Simulation Overview

The simulation position is new this year, created to have a group member focus on designing a simulation for both the rover and the ARM processor. The intent is for the simulation to serve as a tool for the rest of the group as they are designing their respective components, including the ARM processor, rover sensors, and rover motors.

It was my goal to make the simulation as transparent as possible, acting as a complete system as seen by the attached component. This would entail processing data based on the information received from the attached component, receiving and transmitting data with correct message formatting and protocol, and responding with data that is accurate to what would be produced by the physical component it is simulating. In addition, I wanted to create an intuitive user interface, fully featured with control buttons, a display of data flow, and a display of the physical world. This would include a map of the physical environment (provided by the professor,) position of the rover within the map, and a display of live and past sensor data.

WiFly communication format

0xFE	Message Index #	Message Type	Payload	0xFF

Header byte: 0xFE
Message Index #: Incremented #, 0-253, loops upon overflow.
Message Type: Motor command, Motor command acknowledge, moving status update, done moving, sensor data
Payload: varying lengths depending on message type
Tail byte: 0xFF

WiFly communication protocol

The protocol for communication between the ARM WiFly and the rover WiFly was carefully designed to provide accurate data transfers as well as a robust missed message/failure handler.

ARM PIC

A message is formed and written to the WiFly which sends the bytes to the other paired WiFly. When a byte is received, it is put into a message buffer. When the header byte, 0xFE, is received, the message buffer is cleared. This clears out any garbage data that may be stored in the buffer, including data from pairing, data from the last message, or corrupted data. When the tail byte, 0xFF, is received, all of the data in the buffer is analyzed.

At this point, one of three paths is taken:

First path: message index of incoming acknowledge message is not a match to the message index that was just sent. This could occur for a few reasons; mainly the message was not processed correctly on the rover or the message was corrupted in either the transfer from ARM to rover or rover to ARM. No matter the cause, send the exact same message again.

Second path: no acknowledge message is received. The ARM PIC determines this when twice the time required for this process or more has passed with no acknowledge message received. In this case, send the exact same message again.

Third path: message index of incoming acknowledge message is a match to the message index that was just sent. This means that the rover has successfully received the message. The ARM PIC is now ready to process the received data and send a new message.

There are four message types that are recognized: 0x33 (motor command acknowledge), 0x34 (moving status update), 0x35 (done moving), and 0xA (sensor data).

Depending on which message type is received, the ARM PIC will know how long the payload length will be: 0x33 = 1 byte, 0x34 = 0 bytes, 0x35 = 0 bytes, 0xA = 21 bytes.

Rover Master PIC

When a byte is received, it is put into a message buffer. When the header byte, 0xFE, is received, the message buffer is cleared. This clears out any garbage data that may be stored in the buffer, including data from pairing, data from the last message, or corrupted data. When the tail byte, 0xFF, is received, all of the data in the buffer is analyzed.

Currently, we only have one recognized message type, but this message format allows for future updates to add more message types. 0x32 represents a motor command. Immediately, an acknowledge message is sent back to the ARM.

Finally, the rover checks to see if the instruction has been run yet (in the last ten received messages) based on its message index number. If it has already been run, throw out that instruction. If it is a new instruction, execute it.

Reasoning

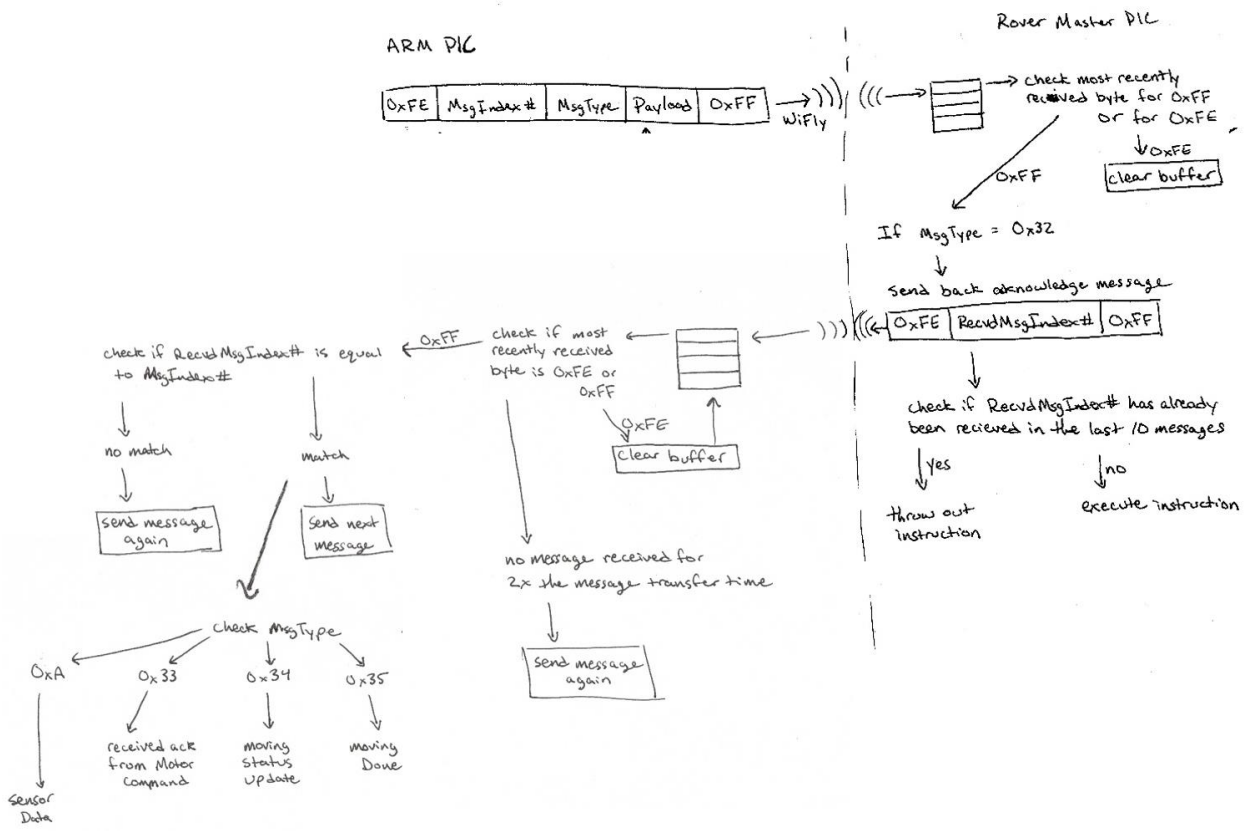
I chose to use this protocol because it provides good protection against missed and corrupt messages.

If a message is lost on the way from the ARM PIC to the rover, the rover will never process it, and the ARM will time out and send it again. This will continue to happen until the rover receives the message and acknowledges the message.

If the acknowledge message is lost on the way from the rover to the ARM PIC, then the ARM still times out and sends the message again. The rover will receive it, but know that it has already executed that command, and throw it out. It will, however, still send an acknowledge message to alert the ARM of a successful transfer.

If the data is corrupt or not what the ARM expected (Information exchange during pairing), then analysis will not be triggered and the data will stay in the message buffer until a clear byte from a real message is sent to clear out the message buffer.

We decided that if a sensor message is missed by the ARM PIC, it is not very important and does not require a resend. This is because it will receive a new set of sensor data very soon which will render the missed data obsolete.



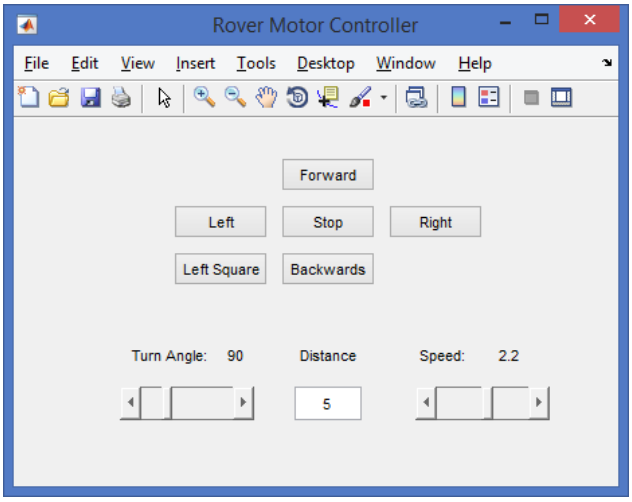
ARM Simulation

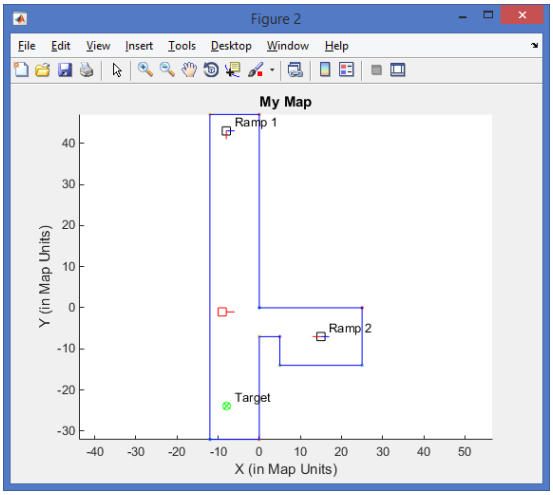
Controls

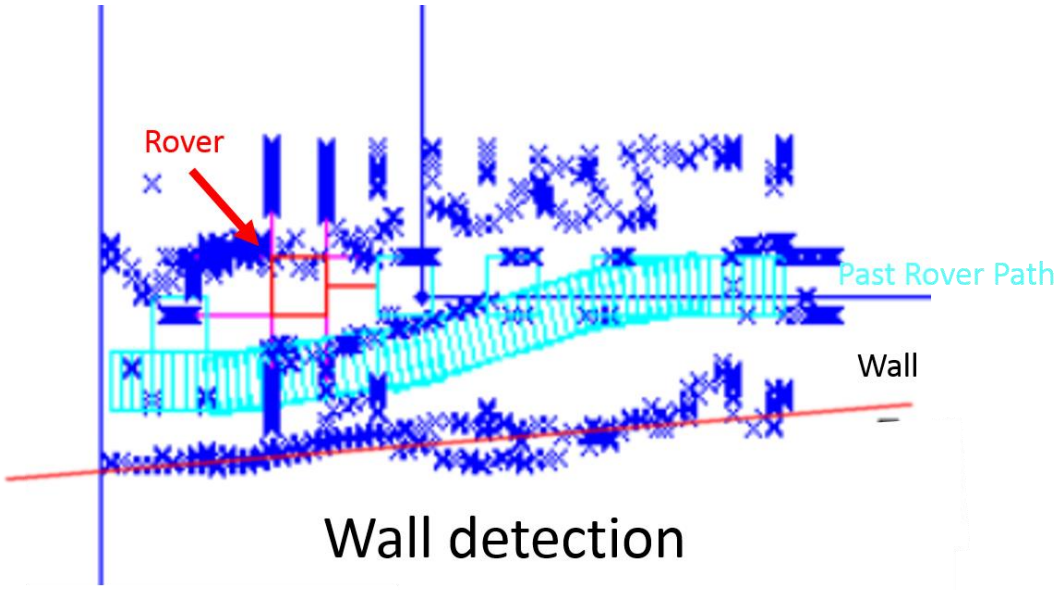
Controls were presented to the user as a GUI with buttons for forward, backward, left, right, left loop, and stop. There were sliders available for choosing turn angle, distance, and speed. When the user selects a command, the turn angle, distance, and speed are sent as the values currently displayed.

Mapping

The map, as well as obstacles and rover, are drawn on another GUI. The rover location is updated in real time* as the rover drives around, including vertical/lateral position and angle. Received sensor information is also drawn in real time. A line protruding from the rover at the sensor length is drawn to show current readings, and a blue x is drawn at the intersection point to show the edge of the physical object. This is very helpful when driving along a wall to show current readings, but also draw out the wall. It will also draw out any obstacles that it finds. Finally, past locations of the rover and colored in light blue to show it's driving path.







Sensor Simulation

The sensor simulation, as it stands, does not fully implement all of the rover capabilities. It replicates all messages exchanged with the ARM PIC, and can simulate moving in real time. However, the sensor data that is generated is centered around a pre-set point, and varies a given amount (simulating noise and unclear readings). The values are not updated according to its surroundings when the rover is moving.

*How it was originally planned and coded. Due to timing restraints and progress of other components, this was changed to a simpler trigger design to still function as designed given less data from the rover. Specifically, instead of redrawing the rover on the *done moving* message from the rover, it triggers on the motor message acknowledge. This redraws the rover when the message is first received by the rover instead of when the rover completes the instruction. This still allows for live updates to the user,

however does not show the location of the rover as it is moving. Status updates are not used to show smooth movement, so the rover jumps from starting position to ending position. The capability is still in the code and will update when the status update is received. However, due to time conditions, those updates are not implemented on the rover end. This can be demonstrated, though, with the full software simulation.