be manually started (by an expert human user) as they become needed. In our approach, we still require the topic name, but the remote URL is found dynamically and is not required as input.
2) **Entity Transparency**: In the ROS MultiMaster approach, a direct access is required to at least one of the two devices passing messages, as the foreign relay node needs to be run on the ROS master advertising topics. This requires an expert user to connect to the device, explicitly launch a foreign relay node and set up the topic advertisement. However with our system, as long as the other machines are running a specific node (described in Sec. III-A.2), a third machine can make a request for two entirely separate machines to communicate with each other. This enables a human user to use a smartphone as a user interface device, for example, and the smartphone automatically manages message passing between two arbitrary ROS entities.

A. Enhanced-MultiMaster System Components

In our framework, the non-ROS entities are capable of sending requests to ROS-based entities such that certain tasks given by the user are carried out by the system (*e.g.*, manipulation, navigation or surveillance tasks). The core of the Enhanced-MultiMaster system is based on three protocols– one to request topics from different ROS masters, one to provide topics if such a request is made, and one to route requests between masters, and also between ROS and non-ROS entities.

1) MultiMaster Requests: MultiMaster requests, encapsulated in datagram (*i.e.*, UDP) packets, are messages that enable a ROS node to subscribe to data published under a different node. In order for the system to pass messages between separate masters, requests have to be made by one or more ROS nodes specifying the type of request. Currently, the protocol implementation contains either an `ADV` or "advertise" request, `SUB` or "subscribe" request, or a `DEL`, which is a request to stop advertising or subscribing to topics. Additionally, the request requires two URLs (the "local" and "foreign" masters), and two topic names (the local and foreign topic). The added advantage of using basic UDP messaging to send such requests is ROS independence – as the request protocol does not have any dependency on ROS-specific services, any process can send such requests to a ROS master using a simple UDP message.

2) The MultiMaster-Listener: The *MultiMaster-Listener* provides access to nodes under a ROS master to other ROS masters in the same network, and is run on a ROS master. The MultiMaster-Listener process is I/O asynchronous and non-blocking. The non-blocking characteristics allows the ROS core to continue sending internal messages, such as checking when the ROS master has been shut down. Once a message is received, the five message components, namely the type (`ADV` or `DEL`), local master and foreign master URLs, local and foreign topic names are extracted. Based on the request type, the master running the MultiMaster-Listener process will either start advertising a local topic to the foreign master, or delete an already-published topic to a foreign master.

3) The MultiMaster-Master: The MultiMaster-Master acts as the supervisory controller for the entire system by sending commands to different ROS masters in the network. The MultiMaster-Master keeps track of all the ROS master entities by maintaining a vector of ROS masters. The MultiMaster-Master entity is able to operate even with just one device in the system, by routing subscription or publish commands. In the current implementation, user commands are received through the Voice Command Service associated with MultiMaster-Master (see Sec. IV-A.1), but any generic human-interface service may be used (*e.g.*, visual, touch or gestural interfaces). Once a command is received, the MultiMaster-Master extracts the desired command, and attempts to route the command to the appropriate entity in the network. First, it loops through a locally-maintained list of all the devices in the system to check whether the name given to the device (*e.g.*, "Charlie") is in the command (for instance, "Charlie, go to the kitchen"). If an exact match to the device identifier is found, it is chosen to perform the action. The next task performed by the MultiMaster-master is to identify and

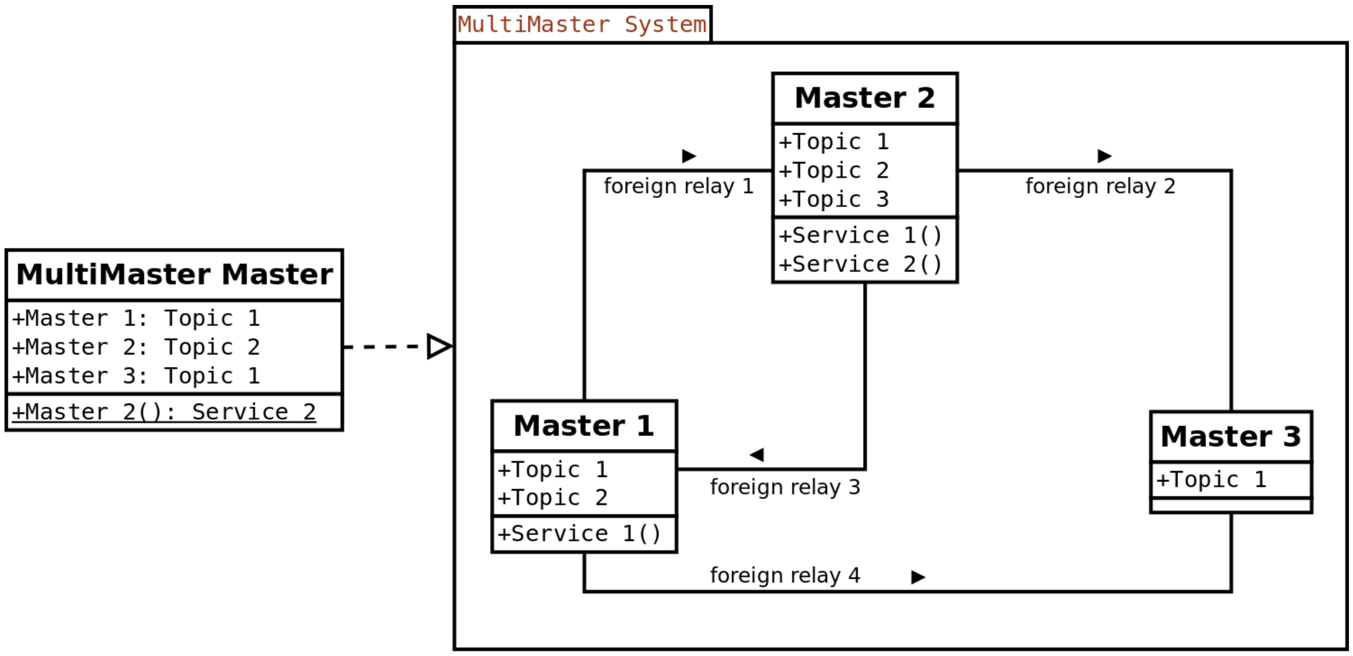


Fig. 1: Schematic diagram demonstrating the Enhanced-MultiMaster system. The foreign relays are used to advertise topics to remote ROS masters. The MultiMaster-master keeps a track of currently advertised topics and accessible services, and is able to initiate new foreign relays and remote topic advertisements.

extract the action phrase from the input. Depending on the nature of the command, and the task-specific requirements, the MultiMaster-Master is capable of initiating a variety of actions. Currently, our implementation includes robot locomotion actions, locate object action (*i.e.*, through sensory perception and locomotion), object manipulation actions and "fetch" actions.

The MultiMaster-Master is responsible for maintaining a list of devices in the network, providing access to globally advertised topics and services, and routing user commands to the correct entities in the network. If one or more of these tasks cannot be achieved, the MultiMaster-Master ensures graceful exit from the task execution stage, informs the user of the task failure, and updates its internal state description.

IV. Implementation

Currently, there are four individual entities in the network, although the number is arbitrarily scalable within the scope of the framework. The current network is comprised of three ROS-based entities and one generic system using UDP messages to request services from the other devices in the network. The non-ROS system is an Android-based smartphone, and the ROS systems are two mobile robots, namely a PR2 and a TurtleBot robot, and a static RGB-D camera node, with an Asus XTion camera running with the OpenNI stack. The Android device serves two purposes, as an human-command interface, and also an interface to add new entities (*e.g.*, robots, sensors) to the network. Other than the request for the URLs of the other devices, at the entities have no need to maintain a constant communication with the MultiMaster-master, or vice versa. This "stateless" communication between entities provide a degree of fault tolerance to the framework, as communication failure or entity malfunction does not affect individual entities (other than the ones experiencing malfunction), and individual entities are capable of performing their tasks independently.

A. Distributed Services

The system implements the following custom services:

1) Voice Command Service: The Voice Command Service is a ROS Service that allows the user to give commands to the MultiMaster-Master, which then decides how to act upon each command, and uses the text-to-speech method to provide feedback to the user.

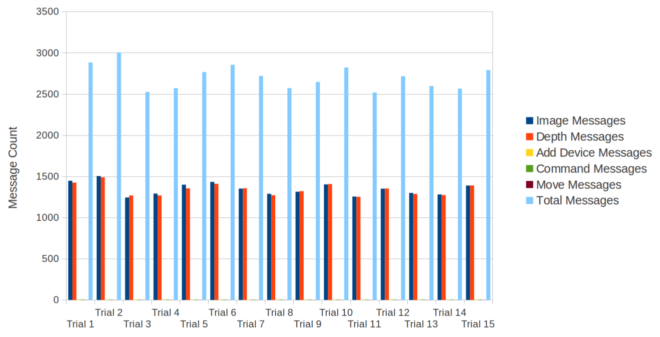
2) Color Finder Service: This ROS Service allows a device to find an object of a specific color in a ROS image topic, and returns its location in world coordinates. This image-to-world coordinate transformation enables a mobile robot to potentially reach an object of interest in the world representation once it is located by a camera.

V. Experiments

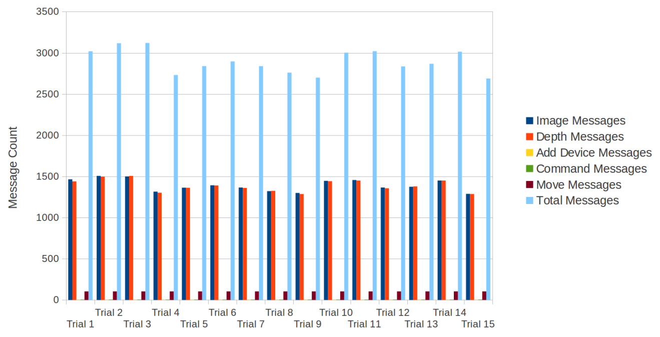
We conduct a set of human-to-multi-robot interaction experiments implemented on our system, both on simulation and also in a real-world setting.

A. Setup

In our simulated setting, the network consists of five mobile robots, a static RGB-D camera, and an Android-based smartphone running the human-interface application. In the real-world settings, the network contains the four devices described in Sec. IV, and is given an annotated map of the environment. We aim to quantify the number of messages



(a) Messages passed in the single master setting.



(b) Messages passed in the Enhanced-MultiMaster setting.

Fig. 2: Comparison of number of messages passed in the single-master and Enhanced- MultiMaster scenarios.

being passed by the MultiMaster-Enhanced protocol, and the bandwidth consumed. In a realistic scenario, minimizing the number of messages sent (and subsequently reducing the bandwidth taken) is an important factor in system performance, as all of the devices communicate wirelessly through a WiFi network, which is susceptible to interference and data packet loss.

B. *Scenario*

The following commands are given to the simulated robots (labeled robot1 to robot5):

1) Send robot1 to the kitchen area;
2) Send robot2 to the kitchen to find a blue mug, where there is a blue mug in the kitchen;
3) Sent robot3 to the kitchen to find a green towel, where there is no green towel in the kitchen;
4) Command robot4 to find the blue mug in its current location;
5) Give robot5 an invalid command (not in the robot vocabulary)

Each command is repeated 15 times, with robots in arbitrary locations in the environment. Also, to quantify the number of messages sent, the commands are sent in both a multiple-master setting and a single-master setting; that is, each of the commands are sent with no other robots active (*i.e.*, no other masters running) and again with all the other robots (and their corresponding ROS masters) active.

C. *Results*

The number of messages passed in each case and also the bandwidth taken by the system can are presented in figures 2a and 2b. On average, under our task setting, the MultiMaster scenario requires approximately 192 more messages across all message types (single-master averages 2704 messages, while multi-master averages 2896); however, in the MultiMaster scenario, all ROS masters exist in the network concurrently. The single-master setting sends 984 bytes *per* trial for all messages in that trial, resulting in an average of approximately 5 bytes *per* message. On the other hand, the Enhanced MultiMaster system sends 17560 bytes *per* trial for all messages in that trial, resulting in an average of approximately 90 bytes *per* message.

VI. Conclusion

This paper presents the design and implementation of a software systems framework based on ROS, geared towards use in distributed multi-modal human-robot interaction applications. This architecture enhances ROS-based robot control, by allowing non-expert human users to dictate commands. Applied in a domestic smart-home scenario, the framework makes it possible for mobile robots, smart portable devices, cameras and humans to cooperate and interact with each other in an efficient manner and also ensures safety in robot operations. Particular use can be foreseen in care homes or medical facilities, where a constant lookout must be maintained for the well-being of the inhabitants. We are continuing investigation into extending our framework towards including generic ROS services, and providing a level of security for exchanging data and services from entities on the network.

References

[1] B. P. Gerkey, R. T. Vaughan, and A. Howard, "The player/stage project: Tools for multi-robot and distributed sensor systems," *11th International Conference on Advanced Robotics*, pp. 317–323, 2003.

[2] M. Quigley, K. Conley, B. P. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "ROS: an open-source Robot Operating System," in *ICRA Workshop on Open Source Software*, 2009. [Online]. Available: http://pub1.willowgarage.com/ konolige/cs225B/docs/quigley-icra2009-ros.pdf

[3] G. Dudek, J. Sattar, and A. Xu, "A visual language for robot control and programming: A human-interface study," in *Proceedings of the International Conference on Robotics and Automation ICRA*, Rome, Italy, April 2007, pp. 2507–2513.

[4] "Google text to speech api," Developer Reference. [Online]. Available: http://developer.android.com/reference/android/speech/tts/TextToSpeech.html

[5] "Ros multimaster-experimental stack," ROS Wiki. [Online]. Available: http://www.ros.org/wiki/multimaster_experimental