Introduction

In our original project, we have four aims. The first aim is to create an agent simulator environment which is used to simulate how drones work. The second aim tries to improve the existing wearable interface designs. The third aim is to design an algorithm to control these agents through the wearable interface. Aim 4 is to improve human-computer interaction. When we finish all parts, users can control multi-drones to rescue people form wicked environment.

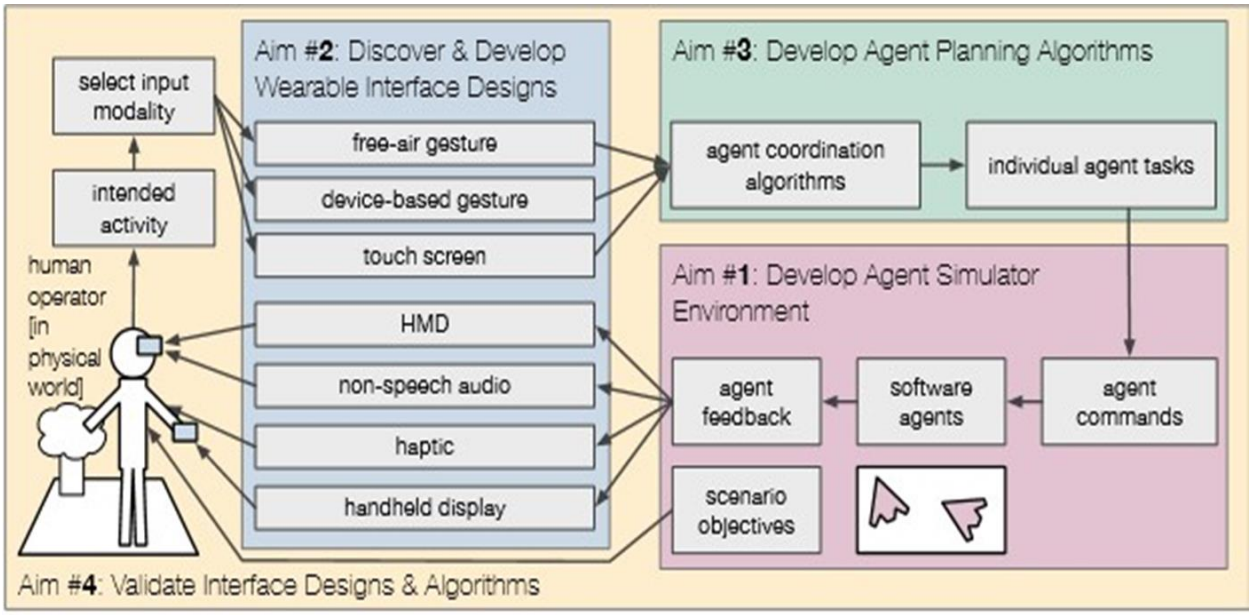
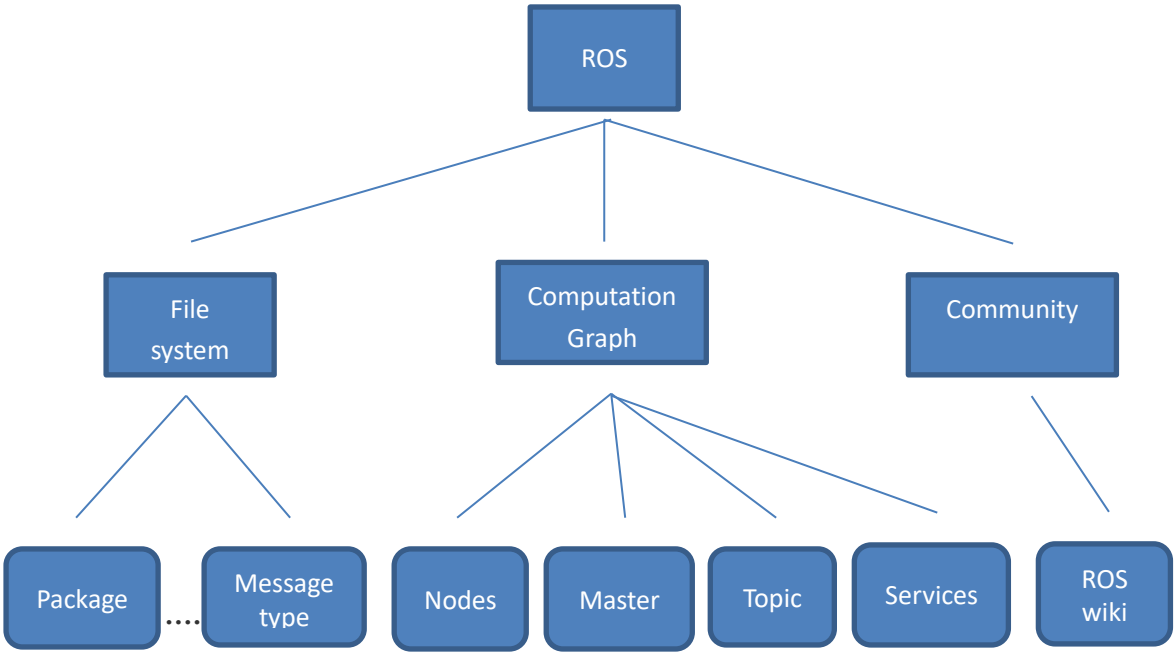


Figure 1. Overview diagram showing flow of control as it relates to the aims of the proposed research.

Based on our project, we need to use the ROS(Robot operate System), which contain Gazebo7. So I will introduce them in detail.

1: General structure of ROS



F2: structure of ROS

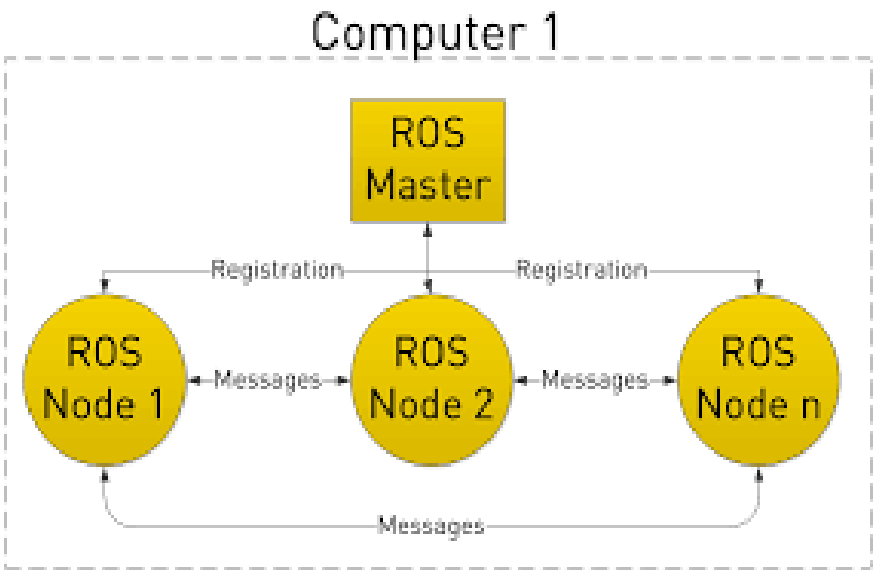
ROS has three levels of concepts: the **Filesystem level, the Computation Graph level, and the Community level**. Filesystem contains Packages, Metapackages, package Manifests, Repositories, Message types, Service types. Graph level contains Nodes, Master, Parameter Server, Messages, Services, Topics, and bags. Community level contains Distribution, Repositories, Ros Wiki. We use the File system and computation graph a lot. Because we use node to represent the drone, and also we need to know how these drone communicate to each other by topic. I will introduce it as blew.

2: Nodes in ROS

2.1 How the nodes communicate

These are lots nodes in the graph level of ROS, such as laser range

finder node, wheel motors node, localization, path planning, graphical view, image processing, camera node. Also these nodes are used for creating a drone often. All objects in gazebo are nodes, which can call the object from the system or we can build our own node. When we call or build the node, which registered already on Master.

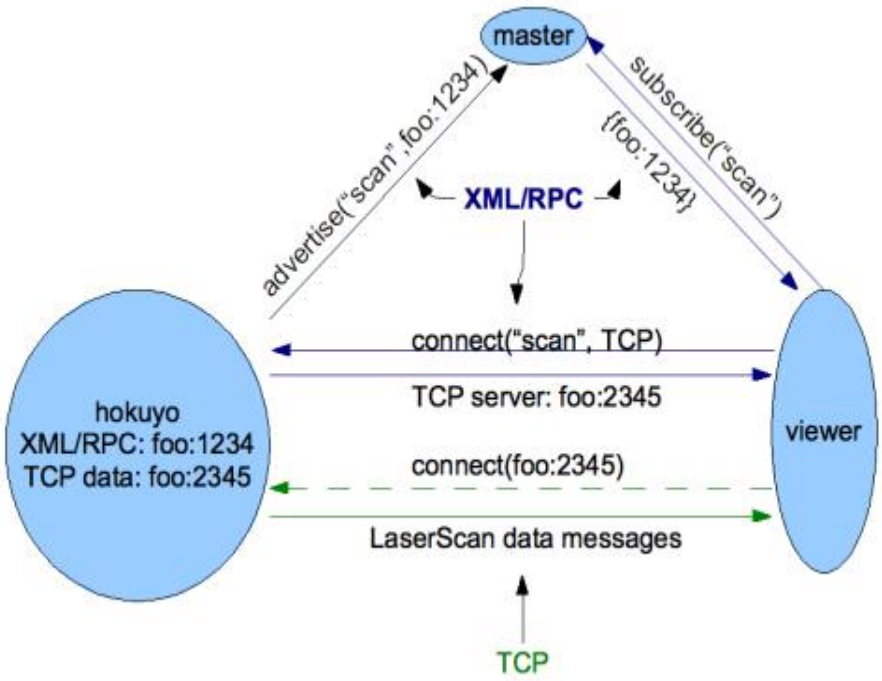


F3: Connection of node and master.

These are lots nodes in the graph level of ROS, such as laser range finder node, wheel motors node, localization, path planning, graphical view, image processing, camera node. Also these nodes are used for creating a drone often. All objects in gazebo are nodes, which can call the object from the system or we can build our own node. When we call or build the node, which registered already on Master, which can manage lots of nodes and recognize the node is exist or not.

Nodes are combined together into a graph and communicate with one

another using streaming **topics, RPC services, and the Parameter Server.** We are going to use topic method to communicate with one another.

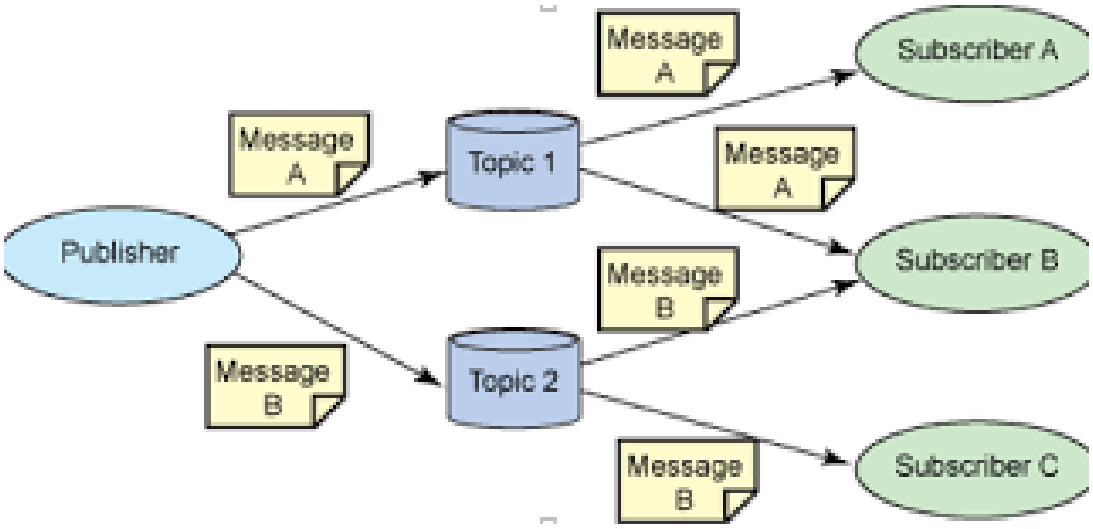


F4: Example of Connection of node and master.

This example describes Hokuyo node, which talks to laser and publishes snsor_msga/laserscan messages on the scan topic. After subscription, the rviz node begins receiving LaserScan messages, which it renders to the screen. All the hokuyo_node does is publish scans, without knowledge of whether anyone is subscribed. All the rviz does is subscribe to Scans, without knowledge of whether anyone is publishing them. The two nodes can be started, killed and restarted, in any order, without inducing any error conditions.

How do the laser_viewer and hokuyo_node nodes find each other? They use **a name service** that is provided by a special node called the **master**, The Master has a well-known XMLRPC URI that is **accessible to all nodes**. Before publishing on a topic for the first time, a node advertises its intent to publish on that topic. **This advertisement sends to the master, via XMLRPC information about the publication, including the message type, the topic name, and the publishing node's URI,** The master maintains this information is a publisher table.

I have another diagram to show the detail for how the nodes communicate with one other.



F5: detail for node to communicate with one other

When a node subscribes to a topic, it communicates with the master, via XMLRPC, sending the same information (message type, topic name, and node URI). The master maintains this information in a subscriber table. In return, the subscriber is given the current list of publisher URIs.

The subscriber will also receive updates from the master as the list of publishers' changes. Given the list of publishers, the subscribing node is ready to initiate transport-specific connections.

Message data does not flow through the master. It only provides name service, connecting subscribers with publishers.

2.2 Establishing a topic connection

Putting it all together, the sequence by which two nodes begin exchanging messages is:

1. Subscriber starts. It reads its command-line remapping arguments to resolve which topic name it will use. (Remapping Arguments)
2. Publisher starts. It reads its command-line remapping arguments to resolve which topic name it will use. (Remapping Arguments)
3. Subscriber registers with the Master. (XMLRPC)
4. Publisher registers with the Master. (XMLRPC)
5. Master informs Subscriber of new Publisher. (XMLRPC)
6. Subscriber contacts Publisher to request a topic connection and negotiate the transport protocol. (XMLRPC)
7. Publisher sends Subscriber the settings for the selected transport protocol. (XMLRPC)

8. Subscriber connects to Publisher using the selected transport protocol. (TCPROS, etc...)

3： Multi - robots communication

Robot needs many nodes: laser range finder node, wheel motors node, localization, path planning, graphical view, image processing, camera node and so on. So when we know how these nodes communicate with one other, then we will know how robots communicate with one other well.

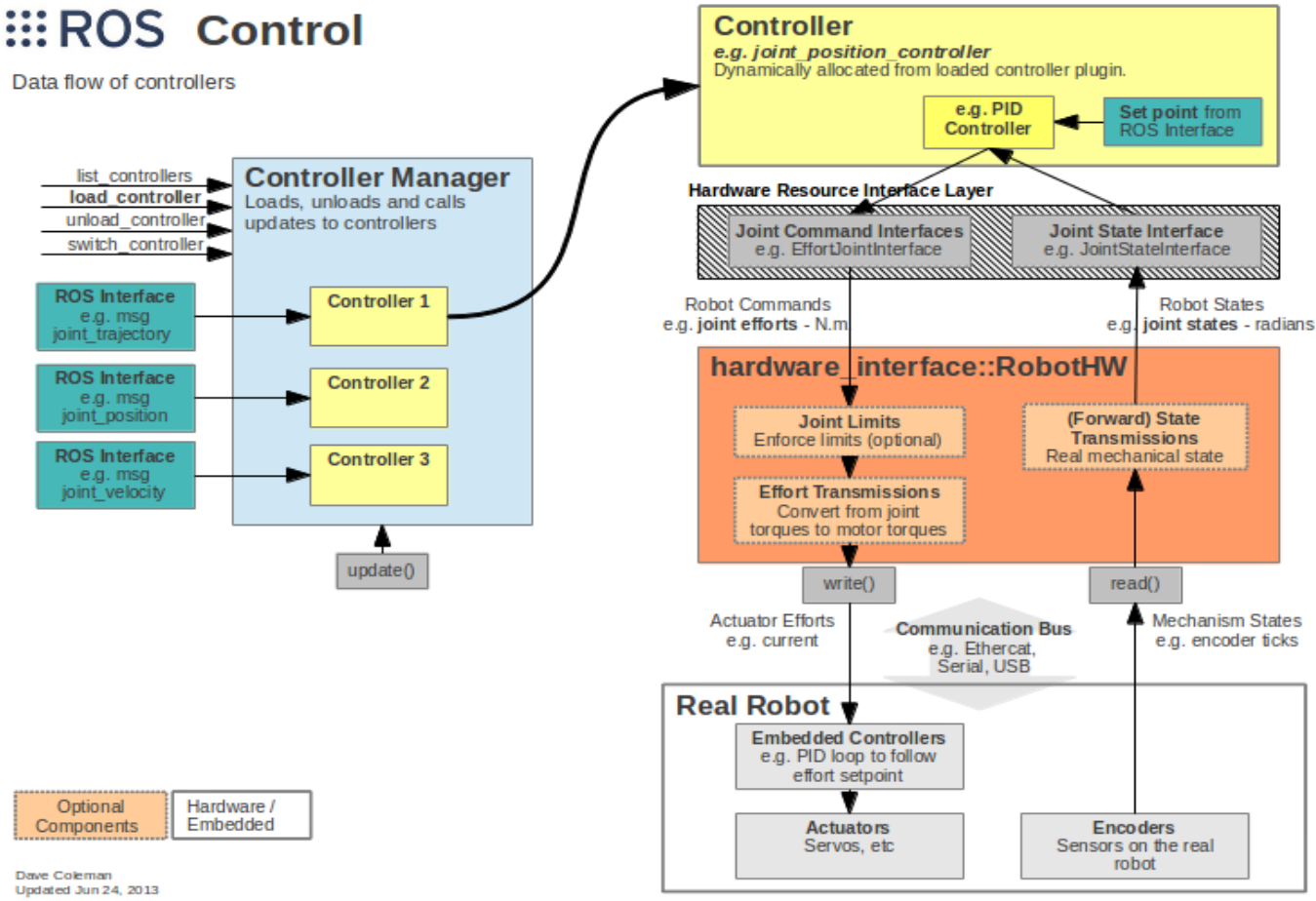
Package: **hector_quadrotor, hector_localization, hector_gazebo and hector_models**

4: User and robots

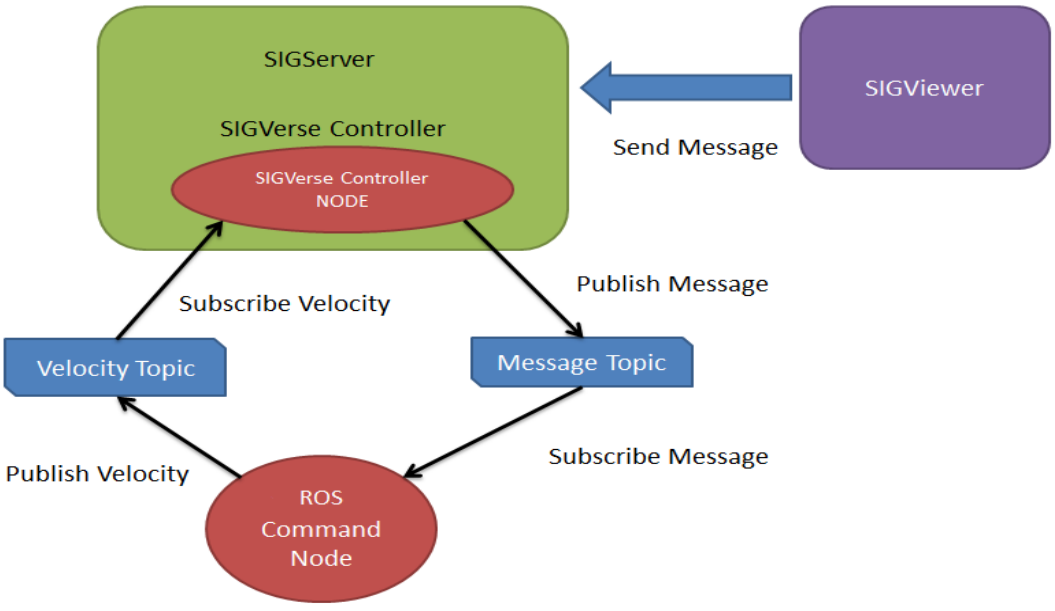
Controller manager provides a hard-realtime-compatible loop to control a robot mechanism. When loading a controller, the controller_manager will use the controller name as the root for all controller specific parameters, most importantly, type which identifies which plugin to load. To interact with controllers form another ROS node, the controller manager provides five service calls:

```
controller_manager/load_controller,controller_manager/unload_controller,controller_manager/switch_controller,controller_manager/list_controllers,controller_manager/reload_controller_libraries,
```

```
controller_manager/list_controller_types
```



F7: ROS control.



F8: Example of controller communicates with node.