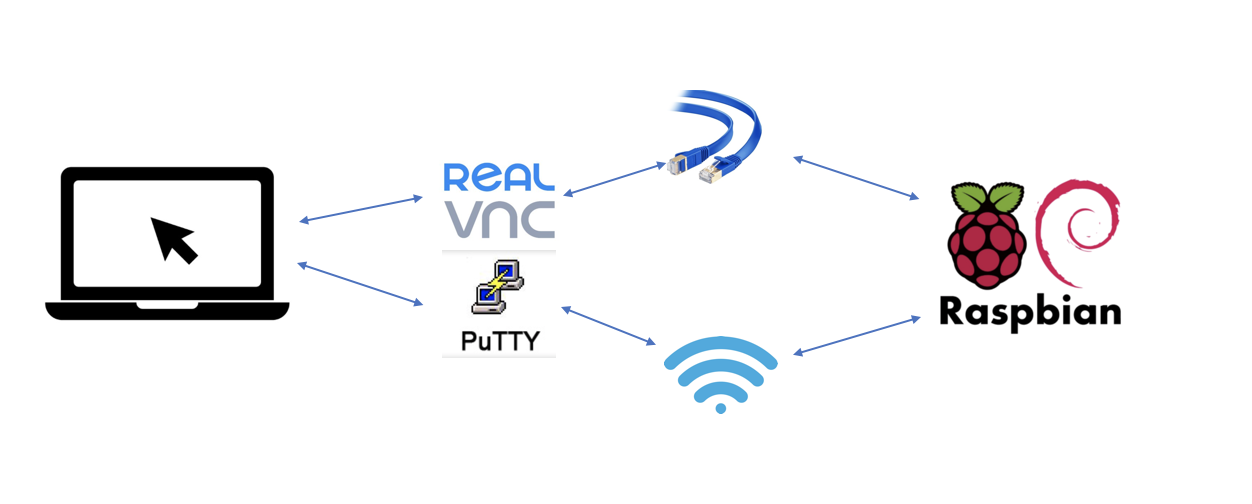
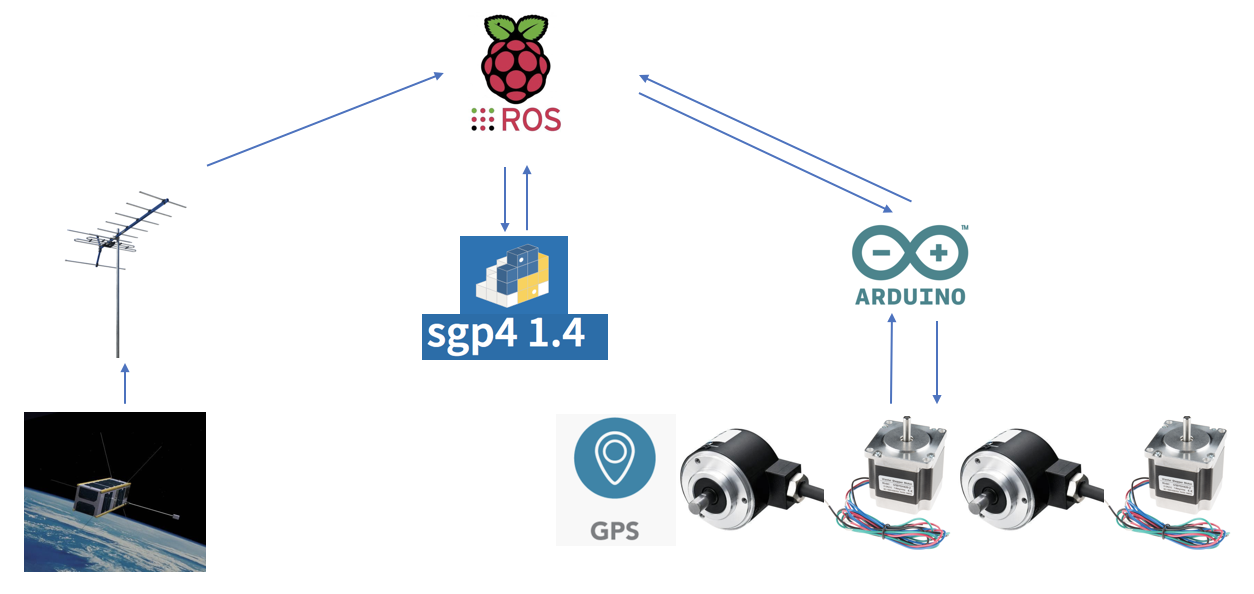
**user interface:**



(I don’t think we have user interface (UI), we don’t need it. The Linux system has its UI). The user interface enable user to have a simple control and set tasks for the ground station. In this project, a convenient method to get remote access to RPI is through Secure Shell (SSH). In this method, the users can access to the Linux control tools including desktop of Raspbian system and Linux console. There are two choices of the telnet clients, PuTTY and REALVNC. SSH connection can be Ethernet or mobile WIFI from laptop. The RPI are set as fix IP address 192.168.137.214 for Ethernet connection, and 192.168.137.245 for WIFI connection.

**system architecture:**



In our software design, the core architecture is based on Robotic operating system (ROS). Robotic operating system (ROS) provides the services from an operating system, including low-level device control, implementation of commonly-used functionality, message-passing between processes, and package management. ROS is language-neutral, and can be programmed in various languages. Kinetic version of ROS is installed on RPI, which can fit the Linux system Raspbian Sketch. There are three main parts: control devices through Arduino board, CubeSat orbit predication through Simplified perturbations models (SGP4 python 1.4), and CubeSat localization. These three parts work as an individual node, and there is a core node named as ROS Master. A node is an executable that uses ROS to communicate with other nodes.

In the first part, the antenna works as publisher. After CubeSat launches, the initial position data sends to RPI as message. The type is a two-line element set (TLE), which is used to calculate and set the angle of antenna.

In the second part, SGP4 1.4 python package receives the initial position data from first node, and calculate the next overhead time and position of CubeSat. When the RPI receives the next overhead position of CubeSat, it will do the calculation to get the intersection angle between ground station and CubeSat.

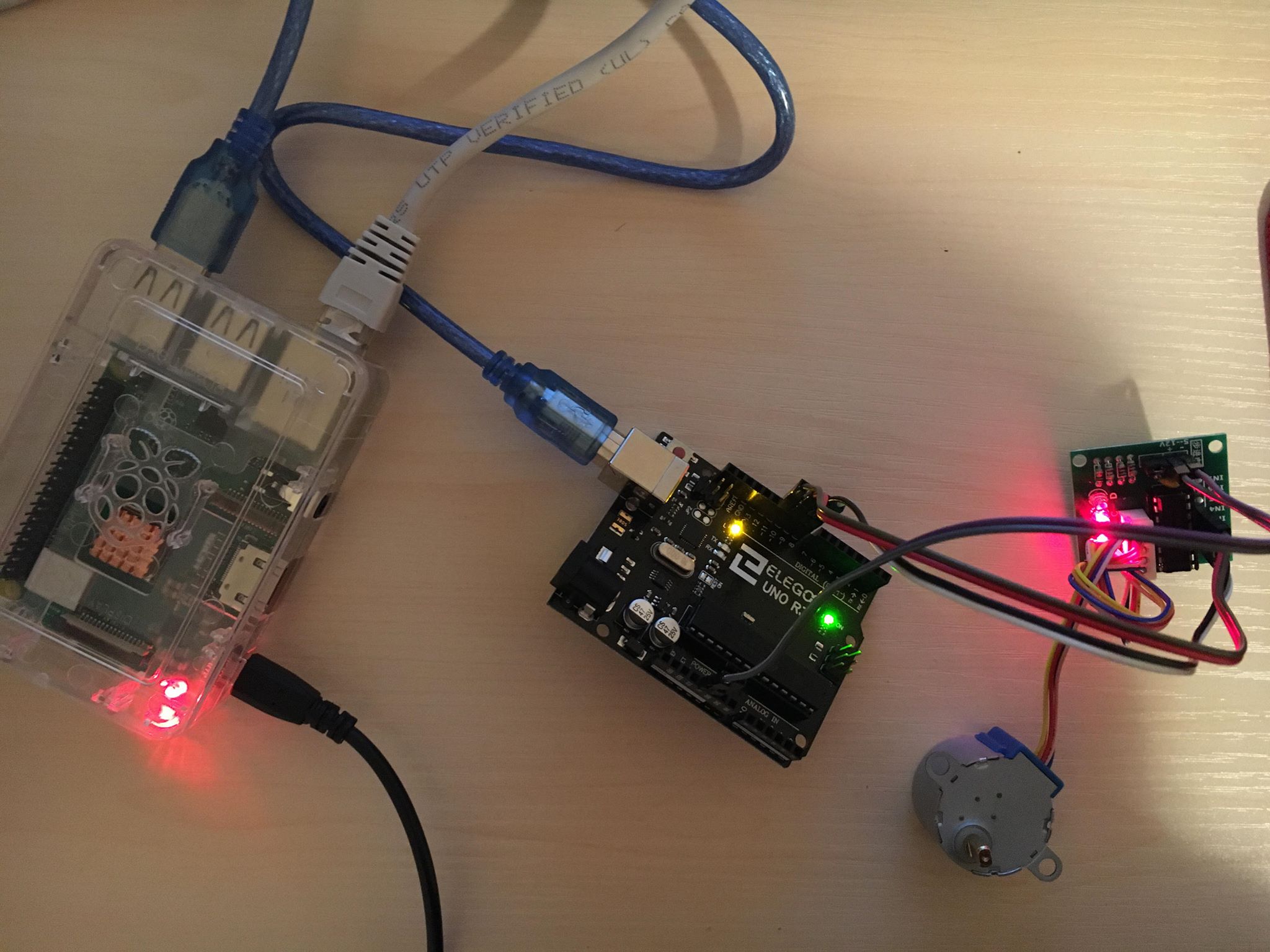
In the last part, the GPS module will run first to get the current position of ground station and send back to ROS master before the other tasks. Then the Arduino board will control the encoder and stepper motor together as an action. The action utilizes ROS topics to send goal messages from a client to the server. While the antenna is moving to the goal position, the ‘percent\_complete’ message periodically transmits feedback values showing the progress in the form of the percentage of the goal point reached. Rosserial needs to install on RPI, which provides a ROS communication protocol that works over the Arduino's UART. In this project, the electronic devices connect to Arduino board instead of RPI, because the Arduino board is a microcontroller motherboard that can simply run one program at a time, over and over again, and only one RPI board cannot support sufficient general-purpose input/output (GPIO) pins to connect these devices. Besides, the Raspberry Pi 3B+ has four USB 2.0 ports; therefore, it can expand the communicate with four Arduino boards.

**Data flow:**



Our primary design of data flow is based on ROS. There are six nodes created for six different tasks. The ROS Master provides naming and registration services to the rest of the nodes in the ROS system. It tracks publishers and subscribers to topics as well as services. The role of the Master is to enable individual ROS nodes to locate one another. Once these nodes have located each other they communicate with each other peer-to-peer. (from <http://wiki.ros.org/Master> ). GPS module (node 2) publishes the position of ground station, and sends this message to Angle conversion (node 4) as subscriber. CubeSat position (node 1) sends the CubeSat position and speed to SGP4 prediction (node 3). After calculation completing, the node 3 publishes the next overhead position to node 4. Angle conversion (node 4) will convert the position of CubeSat to the earth based coordinate system, and then publishes the rotation angles for each stepper motors. Stepper motor (node 5) subscribes the topic from node 4, and rotate to the goal. Encoder (node 6) published the angle value that will be subscribed by node 5 to correct the error.

4. simulation: mimic how the system connected to each other, tell more detail how to do it.



This test mimics a speed control of stepper motor. RPI sends a message to the topic stspeed, and Arduino subscribes the message and control the speed of motor.

Before the test: The Raspberry Pi 3B+ (RPI) is installed Raspbian Sketch system that is highly optimized for the RPI line’s low performance ARM CPUs. ROS kinetic and Rosseial Arduino is installed on this RPI. And we control the RPI through SSH using REALVNC client.

1.Upload the file below to Arduino board.

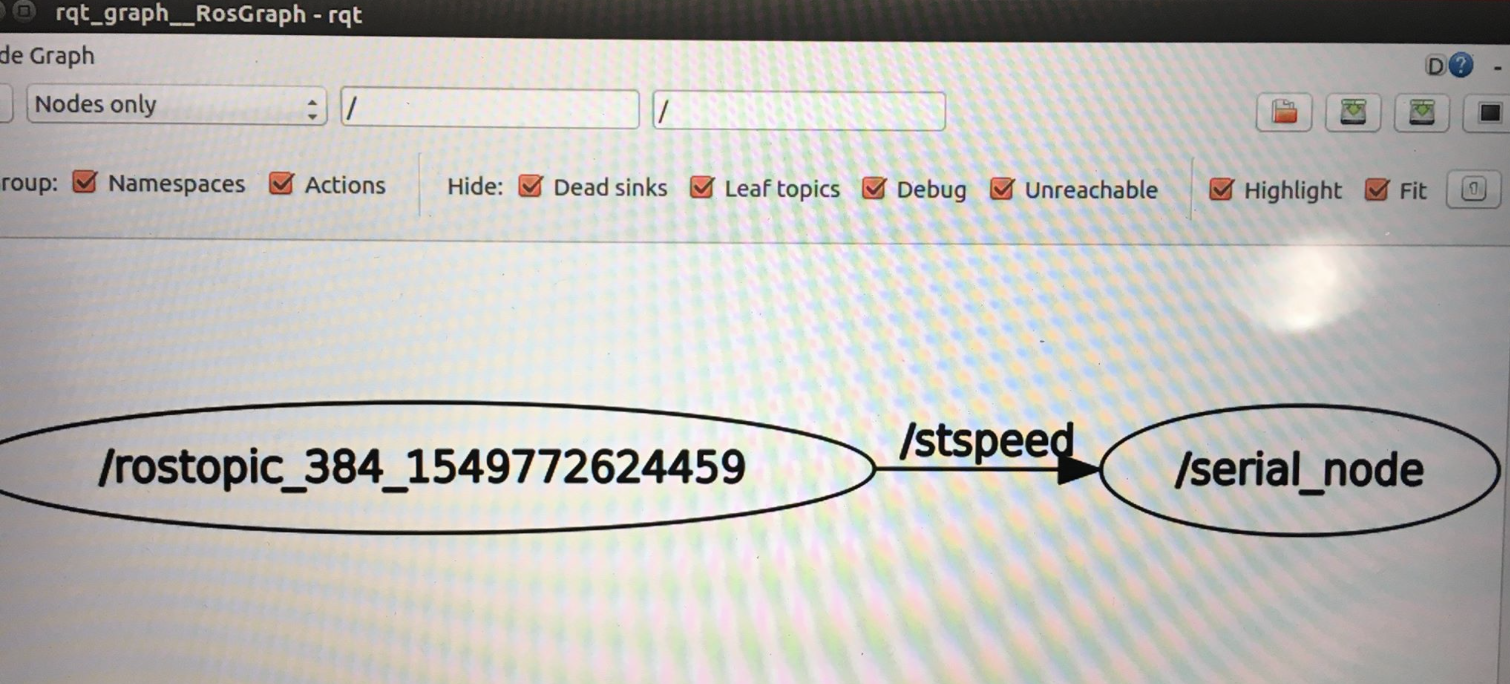
2. Connect the motor in pin 8,9,10,11

2.Run ROS: roscore

3.To connect Arduino with RPI through a USB port. Find the right device port through ls –l /dev/, and then using the command: rosrun rosserial\_python serial\_node.py /dev/ttyACM0. The default baud frequency is 57600 Hz

4. publish a topic: rostopic pub stspeed std\_msgs / Unit 16 (data)

5 run rpt\_graph to see if the node connection is correct.



Result: it is very successful when mimic a part of our ground station software system. We successfully monitor the RPI work through SSH, and run ROS on RPI to communicate with Arduino to control a stepper motor.

