The software controlling the rover will be built entirely in-house by a team of Engineering and Computer Science students. This will take the form of a distributed operating system, in which control is shared across a network of nodes which communicate over serial connections. In contrast to more traditional, centralised models, in which robots are controlled by a single CPU, this system will be more resilient, and less susceptible to the failure of critical components. Our aim is to eliminate single points of failure entirely, and build a robot that will be able to continue operating under extremely hostile conditions. While the nature of the system will make it inherently robust, the level of redundancy can be tailored to the requirements of the robot.

Another high-level goal is to make the operating system scalable and flexible enough to handle a variety of tasks. While commands will initially be sent to the robot via a telemetry system, our design will allow for a greater level of automation to be incorporated in the future. Ultimately the vehicle will be fully autonomous, with the autonomy distributed across several nodes.

The software will take the form of a C++ library, which can be ported to a variety of platforms, allowing us to combine different platforms on the same network and choose the most appropriate device for each node. In our implementation, we intend to use Arduino microcontrollers for sensors and actuators, and Linux machines for more computationally intensive tasks such as computer vision.

Our work to date has been focused on building a reliable communications system for passing information between nodes. To this end, we are implementing a general purpose messaging system, which will allow two-way communication between any pair of nodes in the network. Communication can occur in various forms, including a publish–subscribe model or one-off requests for data, and will allow for remote procedure calling.

An important aspect of the communications system will be its ability to continue passing messages through the network, even in the event that some nodes are no longer operational. For this reason, we intend to use interrupts to handle message passing, so that even if a node has high latency or is no longer able to perform its primary functionality, it will still be able to propagate messages for other nodes through the network reliably and quickly. In the event that a node fails entirely (e.g. due to power loss), the neighbouring nodes will recognise this and find alternate routes for messages. This way if a node drops out, all that is lost is the functionality of that single node, with the minimum possible impact on the overall system.