Documentation for the HiJac Project

The HiJac project aims to implement the leadership election protocol in SCJ and then use SCJ to control three Lego EV3 mindstorms robots. The leadership election protocol elects a leader from one of several identical robots. Once elected, the leader is responsible for issuing commands to its followers.

To achieve this goal, we have built up a software architecture with five layers. Among these layers, there are two fundamental layers that provides the control of the EV3 robot and network connection. Based on these two fundamental layers, the election protocol layer is introduced to provide the primitive election operations and an EV3 command layer is provided to support for the interactions between the robots. At last, an application layer is defined at the top of this architecture to allow programmers to build their own SCJ-EV3 leadership election programs and control the robot actions. In this project, we provide two demo programs in the application layer to illustrate the project. The overall structure of the HiJac project is shown in Figure 1.

Any applications that include the protocol essentially consists of two independent parts. The first part establishes and maintains the leader/follower relationship, which also include the maintenance of the network connection. The second part consists of the logic to control the robot.

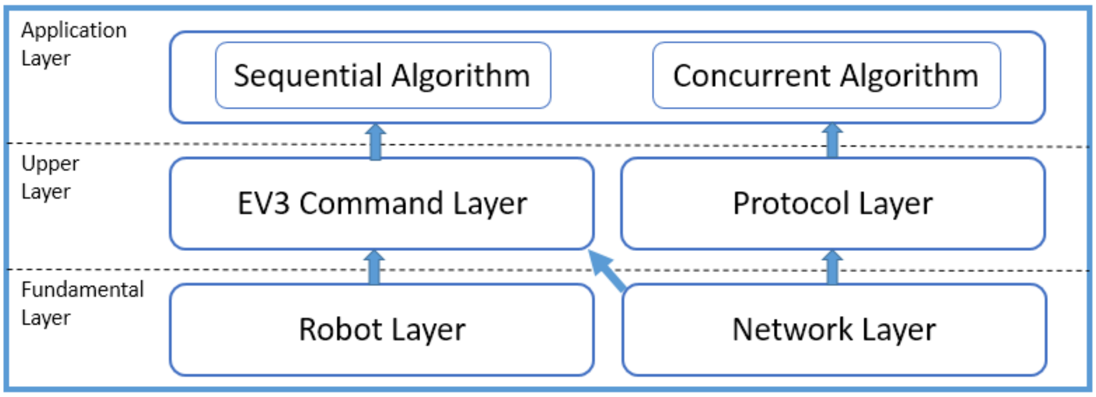

Figure 1: The overall structure of the HiJac project

Two fundamental Layers

The Robot Layer (package *devices.ev3*):

The robot layer is implemented by the icecap team. It provides a set of interface to control the motors or sensors of the Lego EV3 robot. In addition, interfaces are provided to detect whether the buttons on the EV3 brick has been pressed.

The Network Layer (package *com*)

The network layer is implemented by POSIX sockets through JNI. We provide two types of socket: TCP socket and UDP socket. The TCP sockets are used for point-to-point communications while the UDP sockets are for broadcast communications. The TCP socket can be regarded as the "reliable network", as the messages received will be the same order as the remote node sent. As for UDP sockets, it cannot guarantee such order. Hence, can be viewed as the unreliable network. In our project, the TCP sockets will be used for the messages exchange (i.e. the states of robots) while the EV3 commands will be sent and received via broadcast through UDP sockets.

Two Upper Layers

The Protocol Layer (package *leadershipElection*)

The protocol layer implements the body of the leadership election protocol. It maintains an array which stores all the connected robots' information, which includes the state, the petition and the robot's id. The protocol layer also maintains a timer, which will be used to indicate whether the stored information of a specific robot is up-to-date (i.e. whether the robot is still online).

The protocol layer provides two important interfaces: *electLeader* and *collect*. Typically, the method *collect* will be invoked after receiving the state from a remote robot. It will store the robot current information with the timer mentioned above. When the *electLeader* method is called, the state of the local robot will be decided based on the current information of all the connected remote robots (i.e. the information collected by the *collect* method).

The EV3 command Layer (package *devices.ev3.support*)

This Layer is built on the top of the fundamental layers and provides the functionality of sending and executing EV3 commands between robots. The reason for building this layer is to enhance the leadership election. Once a leader is elected, it can use this layer to send EV3 commands to other robots, who should be followers. Meanwhile, the follower robots can receive and execute the command through the interface *getCommand* and *action*.

The Application Layer (package *test.ev3.leaderShipElectionCommunicationBasedAction*)

Two demo applications are provided in the Application Layer to illustrate two major options to implement the leadership election application. In these demos, the TCP sockets are used to exchange states between robots while the UDP socket are used to send and receive EV3 commands.

The Sequential Leadership Election Application (*TCPIPLeaderElectionSequentialCB*.class):

In the sequential version, the whole election procedure is executed sequentially by a single periodic event handler (Elector). Besides, a managed thread (Listener) and several (depends on the total number of robots) periodic event handlers (Connector) are introduced to handle to network connection (i.e. listen and accept connection as well as try to connect to other robots). The sequential algorithm structure is shown in Figure 2.

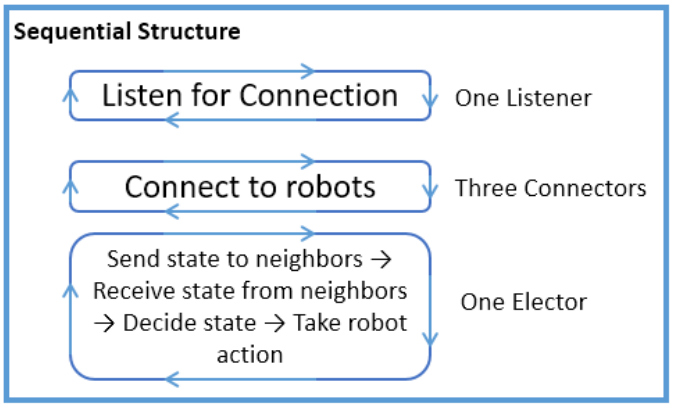

Figure 2: The sequential algorithm structure

When the program starts, the "listener" and "connectors" will try to establish the network connection between each robot. The “elector” has a longer period. When it is released, it will first send its current state to all the connected robots and then try to receive the states from them. Then it performs the election to decide its right states and the take the corresponding action:

- If it is a leader, it will make a movement and then send this movement as an EV3 command to other robots.
- If it is a follower, it will try to receive an EV3 command from the leader and then execute it.
- If its state is undecided, it will do nothing.

The Concurrent Leadership Election Application (*TCPIPLeaderElectionMultiThreadsCB*.class):

The concurrent version splits the election procedure into several independent tasks and executes them asynchronously. This version contains a listener (managed thread) that listening for connections, several receivers (aperiodic event handlers) to receive the state from each remote robot, several senders (periodic event handlers) to connect to the remote robot and send its current state, a elector (periodic event handler) to decide the current state of the robot and two actors (one periodic event handler and one managed thread) to control the robot behavior for different states.

Once the program is started, the listeners, receivers, and senders will try to connect each robot and send as well as receive states between each robot. When the elector is released, it will decide the robot’s state based on the current states received. As with the elector, the robot actor will also take the corresponding actions based on the current state of the robot. The robot behavior is the same as that of the sequential version. Figure 3 shows the structure of the concurrent algorithm.

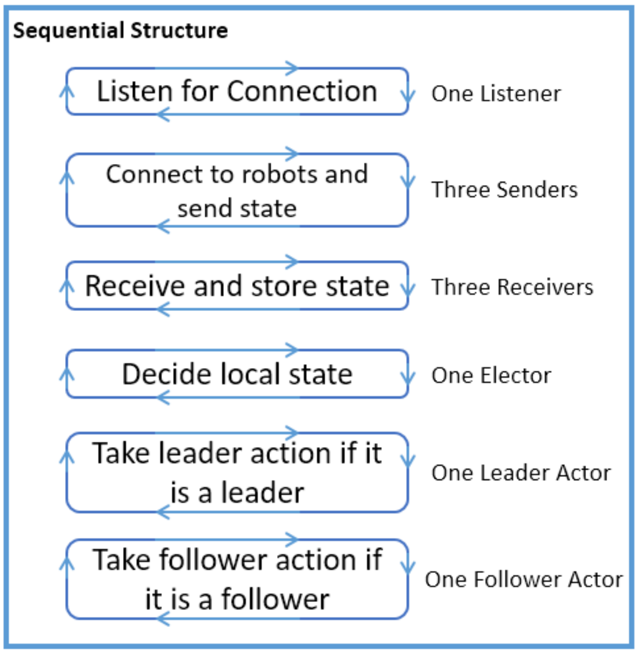

Figure 3: The structure of the concurrent algorithm.