ILR 07 – Progress Review 8

Dorothy Kirlew

Team Daedalus Members: Mohak Bhardwaj, Shivam Gautam, Pranav Maheshwari, and Richa Varma

February 10, 2016

1. Individual Progress

For the progress review on February 10, I worked with Richa to complete the communication subsystem. We modified the routine definition to be more robust, and clearly defined the protocol for the ROS messages sent to and from the XBees. Finally, we coded and tested the entire communication subsystem in a variety of scenarios using three XBees on separate machines.

i. Routine Definition Modifications

The first change Richa and I made was in the communication initiation sequence (Figure 1: XBee Start Flow). We changed the vehicle ID of the first vehicle in the parking lot to be 5 instead of 1. The reason for this is the concept of "last in queue". When a vehicle enters the parking lot, it sends a HELLO XBee message. The vehicle that is last in queue (liq), that is, the vehicle with the greatest ID, will respond with an INTRO XBee message. However, since adding the concept of Virtual Vehicles, we need to ensure that a Virtual Vehicle will not be expected to respond to the HELLO message, as it is completely virtual and does not have an XBee attached to it. Thus, we changed the initial vehicle ID to 5, which leaves four spots (IDs 1-4) free for the virtual vehicles to claim. Naturally, this can change as we begin to test more complex scenarios that demand more virtual vehicles.

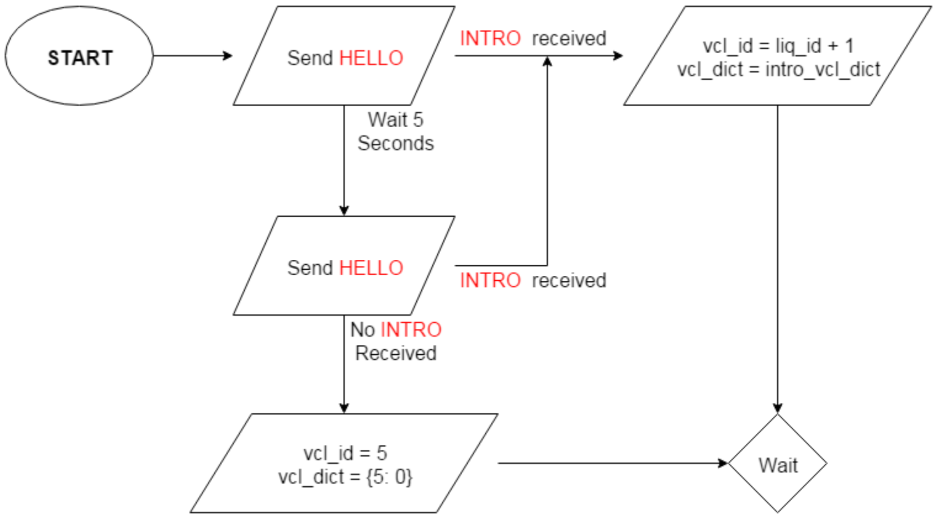

Figure 1: XBee Start Flow

Richa and I created more message types that needed to be sent between XBees (Figure 2: XBee Message Flow) and over ROS (Figure 3: ROS Message Flow). This was again due to incorporating the UI into the communication system. Because the UI needs to know the state of the vehicles as well as their destination, we added a PARKED XBee message. When a vehicle reaches its destination, the locomotion node alerts the master node, which passes this message to the XBee. The XBee will then communicate this to the other XBees in the network, which will alert the UI of this state change. Thus, "Parked" and "Returned" ROS messages were created to pass these state changes along to the communication subsystem.

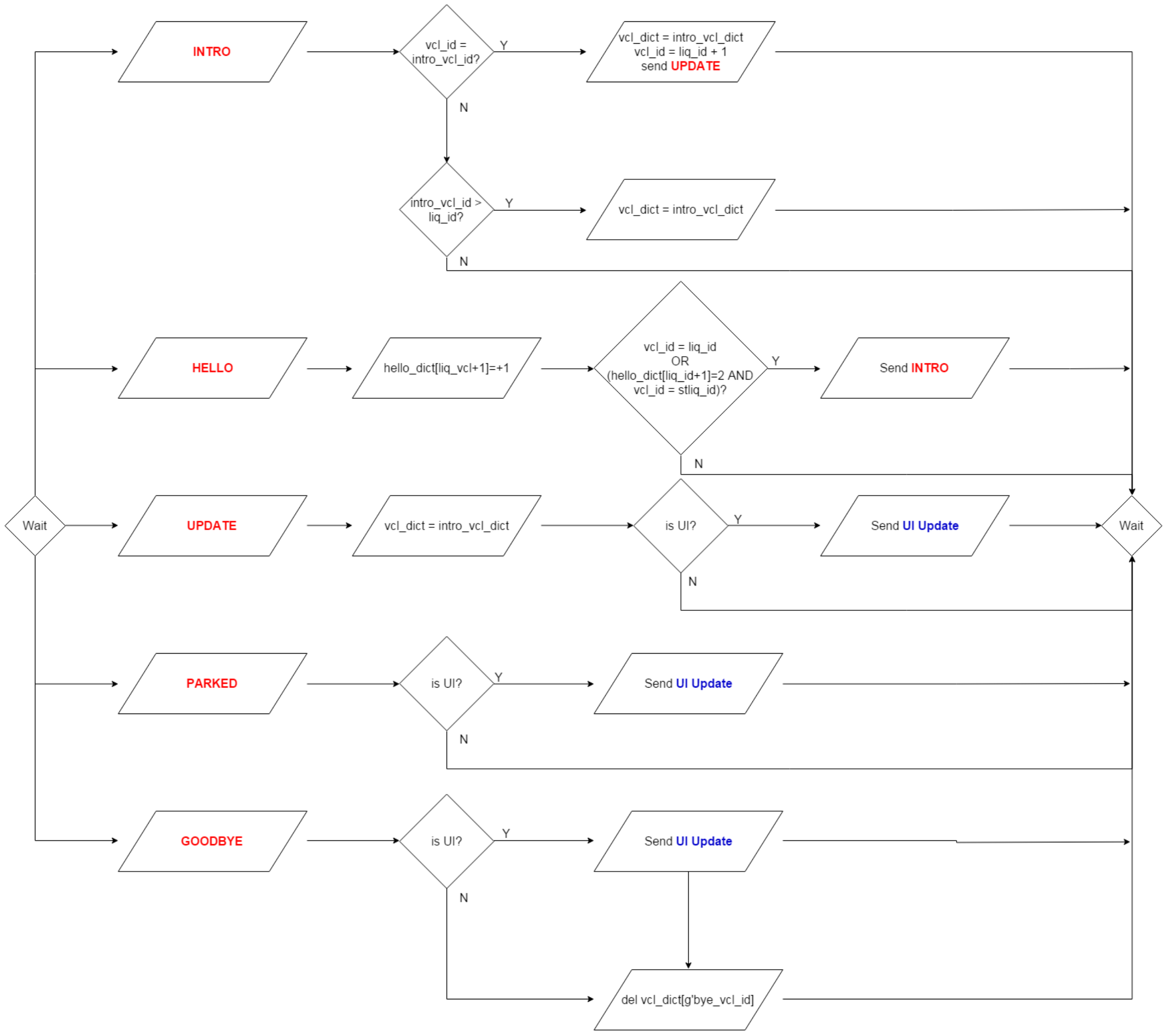

Figure 2: XBee Message Flow

The final messages that were created are between the UI and the XBee. Pranav has created the UI to have the capability of adding and removing virtual vehicles from the parking lot. Naturally, this information has to be communicated to the other vehicles. When a vehicle is added or removed via the UI, the UI sends a String ROS message to the XBee containing the vehicle ID and the spot ID, separated by a comma. If the vehicle is in the entry queue, the spot ID is 0. If it is exiting, the spot ID is 25. Otherwise, the spot ID is between 1 and 24. The UI also needs to know the states of the other vehicles in the parking lot. When the XBee with the UI receives certain messages, such as a PARKED or GOODBYE XBee message, it will pass these messages to the UI. Richa and I created a custom ROS message that contains the vehicle status, vehicle ID, and spot ID. The UI will interpret this data and display it accordingly.

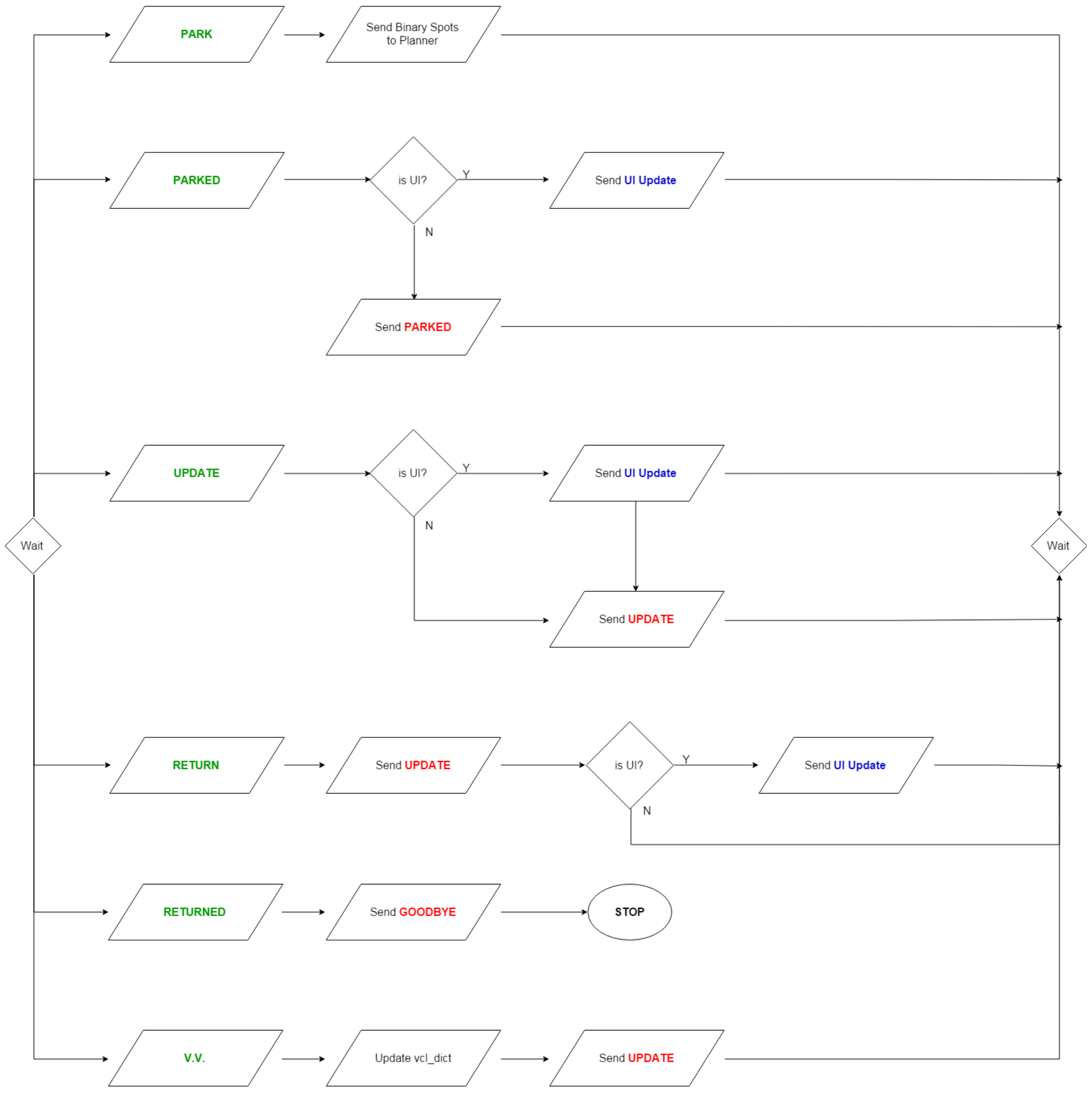

Figure 3: ROS Message Flow

ii. XBee Serial Protocol

The communication system is responsible for handling the interactions, not just between multiple subsystems but also between multiple vehicles. It is very important that the messages being passed be in a known and consistent format. Richa and I clearly defined the protocol for the ROS messages passed between the XBee and other subsystems, as seen below.

Sent to XBees

Master

- **Purpose**: Park Command
  - **Data Type**: String (lowercase)

        - **Example**: “park”
- **Purpose**: Parked Status
    - **Data Type**: String
        - **Example**: “parked”
- **Purpose**: Return Command
    - **Data Type**: String
        - **Example**: “return”
- **Purpose**: Returned Status
    - **Data Type**: String
        - **Example**: “returned”

Planner
- **Purpose**: Spot Chosen
    - **Data Type**: String - SpotNumber
        - **Example**: “5”

UI
- **Purpose**: Virtual Vehicle Update
    - **Data Type**: String - VehicleID,SpotNumber
        - **Example**: “2,17”

Sent by XBees

Planner
- **Purpose**: Binary Spots
    - **Data Type**: Binary List Length 24
        - **Example**: 0,1,0,0,1,0,1,0,0,0,0,0,0,1,0,0,0,1,0,0,0,0,0,0

UI
- **Purpose**: Vehicle State Updates
    - **Data Type**: ROS Custom Message – String Vehicle Status (in_queue, parking, parked, returning, returned), uint16 Vehicle ID, uint16 Spot ID]
        - **Example**: “Parking, 6, 2”

Navigation
- **Purpose**: Exit Coordinates
    - **Data Type**: String - X-Coordinate, Y-Coordinate (same format as what the Planner sends with parking spot coordinates)
        - **Example**: “5.23,7.81”

2. Challenges

As Richa and I worked our way through coding the routines, we encountered some scenarios that we had not thought of yet. Through slight modifications in the code, such as adding a “PARKED” XBee message, we were able to overcome these difficulties. There was the additional problem of testing on three different XBees. To fully simulate how the final communication system will operate, we tested from two laptops with one XBee each and one SBC with an XBee. We were unable to procure more SBCs because they had been lent to other teams for them to use for testing.

3. Teamwork

Shivam and Pranav completed the obstacle detection subsystem and CAD’d and printed mounts for the Parallax Laser Range Finder. Pranav also worked on the User Interface that will be used to model the state of the parking lot and simulate virtual vehicles entering, parking, and leaving. Mohak completed the multi-agent planner that will be used to determine the optimal parking spot for each vehicle. Richa and I improved the routine definition for the communication subsystem, as well as coded and tested it.

4. Future Plans

A new platform arrived yesterday, which will hopefully solve the Xtion problems that the team was facing. This will enable us to map areas, as well as design and fabricate the mock parking lot. Because the other subsystems are complete, we will begin integrating subsystems two at a time. The key subsystems to integrate are the Master node, the communication subsystem, and the UI. Both the master node and the communication subsystem interact with several other subsystems and have the potential to be very difficult to integrate. The UI will be used as a tester and debugger for multiple scenarios, so it is important for it to be integrated with the system as soon as possible.