Development of a Mars Curiosity Rover Simulator

A working model intended for modern space science education and outreach

Prepared by:

Sean Wood
Dept. of Electrical and Electronics Engineering
University of Cape Town

Prepared for:

Professor Peter Martinez
Dept. of Electrical and Electronics Engineering
University of Cape Town

Submitted to the Department of Electrical Engineering at the University of Cape Town in partial fulfilment of the academic requirements for a Bachelor of Science degree in Mechatronics

November 8, 2016

Terms of Reference

Title

Development of a Mars Curiosity Rover Simulator for the Cape Town Science Centre

Description

Our knowledge of the planet Mars has been greatly expanded by several rovers that have landed on the planet over the past twenty years. The most capable of these is the Curiosity Rover, which is currently exploring the surface of Mars. The Cape Town Science Centre has requested the UCT SpaceLab to design and build a model of a Mars exploration rover that will be the centrepiece of a future Mars exhibit at the Centre.

Deliverables

Skills and Requirements

Mechanical Design, Software and Electronics Interfacing and Programming.

Area

Science and Technology

Declaration

1. I know that plagiarism is wrong. Plagiarism is to use another's work and pretend that it is one's own.
2. I have used the IEEE convention for citation and referencing. Each contribution to, and quotation in, this report from the work(s) of other people has been attributed, and has been cited and referenced.
3. This report is my own work.
4. I have not allowed, and will not allow, anyone to copy my work with the intention of passing it off as their own work or part thereof.

Signature:……
Sean Wood

Date:…….

Acknowledgments

Abstract

- Open the **Project Report Template.tex** file and carefully follow the comments (starting with %).
- Process the file with **pdflatex**, using other processors may need you to change some features such as graphics types.
- Note the files included in the **Project Report Template.tex** (with the .tex extension excluded). You can open these files separately and modify their contents or create new ones.
- Contact the latex namual for more features in your document such as equations, subfigures, footnotes, subscripts & superscripts, special characters etc.
- I recommend using the **kile** latex IDE or *TeXstudio*, as they are simple to use.

Contents

List of Figures

List of Tables

Glossary

Abbreviations listed here are used throughout the document.

- MSL - Mars Science Laboratory
- RSVP - Rover/Robot Sequencing and Visualization Program
- RCE - Rover Compute Element
- MEP - Mars Exploration Program
- TMI - trans-Mars injection
- CPU - central processing unit
- MIPS - million instructions per second
- WEB - Warm Electronics Box
- RTOS - real-time operating system
- DSN - Deep Space Network
- DFE - direct from earth
- DTE - direct to earth
- HGA - high-gain antenna
- RLGA - Rover low-gain antenna
- bps - bits per second
- FFL - fixed-focal length
- Mastcam - Mast Camera
- APXS - Alpha Partical X-ray Spectrometer
- MAHLI - Mars Hand Lens Imager
- CheMin - Chemistry and Mineralogy
- SAM - Sample Analysis at Mars
- RAD - Radiation Assessment Detector
- DAN - Dynamic Albedo of Neutrons
- REMS - Rover Environmental Monitoring Station

- MARDI - Mars Descent Imager
- NAC - Narrow Angle Camera
- MAC - Medium Angle Camera
- XRD - X-ray Diffraction
- XRF - X-ray Fluorescence
- SA/SPaH - Sample Acquisition, Sample Processing and Handling
- QMS - Quadrupole Mass Spectrometer
- GC - Gas Chromatograph
- TLS - Tunable Laser Spectrometer
- SMS - sample manipulation system
- CSPL - Chemical Separation and Processing Laboratory
- UVS - Ultraviolet Sensor
- ICU - Instrument Control Unit
- COTS - commercial off-the-shelf
- MMRTG - Multi-Mission Radioisotope Thermoelectric Generator
- CNC - Computer Numerical Controller
- SoC - System on a Chip
- CSI - Camera Serial Interface
- eMMC - embedded Multi-Media Controller
- STL - Standard Tessellation Language
- IoT - Internet of Things
- EVA - Extra Vehicular Activity
- ES6 - ECMAScript 6
- UI - User Interface
- DOM - Document Object Model
- JSON - JavaScript Object Notation
- AJAX - Asynchronous JavaScript and XML
- SPA - Single Page Application
- RTC - Real Time Communications
- RTSP - Real Time Streaming Protocol
- J5 - Johnny-Five
- SDP - Session Description Protocol
- JSON-RCP - JSON encoded Remote Procedure Call protocol
- SPA - Single Page Application

- RoSE - Rover Sequence Editor
- CAD - Computer-Aided Design
- HTTP - Hypertext Transfer Protocol

Chapter 1

Introduction

1.1 Background to the study

A very brief background to your area of research. Start off with a general introduction to the area and then narrow it down to your focus area. Used to set the scene [**?**]. The section should highlight challenges in the study area to put your work in context [1].

1.2 Objectives of this study

1.2.1 Problems to be investigated

Description of the main problem(s) to be solved and/or hypothesis of your work. Questions to be answered in order to confirm the hypothesis or solve the problems are also articulated here.

1.2.2 Purpose of the study

Give the significance of investigating these problems. It must be obvious why you are doing this study and why it is relevant. Contributions of your work should also be given here.

1.3 Scope and Limitations

Scope indicates to the reader what has been and not been included in the study. Limitations tell the reader what factors influenced the study such as sample size, time etc. It is not a section for excuses as to why your project may or may not have worked.

1.4 Plan of development

This section summarizes the methods, tools, techniques and the order of doing things followed in order to accomplish your work. It also includes such planning tools as project Gantt chart, Critical path analysis and mind mapping.

1.5 Report Outline

Here you tell the reader how your report has been organised and what is included in each chapter. You should give a synopsis for each of your chapters here.

I recommend that you write this section last. You can then tailor it to your report.

Chapter 2

Literature Review

2.1 Space Exploration and NASA's Journey to Mars

2.1.1 A Brief History

The human race possesses a trait that proposedly sets us apart from the majority of life forms around us; the powerful will to explore what is unknown. It is the curiosity and the thrill to push past the boundaries of what is thought to be possible, perhaps felt stronger by some, that forms the basis of many scientific endeavours relating to facts of life and existence around and outside of the immediate environment in which we live.

A prime example of such a drive to explore is in the research and exploration of outer space, which, from a technological perspective, transitioned from astronomer's dream to scientist's and engineer's reality during the Cold War. Although space exploration as we know it today is motivated by human curiosity, it was during this period of political tension that significant breakthroughs in spacecraft and rocket propulsion technology were brought about. This period is referred to as the "Space Race" and stemmed from research and development of nuclear weaponry during World War II [2, p. 147]. The race began with the attempted launches of artificially made satellites [3, pp. 3-5] and within the 40 years following the success of the USSR's *Sputnik I* in 1957, the first object to be put into orbit by man, space technology progressed from early manned flights beginning in 1961[1] through the *Apollo 11* lunar flight to having flown by of the majority of the planets in our solar system.

By 1981, the launch of *Columbia* [4], a space shuttle designed to be used for more than one flight, marked the beginning of reusable space technologies answering to the problem of cost and with the forethought of future increase in space flight frequency and demand. Today, the efforts to lower the cost of space travel and the attempt to bring space exploration into the private sectors to make these opportunities more realisable by the public are evident in Elon Musk's SpaceX development of the Falcon 9, a reusable rocket booster stage that returns and lands safely back on the surface of Earth [5].

[1] First human in space, Soviet launched

The National Aeronautics and Space Administration (NASA) of the United States has been and still is responsible for a large chunk of mankind's search among the stars and, with respect to research and exploration, has made great efforts to better understand the planet that we live on in conjunction with the immediate spacial environment around Earth, the solar system and the planets within, and that which lies in deep space. After the Apollo lunar missions, efforts by NASA to explore involved one of the first space stations, the *Skylab*, which suffered technical difficulties originating from launch but proved the ability to conduct research in space as well as allow astronauts to perform repairs and maintenance to artificial bodies in that environment [6]. *Skylab* was followed by the International Space Station (ISS), intended to be a more sustainable microgravity environment in which to conduct research that might require such conditions. Research of this type include a very broad range of investigations from the effects of near-weightlessness on plants and animals through to growth of human-like tissues and protein crystallisation [7]. An area of research that specifically relates to this project is in the development of technology to allow for longer, cheaper and faster flights in space, both in spacecraft materials and systems, and in astronaut health and performance. This is closely coupled with the search by entities around the world for other forms of life outside of Earth's atmosphere fuelled by the prospect of finding environmental architectures similar to ours. One of NASA's goals outlined in [8] is to send humans to Mars and this has lead to enormous amounts of research, promising engineering and technological successes that will ultimately allow humankind to extend civilisation across more than one planet.

2.1.2 Mars

NASA has identified that Mars is a planet with greater similarity in formation and conditions in its history and as a result has been a target of exploration for more than 40 years. This has involved multiple flybys and orbits starting from 1962 through to the first lander, the *Viking 1*, to touch down on the surface of the planet in 1975 [9]. NASA's Jet Propulsion Laboratory (JPL) landed the spacecraft, named *Pathfinder*, that contained the first successful rover vehicle, the *Sojourner*, in 1997 [10]. The purpose of this mission was to prove the possibility of cheaper spacecraft development and the transport of scientific equipment to the planet as well as taking photographs of the red surface, from the surface.

```
**
A short paragraph on current goals by NASA relating to modern Mars exploration
and the need for rovers, to introduce the next section
**
```

2.2 The Mars Science Laboratory and Curiosity

2.2.1 Overview

The Mars Science Laboratory (MSL) is a mission that was launched by NASA to further explore the surface of Mars, one of many orbiter, lander and rover type missions as part of

the Jet Propulsion Laboratory's (JPL[1]) Mars Exploration Program (MEP). The program is structured to work towards a set of goals to ultimately understand and determine the potential for life on Mars [11] by observation of the current climate and geology. The MSL is the latest mission in operation as part of MEP and was intended to span roughly one Martian year after touchdown on Mars. However, it has continued to operate for more than double that amount of time.

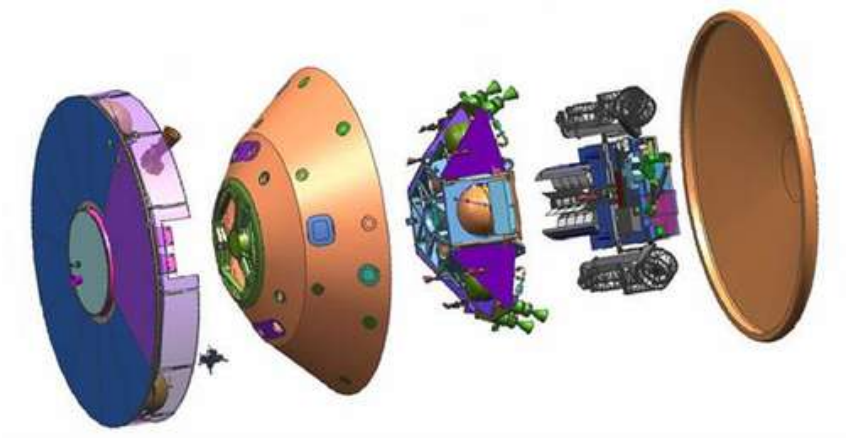


Figure 2.1: An exploded 3D model of the Mars Science Laboratory spacecraft including the cruise stage (far left) and heat shield (far right) [12]

MSL was launched from Cape Canaveral Space Station, Launch Complex 41, atop an Atlas V vehicle, a two stage rocket [13]. The mission required the launch vehicle to insert the five-piece MSL spacecraft into a transfer orbit in a process known as a Trans-Mars Injection (TMI) allowing the spacecraft to arrive at Mars after a 566 million kilometre trip that lasted 256 days. Figure **??** shows a 3D render of the components of the spacecraft that made the trip. Four trajectory correction manoeuvres were made during the flight to result in a landing near "Mount Sharp" in Gale Crater, deemed the most accurate landing on Mars of any other spacecraft [14].

***
A piece on the choice of a landing site for Curiosity
***

2.2.2 Primary Mission Goals and Objectives

Touching down on the surface of Mars, the MSL had primary objectives tailored to contribute to the four goals as outlined in the MEP. The objectives were carried out by the MSL's flagship component, the Curiosity rover, and consisted of a wide range of biological and geological observations such as to determine the chemical building blocks that exist on the surface including organic carbon compounds, prospective historical biological activity, atmospheric processes of evolution, surface radiation and state and distribution of water [15].

Apart from the primary objectives, the MSL mission pushes further the boundaries of

[1]Jet Propulsion Laboratory of California Institute of Technology

space exploration in that it proved the ability to land heavier vehicles at incredibly precise landing accuracy as well as the achievement of wider surface coverage to collect and observe more diverse samples of the surface of Mars.

2.2.3 Curiosity Technical Breakdown

23% of the MSL spacecraft's total mass of 3.893 metric tonnes was thanks to the missions vehicle, *Curiosity*. The six wheeled, instrument-bearing rover features much improved hardware over previous vehicles along with a multiple systems of new instruments to enable the carrying out of the mission objectives.

The mechanical and technological specifications are broken down in the sections that follow.

Mechanical Structure

Structurally, *Curiosity* comprises of mechanical features and principles borrowed from the previous three rovers, *Sojourner*, *Spirit* and *Opportunity*, however, was made much larger (almost double the size). The reason behind the increase in size was the need for extra volume in which to fit the significantly larger set of scientific instruments, 100 times larger than the suite on *Sojourner* [16].

![Body structure image]

The body of the rover, a shallow, rectangular box, dominates its structural layout and serves as the central feature onto which all others subsystems are mounted. The chassis is also host to some of the rover's scientific instruments as well as the avionics box. The electronics that make up the avionics operate in a warm environment [17], thus requiring the body to provide thermal insulation from the external conditions of Mars, giving it the name the Warm Electronics Box (WEB). The regulation of internal temperature, aided by the use of electrical heaters, is taken care of by a heat rejection system involving a pumped-fluid loop with the source of heat being the power generator, discussed in a section to follow. Thermal regulation also widened the range of potential landing sites with respect to their distance from the equator.

Overall, *Curiosity* was designed to exceed normal standards of mechanical robustness given the fact that hand-on maintenance is not a possibility when operating so far away from Earth. All subsystems on the rover minimised the opportunity for accidental collisions that might result in unfixable damage to the subsystems and thus jeopardy of the entire mission. In addition to the stringent design procedures, complex simulations of the rover's mechanical operation were done in virtual environments which allowed engineers to ensure, as far as possible, the success of the design in the differing environment that is on the surface of Mars.

Manoeuvrability

One of the main similarities between *Curiosity* and its predecessors is the mechanical subsystem that provides the rover's ability to move around the surface of the planet. The six wheels, each half a meter in diameter, are constructed from aluminium with titanium spokes specially designed to allow for an amount of flexibility required for shock absorption and support. Protruding from the skin of the wheels are cleats in the shape of chevrons. This is an improvement over previous rovers where the cleats were horizontal, a flawed design in that sideways slippage was possible. The angled nature of the chevron cleats on the wheels of the *Curiosity* aimed to prevent this motion. The thin, tubeless design allowed the wheels to be as light as possible which is important not only for driving on soft parts of the Martian landscape (termed "floating"), but also for the unique landing sequence the rover had to carry out. The significant increase in the total weight of *Curiosity* meant that conventional means of landing, such as the use of air-cushion support, was not possible. The MSL leveraged the mechanical suspension subsystem on its rover for touchdown instead of providing a separate lander itself. Here, the springy wheel design helped minimise the damage brought about by the impact. As far as weight minimisation of the wheels was concerned, during the moments before the rover was released to land on the surface, the wheels were deployed in a dynamically stressful fashion from their folded position kept during flight. The deployment was sudden and extra weight would have increased the already significant forces imparted on the suspension subsystem during this manoeuvre [18].

However, the feature that is definitive of current and previous Mars mobility systems is the structural arrangement of the wheels in the mechanical suspension subsystem. Each wheel is mounted to an end of the mechanical linkage design based on the "rocker-bogie" principle. On each side of the rover, the linkage consists of two pivoting beams, one mounted to the side of the rover body, named the "rocker" and the other mounted to the middle-facing end of the rocker, called the "bogie". The front-facing end of the rocker and both ends of the bogie each host a wheel structure which consists of a pivot and strut for the front and rear wheels and a strut for the middle wheel. Both mount points allow for rotation of the beams such that, to a certain extent, the linkage as a unit remains level despite uneven terrain. This means that any of the three wheels on a side of the rover may lift due to an obstacle, up to the size of the wheel itself, without any of the other wheels lifting off the ground. This results in the obvious benefit of a maximisation of stability, minimisation of angular displacement of the rover body and maximisation of wheel contact with the surface of Mars. Figure ![] shows one of the sides of the mobility system.

![RockerBogie image]

In addition to the freedom of movement of each wheel, the rocker beams from both sides of the rover are connected via a differential bar mounted atop the rover body. The bar, which pivots about a central point on the deck of the body, limits the relative movement of the rocker beams such that one rocker will rotate absolutely in the opposite direction of the other. This significantly reduces the amount of tilt and pitch the body experiences when wheels on one corner of the rover are lifted above the other corners as well as maintains even load across all wheels. In addition, the differential provides the second

axis of stability needed to keep the body from toppling forward or backwards about the rocker pivot points.

All six wheels have drive motors that may act independently with each motor mounted to a strut. The four corner wheels' struts are connected to a pivot, actuated by a highly geared motor to allow independent rotation for steering. The configuration allows for *Curiosity* to turn conventional arcs as well as turn on the spot, an advantage for its mobility. Priority was not placed on speed for the drive motors but rather they were designed to provide high torque for robustness and for travelling on Martian terrain. The maximum speed of *Curiosity* is approximately 4 centimetres a second [19].

The mechanical mobility systems are coupled with the advanced navigational system aboard the rover, a pairing between an arrangement of navigational cameras and software. Four pairs of black and white "Engineering Hazard Avoidance Cameras" (Hazcams) with a field of view of approximately 120 degrees are positioned at the lower front and rear of the rover body, providing the rover with awareness of obstacles. The pairs of cameras create 3-dimensional maps of the terrain in front of and behind the rover. Together with the aid of this environmental mapping, two additional pairs of cameras with a much narrower field of view, namely the "Engineering Navigation Cameras" (Navcams), are mounted to the mast of the rover to provide a complementary perspective of the terrain.

Rover Compute Element

At the heart of *Curiosity* is the computational entity responsible for control of all systems on-board the rover as well as facilitation of communications with the team on Earth. This set of pairwise redundant computers is called the "Rover Compute Element" (RCE) which contains more memory than previous rovers and is hardened against the effects of radiation from the outside environment. The RCE makes use of a *RAD750* CPU designed by IBM and manufactured by BAE Systems Electronics, the radiation-hardened version of the *PowerPC 750*. The *RAD750* has a clock frequency of 110-200 MHz providing more than 266 MIPS of processing power. The pair redundancy of the RCE is such that one of the "sides" of the RCE is operating at a time while the other side kept in "cold backup". A software feature named "second chance" was built into the system whereby the alternate side of the RCE could take over basic control during the critical moments of the MSL's entry, descent and landing should the primary side fail [17]. During the flight to Mars, multiple versions of the entry, descent and landing software was sent to the spacecraft as improvements to the complicated procedure. After the landing, the original software was replaced by one which included control of the rover specifically on and around the surface of Mars. The RCE did not have enough memory to accommodate both flavours of the governing software and as such, each was installed at different points during the mission [20].

The RCE software involves the use of a real-time operating system (RTOS) approach to core scheduling and operation. JPL opted for a COTS solution for the RCE software and used an RTOS product from Wind River Systems called VxWorks. The operating system was first released in 1987 and has been used in multiple industries from space and defence to consumer electronic and automotive applications [21], not to mention having

been a part of 20 previous JPL's missions. Over the years, VxWorks has been improved in areas of modularity and upgradeability and offers a wide variety of application layers aimed at the Internet of Things. The choice by JPL, yet again, to use VxWorks on the RCE was motivated by the operating system's reliability and maturity, an extensive set of supporting tools and low-level scheduling hooks for critical real-time operations [22].

Additional Internal Systems

Stemming from the central control principle of the RCE are other computing and sensory subsystems that monitor and maintain healthy operation of the rover. One of these systems is the Inertial Measurement Unit (IMU) which gives *Curiosity* a rotational awareness about three axes: roll, pitch and yaw. It is used with the acquired 3D map of the rover's immediate surroundings to estimate the angular position of the rover during navigation and thus ensure that the rover is stable and safe.

Curiosity also has an internal control subsystem that monitors various measurements including temperature, power consumption, power storage and communication systems. The control loop will ensure that the rover remains operational and can produce warnings should any of the measurements be abnormal.

Communication

Communication with the rover from the ground station on Earth is arguably the most critical component of the mission besides the rover itself. It allows the upload of series of commands generated by the team together with the software here on Earth as well as the download of scientific data, rover telemetry and images to aid the team in keeping the rover geographically aware. The communication systems were designed to be redundant and to ensure good quality links despite challenges involving the Earth's and Mar's rotation about their own axes and obstructions as a result. Figure 2.2 shows a depiction of the telecommunication system structure.

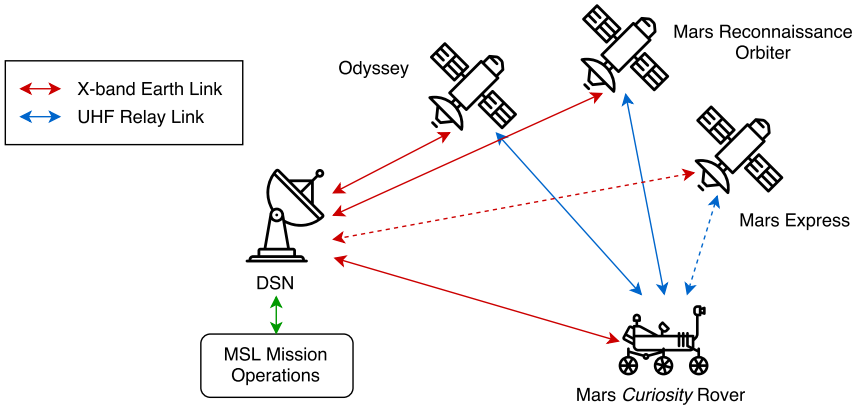


Figure 2.2: Diagram showing the structure of the MSL telecommunications system. Adapted from [23].

The Earth based component of this communication link originates from a collection of large antennas placed strategically around the Earth that alongside performing astronomical

observation, provides communications for spacecraft that are travelling at an interplanetary scale. The system, a part of JPL, is called the NASA Deep Space Network (DSN) which consists of three facilities positioned in California, Spain and Australia [24]. The positioning of the antennas in this way allows effective communication irrespective of the angular position of the Earth, meaning longer contact time between the team on Earth and *Curiosity*. For JPL, the DSN is close to home and thus mainly hosts the central data hub for communications with the rover.

On the Martian side of the link, the rover has aboard three antennas, two of which support the X-band[1] communication frequency and a third for Ultra-High Frequency (UHF) software radio communication. The X-band telecommunication system gives the rover a direct communication connection between Mars and the DSN on Earth and consists of a high-gain antenna (HGA) and the Rover low-gain antenna (RLGA) [25]. The HGA is movable with two degrees of freedom allowing *Curiosity* to point it accurately back at Earth. This antenna facilitates direct to Earth (DTE) command transmission and direct from Earth (DFE) telemetry at between 160 bps and 800 bps depending on the DSN station size. The RLGA is used mainly for contingency DFE commands and is kept as more of a redundant communication feature. Downlink communication via the RLGA is also possible but again used in case of primary communication failure.

The main method of communication with the rover when on the surface of Mars, however, is via the UHF system which uses the currently operational Mars orbiter spacecraft, the Mars Reconnaissance Orbiter (MRO) and Odyssey, as relays to the DSN. Relaying communication via multiple spacecraft which are orbiting the planet means that less power for signal amplification is required from the rover itself and the time of coverage from the perspective of the DSN stations is increased because objects orbiting the planet are obstructed by the planet body for shorter periods of time. MRO is the primary relay and Odyssey remains the redundant relay for when MRO is unavailable, provided the significantly lower data transfer speed allowed data transfer within DSN time and UHF energy constraints.

Instrumentation

Scientific observation and investigation of the surface of Mars by the *Curiosity* rover forms the crux of the MSL mission and the instruments aboard the rover are the tools with which JPL and NASA are doing such. The ten instruments, primarily scientific, hosted by the rover body, are each designed to perform specific tasks on different aspects of the rover's surroundings and samples from which it may acquire. The typical flow of investigation would be initiated by inspection of high resolution images from the rover's array of cameras. Features of interest are then located, navigated to and further inspected by the instruments mounted on the rover's robot arm and hand. Features may be inspected using those tools, or brought into the rover's body for further analysis, should that be required. Additionally, atmospheric features may be observed using the instruments design for these types of investigations.

[1] X-band - a radio frequency band within the microwave region (specifically between 8.0 and 12.0 GHz) used for engineering communication and radar

The range of instruments, as highlighted by the flow of investigation above, are split into four categories based on their method of contact with their subject. The list below shows the classification as mentioned and provides a short summary of each of the instruments.

- **Remote Sensing Instruments**
  - *Mastcam (Mast Camera)*: a suite of two fixed-focal length (FFL) cameras, one the Narrow Angle Camera (NAC) with a 5.1° FOV and 100 mm focal length, and the other the Medium Angle Camera (MAC) with a 15° FOV and a 34 mm focal length [26]. Each camera contains 8 Gb of buffer memory able to store over 5 500 raw frames, as well have the ability to pass the images through a collection of filters. Both cameras, although different in their FOV, focal length and color filter specifications, were designed to work together to provide stereoscopic views of landscapes, rocks and structures and the atmosphere.
  - *ChemCam (Chemistry and Camera)*: a suite of two remote sensing devices, the Laser-Induced Breakdown Spectrometer (LIBS) and the Remote Micro-Imager (RMI) [27]. The LIBS is the first ever laser sensing device in the field of planetary science and has the ability to investigate the elemental breakdown of rocks and other material under its sub-millimetre beam, an advantage over other breakdown spectrometers in that it can target very specific points on the surface. The RMI, which images through the same telescope as the LIBS, provides context and a highly targeted visual on the point at which the LIBS is operating. The RMI has a very small FOV of 19 milliradians and can distinguish the LIBS target point at any range within that of the LIBS laser beam. ChemCam provides further advantage over other contact-based analysis devices in that the team can use it to take samples more often without the need of the already tricky terrain traversal of the rover.
- **Contact Science Instruments**
  - *APXS (Alpha Partical X-ray Spectrometer)*: a compound instrument consisting of an electronics system situated in the body of the rover and a sensor module on the hand of the rover's robot arm. Spectral measurements are made by placing the sensor in direct contact with the material of interest, or up to 2 cm away from it, and observing X-ray emissions for a time between 15 min and 3 hours [28]. The sensor will then transmit the resultant data to the rover which contains up to 13 spectra and additional engineering information. The APXS on this rover is a significant improvement over that on *MER* with between three and six times the sensitivity for low and high atomic number elements respectively.
  - *MAHLI (Mars Hand Lens Imager)*: a focusable, high-resolution, colour camera positioned on the end of the robotic arm used to take close-up images of subjects on the surface in places to which the rest of the rover's cameras do not have access. The camera has a range of features which give it flexibility in the nature of subjects that it might capture, including night illumination, auto focus, focus stacking and video [29]. Other use cases include searching for UV material, sky imaging, sample observation, stereo-pair imaging and rover self-portraits (fault diagnosis and for education and outreach).
- **Analytical Laboratory Instruments**

- *CheMin (Chemistry and Mineralogy)*: an in-body chemical and mineral analyses instrument which operates using the principles of powder X-ray Diffraction (XRD) as well as X-ray Fluorescence (XRF) on a nominal (but not maximum) amount of 74 samples as delivered by the Sample Acquisition, Sample Processing and Handling system SA/SPaH [30]. Drill or scoop samples from this system reach the CheMin's funnel system on the deck of the rover, piezoelectrically vibrated to ensure transfer of the sample. Sample material is filtered numerous times, initially in the CHIMRA sorting chamber and then through filters in the sample cell. Sample analysis can there-onwards take up to 10 hours. The primary goal of the analytical observations performed by the CheMin is to identify and assess the historic or even current presence of water in the samples in an attempt to better understand the state of Mars with respect to the possibility of life on the surface. Raw CCD frames of the diffraction patterns and histograms are processed on the rover and then sent via downlink transmissions with the possibility of indication of indicators of previous inhabitance by life forms.
- *SAM (Sample Analysis at Mars)*: a collection of three instruments: the Quadrupole Mass Spectrometer (QMS), a Gas Chromatograph (GC) and a Tunable Laser Spectrometer (TLS) [31]. The instruments can work together and separately for much the same reason as the CheMin in terms of the search for evidence of life forms, but with focus in the area of organic chemistry in general as opposed to just water. The analyses form part of five science and measurement goals outlined for SAM are tightly coupled to the core MSL mission goals, and involve a multitude of investigations into the state and history of formation and destruction of compounds to reveal indicators of previous life. The sample manipulation system (SMS) and Chemical Separation and Processing Laboratory (CSPL) provide a means for the samples to be in the correct state and to reach the three instruments. The two support devices ensure correct environments are maintained within SAM for the analysis of the samples.

- **Environmental Instruments**
  - *RAD (Radiation Assessment Detector)*: a charged particle telescope, mounted to the deck of the rover, which analyses particles with the aim of obtaining and characterising the spectrum of particle radiation on Mars. This is done to estimate the amount of radiation a human would encounter if they were to be on the surface on Mars and further understand what this radiation may have meant for life on Mars above and below the surface [32].
  - *DAN (Dynamic Albedo of Neutrons)*: an active/passive neutron spectrometer provided by the Russian Federal Space Agency with the aim of estimating hydrogen content in the subsurface layers when the rover is traversing the planet. Most of the measurements take place during short stops that the rover may make during these journeys, the longer the measurement time resulting in more accurate measurements [33].
  - *REMS (Rover Environmental Monitoring Station)*: a pair of horizontally outward facing booms attached to the Remote Sensing Mast, below the Chemcam, as well as an additional sensor on the deck of the rover. The Instrument Control Unit (ICU) for the REMS is positioned inside the rover body. The booms, sitting approximately 1.5 m above the surface of Mars, record wind

speed and direction, pressure, relative humidity, air temperature and ground temperature while the deck-bound Ultraviolet Sensor (UVS) ultraviolet radiation. The position of the booms relative to each other and to the rover and its RMS was carefully engineered such that the wind perturbation would be as minimal as possible, an attempt to keep the wind measurements as accurate as possible. The measurements that the REMS takes are systematic and 5 minutes of observation takes place every hour of every sol[1] regardless of what operational state the rover is in, with the sensors operating at a data frequency of 1 Hz. Energy constraints allow total use of the REMS for three hours a day, which means that the REMS may autonomously increase the length of any of the 5 minute measurement operations if an atmospheric event has been detected.

- *MARDI (Mars Descent Imager)*: a FFL colour camera fixed to the body of the rover, pointing directly downwards, which is capable of taking 1600 x 1200 images used during the landing of the MSL spacecraft. The camera consists of a 90 degree circular FOV lens behind which sits a rectangular FOV sensor. The camera started taking images on command at the time of heat shield separation and continued to do so at 5 images per second until approximately 2 minutes after touchdown. Each image stored realtime into flash memory (for later transmission) can be compressed, also in realtime. The burst of images was used to provide geographical indication of the exact landing point of the rover and a framework within which engineers could base early operations. Downlink transfer of these images would have been in the form of thumbnails first and then a subset of full resolution images afterwards.

Power

Unlike conventional spacecraft and planetary space vehicles, *Curiosity* is not powered using solar means but rather energy is in the form of heat given off by the decay of a radioactive isotope, plutonium-238 dioxide. 4.8 kg of the decaying material is hosted inside of the generator named a Multi-Mission Radioisotope Thermoelectric Generator (MMRTG) produced by Rocketdyne and Teledyne Energy Systems, assembled and tested by Idaho National Laboratory [34]. The heat produced by this decay process is then converted into electrical energy through the use of thermocouple devices and excess heat is transferred to the rest of the rover body for heating, as mentioned in a previous section. The thermocouple has the advantage of being able to leverage the cold outer-space environment for the "cold junction", making RTGs well suited to interplanetary travel. Radioisotope Thermoelectric Generators (RTG) are not new to the space industry and provide missions the longevity and more consistent and reliable sources of power that one might require.

![Image of rear end of the rover, the position of the MMRTG]

The design concept behind the MMRTG is a generator that is more flexible in its field of applications as well as one which includes a high degree of safety, a desirable design feature in any space technology. Another goal of the MMRTG design is to optimise the

[1] sol - the duration of a solar day on Mars

power level over a lifetime of 14 years whilst minimising its weight [35].

2.2.4 Robot Sequencing and Visualisation Software

While many portions of the *Curiosity's* operation and movement are autonomous and require little direct input from the control team on Earth, the ground station has ultimate control over the functioning of the rover, the set of tasks it carries out and the timing with which to do so. The team also requires the ability to assess the state of the rover, its geographical position and its configuration along with, at the very least, depictions of its immediate surroundings. JPL engineers and scientist achieve such control and awareness with a specially created software suite which has been progressively developed and further improved alongside and for multiple recent Mars rover missions. The Rover Control Workstation (RCW) was developed initially to control the earlier Mars rover vehicle, *Sojourner* [36], and was later used as a basis upon which the *MER* control suite was written, the Rover Sequencing and Visualisation Program (RSVP).

The principle behind the RSVP involved the goal to maximise the use of each sol on Mars whilst relieving the requirement for operators on Earth to have to endure the asynchronous timing of sols and Earth days which was viewed as an operational constraint. Another issue was the fact that traversal of a rover on Mars resulted in the lack of knowledge of the vehicle's final position, requiring the downlink transfer of telemetry and analysis of that data upon which to plan further activities. This process introduced time lag and a suboptimal use of time for rover operation. The RSVP, along with the RCW, introduced a set of tools which make the daily command cycle a more optimal procedure, starting with the ability to rapidly interpret data received from the instruments on the rover and reconstruct the state of the rover in the most accurate way possible, presenting this depiction of state to the operator. This includes spatial positioning of the rover and the environmental context within which it is situated. Further, the RSVP provides rapid composition and simulation of commands to send to the rover, which will autonomously carry out the commands therefore freeing the operator from the time-frame of another planet.

Thus, RSVP was designed around the concept of the downlink-uplink cycle and contained two parts, Data Analysis and Sequence Generation. The Data Analysis facilitated functions of state awareness and immersion broken down into the following features:

- **State Analysis**: Analysis of the data obtained by sensors on the rover body pertaining to the state of subsystems of the rover to result in an understanding of the overall state of the rover.
- **Image Browsing**: Review and processing of images taken by the rover's cameras allowing the user to construct mosaics of panoramic sequences.
- **Terrain Modelling**: Construction of a 3D model of the immediate terrain as acquired from the rover's stereoscopic imaging systems which can later be used to plan traversals and ensure safe paths on the surface of Mars.
- **Terrain Visualistion, Immersion and Telepresence**: The use of the constructed

3D model of the terrain to immerse the operator into the environment giving the operator a better understanding of the surface and the positional state of the rover.

The RSVP's second part involves the notion of intelligent Sequence Generation and a convenient and efficient process flow in the construction of sequences of commands to send to the rover [37]. The range of commands that an operator can send to the rover is wide and these commands have varying target levels of operation and an associated variation in the level of autonomy as a result. An example of a very low-level, non-autonomous command might be to turn on a heater while a heavily autonomous, complex command might involve setting a target traversal destination and allowing the rover to construct a route based on sensor data and algorithms. Sequence generation follows a fairly comprehensive process initiated by a meeting of the group of operators whereby they will discuss the state of the rover and the aim of the particular day's events. Scenarios are analysed and weighed against the constraints of the situation. Further meetings are held to plan the detail of the activities agreed upon and to receive approval of the plans. The RSVP aids this process by allowing the team to input a draft of the set of sequences and outputting a simulation of the execution of such sequences. Distribution of the set of commands to all the teams involved allows understanding of the objectives and events by all parties and after the final sequence has been approved and the RSVP has completed validation of the sequence, the commands are built to be sent via uplink.

The variant of the RSVP used for *Curiosity* was confusingly renamed to the *Robot* Sequencing and Visualisation Program and contains much the same principle of operation and software features as the RSVP used for MER. The RSVP, in both MER and MSL cases, is clearly separated into two program user-interface components. The sequence generation is carried out in the Robot Sequence Editor (RoSE) [38] which is aware of the wide set of commands and how they will affect the rover as well as the compilation of commands for uplink. A view of the RoSE is shown in Figure 2.3. Visualisation is taken care of by the program component called HyperDrive host to a high-fidelity set of 3D and stereo Martian surface displays [39], and example of which is shown in Figure 2.4.

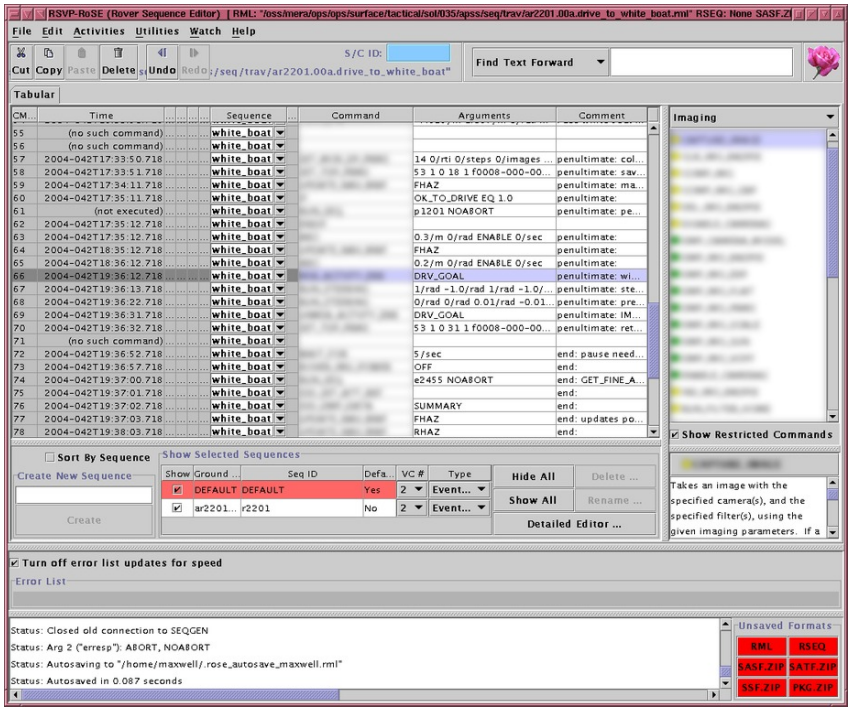


Figure 2.3: A screenshot of the RoSE as implemented in the RSVP used for MER [40]

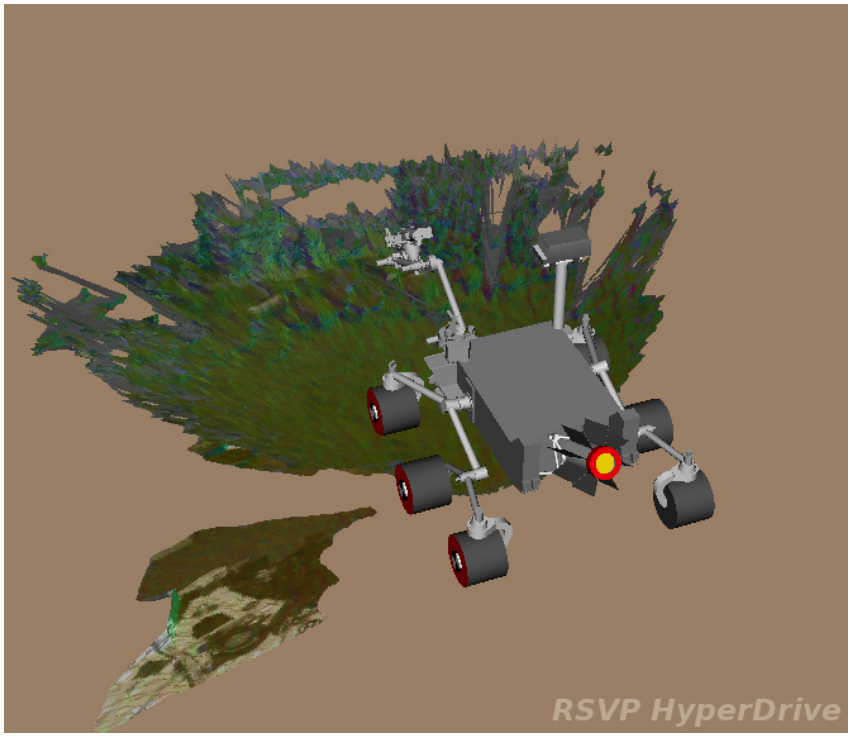


Figure 2.4: A view of the 3D model output by the RSVP HyperDrive program component for visualisation and immersion [41]

2.3 Space Education and Outreach

2.4 Web Technologies for Modern Outreach

Until recently, the world of computers and their interconnectedness has been primarily a means of personal communication and collaborative, distributed computation with minimal focus on openly sharing information and educating. It has been with the exponential increase in performance and availability of communication and network technologies that the notion of mass sharing and consumption of information in a real-time sense has made a prominent standing in the way that the world makes use of the computational devices that they might possess.

In recent years, online education has gained significant popularity due to the flexibility that it provides to the learner and the convenience to the educator. In 2001, MIT launched the OpenCourseWare initiative [42] where learners may view educational content from MIT at no additional cost other than the cost of an internet connection by any means. UC Berkeley followed suit in 2006 along with Yale, Stanford and Harvard in later years and online education, beginning simply as a prospective endeavour, turned rapidly into the educationally rich internet that exists today. Multiple other organisations joined the fast growing culture, such as Kahn Academy, offering free education to those who have access to the internet.

It is this ease-of-access that has driven the appeal of the web as a distributive platform for educational material. Institutions and organisations realised that education need not be hindered by the logistical constraints imposed on educators and that the web allowed them to educate learners in developing countries and other remote locations right from where they might be situated. They also realised that the material could be shared to a significantly larger audience compared to that in a classroom or lecture theatre. Today, advancements in web technologies and the progressive nature of current web standards means that the level of interactivity possible through a browser is only increasing.

The web has, more recently, developed itself around a core culture of open source. It is the concept where development of a product or project can be accessed, used and contributed to, at no additional cost [43]. Open source culture operates with a notion of constructive debate in collaboration in order to promote development and quality of the project and it inherently provides a highly educational environment in which people can tackle steep learning challenges with a positive and constructive outcome. Additionally, the availability of use of open source projects and culture make it possible for anyone to create and share a service or product on the internet with very little initial resources or funding required and it is this opportunity that many of the sources of online education have taken to their advantage. In a short amount of time, one can share large and complex forms of data securely and remotely for the benefit of others around the world.

2.5 Existing Curiosity Rover Models

Chapter 3

Rover Model Development Methodology

3.1 Development Objectives

3.1.1 Problem Definition

The project aimed to propagate the theme of science education and outreach, leveraging the modern technologies of today, through the development of a working, scaled down version of the *Curiosity* rover. The project brief indicated that the typical use case as desired by the client was to have the rover simulate chosen, significant features of the rover on Mars to shed some light on the level of capability of space technologies that are currently in operation. The model will be set either in a simulated Martian environment or in a small display area and be required to be remotely controllable, providing video and telemetry to viewers and viewer's devices the same way the RSVP would to the flight team at JPL.

As an initiation of the project, the requirements were explored and collated below into a list of those pertaining to the vehicle itself in a sense of the hardware as well as the software that encompassed the operation of the vehicle as an educational piece. The requirements made sure to maintain as little reference to technologies available as possible as this detail was to be further developed after analysis of the requirements on a functional level.

3.1.2 Project Requirements

1. Develop and build a model of the Mars *Curiosity* rover. The rover model should:
   (a) be a scaled down representation of JPL's rover currently operational on Mars with a level of resemblance adequate for use in a realistic exhibit. In other words, it should have been realistic enough such that someone who might have seen a picture of *Curiosity* beforehand could identify the rover,
   (b) have traversal capabilities that reflect those on *Curiosity*,

(c) be able to make use of these traversal capabilities on uneven terrain such as one which would be a simulated surface as part of the exhibit without resulting in an unrecoverable state,

(d) offer video streaming to connected clients

(e) have reasonable awareness of obstacles to prevent resulting in an unrecoverable state as well as to provide an indication of the navigational and environmental awareness systems on *Curiosity*,

(f) have data communication facilities available to best represent the communication systems and subjects of those that are a part of MSL, and

(g) be completely wireless, again reflecting the nature of operation of *Curiosity*.

2. Develop a software system to accompany the above rover in its functioning. The software system should:

(a) be able to receive data in the form of video and telemetry from the model,

(b) be able to present the data received to users or operators in an interactive manner on a platform that is available and accessible,

(c) allow input of commands or control by the users in a manner which is both friendly to a wide range of audiences and age groups and as closely representative of the manner in which JPL's flight team would do so,

(d) transmit these commands to the model to be executed, and

(e) facilitate the reception and transmission of the above data wirelessly.

3.1.3 Analysis of Constraints

As with any engineering design project, the rover development was faced with multiple constraints that affected the resulting design. Below is a brief list of the constraints known at the beginning of the project.

- Typical exhibition space limited to a dimension of 3m x 3m
-

![Fill out]

3.1.4 Functional Analysis

The client requirements as highlighted in the previous sub-section were analysed to result in a functional outline in lieu of developing a list of specifications. This analysis served as the starting point for the componentization of the project, allowing for the conceptual development to follow the breakdown. This is discussed further in the next section. Here, each of the requirements, and combinations of them, were used to result in a breakdown of functions and aspects. A significant effort was made from the start of the project to develop the rover in a modular fashion. This is not to be confused with the outcome being modular (although this was still a desirable feature) but is instead the way in which

ideas were formed and developed. The functions outlined in this analysis were treated as modules, where possible, and developed so that each module had as little dependence in operation as possible on another. Following this mindset allowed for the simplification of the design process and the robustness of what was developed against constraints and unforeseen obstacles during development.

The current requirements distinguished clearly the two aspects of the project which inevitably became the two major points of development. Both aspects, the mechanical vehicle and the software system, and their differing natural design approaches made it suitable to discuss them separately, where appropriate. The specifications indicated the possibility of a subset of *Curiosity's* primary functions be included in the model. The requirement of a video feed as well as simulated terrain traversal implied there be at least the mast subsystem, which included the moving head components, and a functioning wheel and suspension system that could be controlled. The ability to drive the rover as well as point the camera via motion of the head component was deemed a combination of functions that would contribute well towards providing an engaging experience for the users. These two systems were driving of the inclusion of the accompanying systems and components, a breakdown of which can be seen in Figure 3.1.

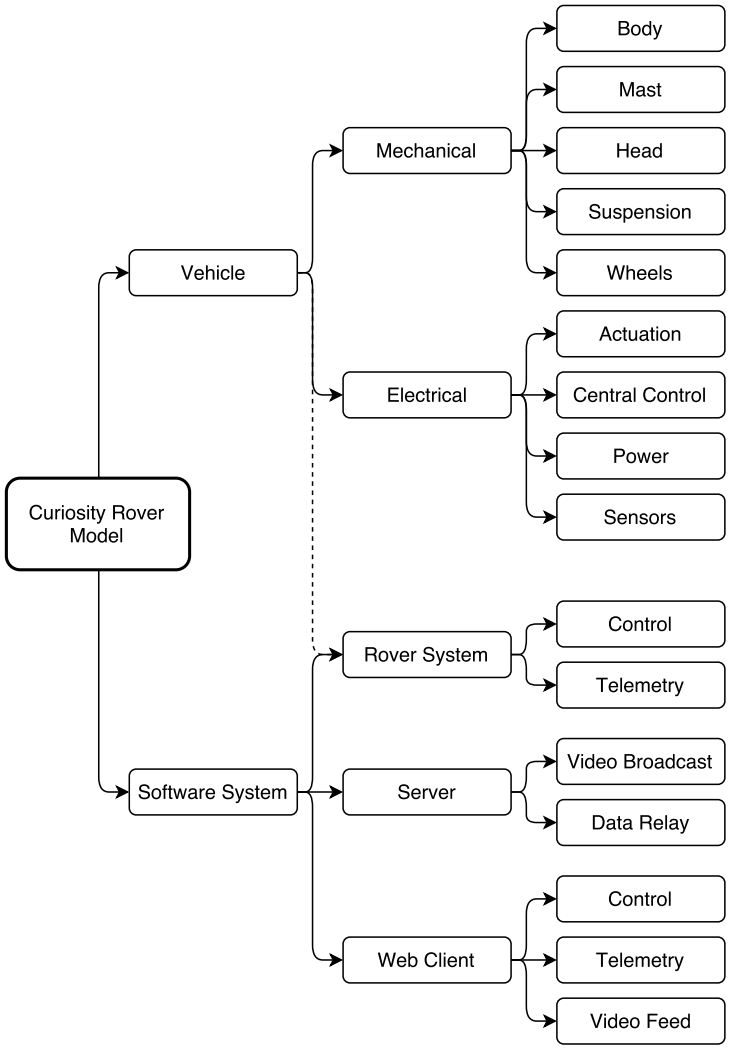


Figure 3.1: Diagram showing the simplified breakdown of functional entities of the project

![Fill out] ![Need to include something about outreach]

Rover Equivalence and Terminology

Following the educational theme of the project, much of the terminology used on *Curiosity* was borrowed for use in subsystems and components on the model, in both mechanical and software aspects.

![Fill out]

3.1.5 Technical Specifications

The technical specifications were derived from the client requirements and the functional analysis, with knowledge of the systems and subsystems on *Curiosity* (taken from review of literature as covered in Chapter 2. The specifications further compartmentalised the vehicle and software systems as evident in the structure of them in the lists that follow. These technical specifications served as the baseline requirements for the final design. Section 3.1.6 adds to the these specifications a list of secondary objectives which were considered during the design process, but were not mandatory.

Vehicle Specifications

- **Mechanical**

RM-1: General Specifications:

RM-1.1: Must be as proportional as possible to the Curiosity rover on Mars

RM-2: Body:

RM-2.1: Box shaped

RM-2.2: Allow mounting of the mast and differential on the top surface

RM-2.3: Allow mounting of the suspension system on either side

RM-2.4: Allow mounting of additional sensors

RM-2.5: Allow mounting of additional detail such as side panels and other mockup objects

RM-2.6: Allow mounting of electrical internals

RM-2.7: Provide protection/coverage of electrical internals from the external environment

RM-3: Mast:

RM-3.1: Provide mount point for the head module

RM-3.2: Facilitate full rotation about the z-axis (camera panning/yaw axis)

RM-3.3: Facilitate at least 120° degrees rotation about the y-axis (camera pitching axis)

RM-3.4: Be structurally secure providing robustness against lateral forces on mounted head module

RM-4: Head:

RM-4.1: Be mounted onto the mast module

RM-4.2: Provide mount point for a sensor
RM-4.3: Allow mounting of a camera module

RM-5: Suspension:

RM-5.1: Ensure body stability
RM-5.2: Maintain stability despite uneven terrain. This includes terrain which might require asymmetrical articulation of the system (a rock underneath one side of the rover and flat terrain underneath the other)

RM-6: Wheels and Hubs/Pivots:

RM-6.1: Match the shape and proportion of the wheels on Curiosity
RM-6.2: Provide traction required for the proposed terrain
RM-6.3: Have steering capabilities which would amount to arc pattern traversal as well as rotation of the rover around a central fixed point

- **Electrical**

RE-1: Actuation:

RE-1.1: Provide continuous rotational actuation for driving. The actuation speed should be controllable and must be of high enough torque to satisfy traversal specifications
RE-1.2: Provide sufficient magnitude rotational actuation for turning of the wheels for steering. The rotational position should be controllable and must be of high enough torque to facilitate turning of the wheels in-place
RE-1.3: Provide required magnitude rotational actuation for panning and pitching of the head/mast module. Both axes must allow positional control

RE-2: Central Control:

RE-2.1: Host onboard software system for control of hardware
RE-2.2: Facilitate wireless communications with the server
RE-2.3: Interface with actuation and sensory input hardware
RE-2.4: Have performance capabilities sufficient for processing and streaming of video data from the head module

RE-3: Power:

RE-3.1: Provide power for the central control hardware as well as actuation and sensor hardware components
RE-3.2: Fit inside or on the body module
RE-3.3: Have the ability to be turned switched off or on
RE-3.4: Allow convenient removal of source
RE-3.5: Allow easy access to charging ports
RE-3.6: Provide a means of indication of voltage for telemetry and low-battery warnings

RE-4: Sensors:

RE-4.1: Provide immediate environment data required to implement elementary obstacle detection and avoidance
RE-4.2: Be mounted in locations similar to those on *Curiosity*
RE-4.3: Be compatible with the central control module in terms of data interface

RE-5: Camera:

RE-5.1: Facilitate a monoscopic video feed
RE-5.2: Be mountable to the inside of the head module
RE-5.3: Be compatible with the central control module in terms of data interface

Software System Specifications

- **Rover Embedded Software**

RS-1: General Specifications:
RS-1.1: Allow connection of a remote client for telemetry and control as well as another for video streaming
RS-1.2: Be robust against hardware errors and intermittent communication so as to maintain operation in these circumstances

RS-2: Control:
RS-2.1: Provide a programmatic means to peripheral hardware access
RS-2.2: Translate control input commands into hardware output signals for control of peripheral hardware components
RS-2.3: Declare/define and execute programmatic sequences facilitating procedures such as system booting, communication initialisations, hardware initializations, self diagnosis ![probably need to make a list of the required sequences]

RS-3: Telemetry:
RS-3.1: Emit system telemetry to the connected client consisting of system, process and hardware state as well as sequence execution notifications

RS-4: Video Stream:
RS-4.1: Provide a stream of the video data to the connected client
RS-4.2: Provide video resolution on or above VGA (640x480) resolution

- **Server**

RS-5: General Requirements:
RS-5.1: Manage communication with the rover system
RS-5.2: Manage communication with the connected web clients
RS-5.3: Serve web application to the connected web clients
RS-5.4: Manage roles of the connected web clients with respect to their level of access and ability to control the rover

RS-6: Video Broadcast:
RS-6.1: Connect to and accept video data from the rover video stream
RS-6.2: Broadcast video stream to a scalable number of connected user clients
RS-6.3: Be robust against communication intermittency in terms of connection with the rover system

RS-7: Data Relay:
RS-7.1: Relay control input commands from a controlling web client to the rover
RS-7.2: Provide a means of simulating long-distance communication
RS-7.3: Relay telemetry data from the rover system to the connected web clients
RS-7.4: Relay state information of the rover system and server system to the connected web clients

- **Client**

RS-8: General Requirements:

RS-8.1: ![Fill out]

RS-9: Control:

RS-9.1: Provide two means of control of the rover, if access is granted:

1. RoSE-style command sequence input, allowing composition of a sequence of commands and playback of such commands
2. Interactive joystick/button interface

RS-10: Telemetry:

RS-10.1: Accept and display telemetry received from the rover via the server and from the server itself

RS-11: Video Feed:

RS-11.1: Accept and display video feed received from the broadcast

![Need to include something about the outreach specifications]

3.1.6 Secondary Objectives

3.2 Conceptual Design and Development

It was clear from the composed list of specifications that both electrical and mechanical aspects of the project required development of a large number of parts. In a typical conceptual design process, one could propose a number of complete concepts (i.e. incorporating all aspects of the project) and then make a judgement based on analysis of these concepts. For this project, it was decided that each of the subcomponents outlined in the specifications be conceptually envisioned separately with consideration of neighbouring or related subcomponents and the compatibility between each. Analysis was then undertaken for each of the subcomponents and a final design was composed in a convergent manner, taking the best concept from each of the analyses.

The fact that the design of *Curiosity* off which this model was based helped to maintain structure during a highly parallel, componentised conceptual development. However, the majority of the software components and some hardware components relied on the design of other components therefore the design process was not *entirely* parallel. In fact, in the case of the vehicle development, it was due to it being largely a process of replication that most of the conceptual development involved material and manufacture design choices as opposed to brand new conceptual ideas that required analysis of design feasibility.

3.2.1 Rover Concept Proposals

Body

All of the proposed ideas for the body component of the model revolved around the idea of a hollow box structure. The box was required to host electronics but at the same time, provide structural stability for all other components that were to be mounted to it. Therefore, the choice here was between the materials from which it would be built.

Concepts

1. **Carbon Fibre:** The first idea envisioned the use of carbon fibre to form a box structure that could be very thin and light but still offer the required strength. The carbon fibre would be cured around a mould made from another rigid, easy to use material. When rigid, holes would be drilled for mounting components and electronics. This concept includes the use of fibre glass which is commonly interchanged with carbon fibre. Both materials offer similar tensile strength, however, carbon fibre is far more robust in flexure [44].
2. **Perspex/Acrylic Sheet Assembly:** The next idea involved creating the box by designing and cutting panels from acrylic sheet of acceptable thickness, and later fusing the panels to form the structure. Cut-outs could have been included in the design together with holes for shafts and mounting points, which may also have been drilled after the fact. Internal support structures could have been included if the strength of the bonds or of the structure in general was in question due to the fact that acrylic sheet offers high flexibility. Figure ![] shows an example of how the panels might be assembled.

![Perspex concept render]

3. **3D Printing:** One of the aims of the project was to develop the model with high realism in an attempt to make the use of the simulator an engaging and appealing experience. The idea of 3D printing the box structure was considered and it would have allowed for a large degree of detail to be included at little additional effort or cost. Most features such as mounting points (those beyond just holes) and aesthetic detail could have been designed on top of base and internal structural support. A material could have been chosen which might offer the required rigidity, however, due to the nature of the manufacturing process, specifically the reliance on heat for the deforming of the plastic filament in the printing process, 3D printed components would not provide the same strength and robustness as compared to that of the other concepts. A 3D model of the rover created and published by NASA was found which was intended for 3D printing. Of specific interest was the body component which shows the detail that is achievable with this method a render of which is in Figure ![].

   ![Figure of 3D nasa body model]

4. **Milled Aluminium:** Aluminium was another concept that was considered due to its easier manipulative qualities (compared to those of steel) as well as significant reductions in weight. The box structure could have been milled from a block to form the hollow structure that is required, using CNC technology. Holes for mounting and a fair degree of aesthetic detail, which may not have lived up to that achievable by 3D printing means, may have been possible as well. Having the box structure made from aluminium would have meant that threaded holes for mounting would have been possible, eliminating the need for full-stack fasteners.

Discussion

All of the above concepts were achievable, however, each drew on very different material requirements and manufacturing techniques. Carbon fibre moulding and setting was seen as being a potentially difficult process in terms of ensuring an accurate outcome as it relied on a larger degree of manual manufacturing input. It was also the only idea that required extra components to be manufacture in support, namely the mould around which it would have been formed. The other three concepts allowed for more direct CAD-to-finished-product processes and the automation involved in the manufacture of them meant higher accuracy and less manual input. Since the model was small in scale, a design choice discussed further on in this report, strength of components and the weight of other components was far less of a priority as compared to resistance to heat and level of detail.

Comparative Analysis

Table 3.1 shows the weighted comparative analysis of the body concepts.

Attribute	Weight	Carbon Fibre	Acrylic Sheet Assembly	3D Printing	Milled Aluminium
Ease of Manufacture	5	1	5	3	4
Cost of Manufacture	4	4	5	3	4
Duration of Manufacture	5	4	5	3	4
Cost of Material	4	2	4	3	3
Weight	5	5	5	4	2
Tensile Strength	2	4	3	3	5
Modulus	3	5	4	4	1
Achievable Detail	3	1	1	5	4
Achievable Accuracy	3	1	3	4	5
	Total	2.735	**4.147**	3.353	3.324

Table 3.1: Comparative analysis of the body component concepts

Suspension System

The suspension system of the rover was a critical part with respect to the traversal requirements. It was decided up front that the system replicate the feature as it was on *Curiosity* in both appearance and in operation. The Rocker-bogie mechanical principle employed for the *Curiosity's* suspension system was simple and robust and therefore made clear the decision to use the principle in the model as well. The design problem here was more concerned with the structure and material of the joints as well as how they would be fitted with the beams/rods that links the system together. Another design consideration was that of the differential cross-beam that was mounted to the top of the *Curiosity* which articulated around a center point. The choice of differential bar was dealt with in a separate analysis.

All of the concepts imply the use of shafts and bearings for free articulation between each of the rocker-bogie sections.

Concepts

- **Fully 3D Printed Assembly:** In this concept, all the parts were to be 3D printed in full. This meant that the joints and beams of the suspension system were not separate pieces, lowering the number of pieces required to be manufactured. Since the suspension system was the largest load bearer compared to that of the other subsystems, the fully printed pieces could have been reinforced with an aluminium or steel rod set down the center of the beam sections. The reinforcements could have extended partially into the joint section of each piece, as the most amount of structural risk would have been at the point where the joint and the beam meet.
- **Printed Joints with Aluminium Tubing:** Instead of printing the entire system, an option of printing the joints only and fitting them with aluminium tubing, as the beams, was considered. 3D printing provides the benefit of being able

to materialise complex objects which may contain features which conventional manufacture methods might not be able to accomplish, thus making it well suited to the unique nature of the joints in the suspension system. The beams, however, were standard in shape and would not have put this benefit to use, hence motivating the suitability of a light but strong product such as aluminium tubing. The joints would have been designed either to have the tubing fit into the joint, or have a plug onto which the tubing could be pressed.

![Render of an example plug joint and alu]

- **Sheet Brackets with Aluminium Tubing:** This concept built on the previous concept with the joints being made from sheet metal bent into bracket-type shapes onto which the tubing could have been fastened (by means of clamps). The bending process would have allowed for formation of the non-conventional angles that the suspension system required.

  ![Render of an example sheet bracket]

- **Milled Aluminium Joints with Aluminium Tubing:** Again, instead of the 3D printed joints or the sheet metal, this concept made use of milling aluminium to form the joint structures and having aluminium tubing be fitted into these joints. The milled joints would have been able to offer more surface area to features such as mounting points for the tubes and bearings, meaning that these parts would be more secure. This idea was borrowed from the OpenCuriosity project![] as highlighted in Section 2. ![Image of the machined joint of OpenCuriosity]

Discussion

The fully 3D printed concept was appealing in that it offered the most direct path from CAD design to the finished product but had much reduced structural qualities as opposed to that of the other concepts. Using a combination of tubing and manufactured joints made sense in terms of the nature of the features and aluminium tubing provided strength beyond what was required, at least as far as the tubing itself was concerned. Brackets made from sheet metal may have offered the best weight (that is, the lightest weight contribution) but required extra manual manufacture as well as would not have been suited to mounting bearings and the tubes whilst maintaining mount rigidity. Both milled aluminium and 3D printed joints solved this problem with the ability of being able to provide more rigidity for fastening tubes and fitting bearings. However, milled joints were bound in structure to the block of aluminium from which they were to be milled, meaning that in one particular plane, the axes of the joints would not have been able to be angled such as required by the suspension design.

Comparative Analysis

Attribute	Weight	Full 3D Print	3D Printed Joints w/ Tubing	Sheet Joints w/ Tubing	Milled Joints w/ Tubing
Ease of Manufacture	5	4	3	2	3
Duration of Manufacture	3	2	4	5	3
Cost of Manufacture	4	2	3	4	3
Cost of Material	4	3	4	5	4
Weight	4	4	4	3	2
Link Mount Rigidity	5	3	4	1	5
Aesthetic Accuracy	3	5	5	1	3
Suitability for Wheel Mounts	4	5	5	4	5
	Total	3.500	**3.938**	3.031	3.563

Table 3.2: Comparative analysis of the suspension system concepts

Differential System

The differential system comprised of a beam or arm that articulated about a center point on the top surface of the body and linkage mechanisms connecting the ends of the arm to the rocker pivot point on either side's suspension system. Due to the motion of the differential, strength was only required in the horizontal plane which gives reason for the thin, flat design of that on *Curiosity*. The linkages on the ends of the bar required hinges with two degrees of freedom, the detail of which is discussed further on in this report. Once again, the principle of operation of this subsystem was taken from *Curiosity* itself and therefore was not the design choice to be made here.

Concepts

- **Acrylic Sheet Bar with Steel Cord Linkage:** Since the differential bar was required to take forces in the horizontal plane, the bar did not have to be round and a conceptual idea involved cutting out the flat bar from acrylic sheet. The acrylic sheet would have been thick enough such that it be press fitted onto a bearing and shaft in the center of the body deck. The ends of the differential would then have been connected to the extensions on the main suspension joint by means of steel cord. The cord would have allowed for the degrees of freedom required given the interface between the differential bar and the suspension and each of their component axes of motion.
- **3D Printed Bar with 3D Printed Hinge Pieces:** Instead of cutting the bar from acrylic sheet, this concept envisioned the bar being 3D printed. The linkages would have also been 3D printed as two parts per hinge bolted together and each end of the hinge (one at the suspension and one on the differential bar) would be joined together using threaded bar, secured by use of fasteners.

Discussion

Since the acrylic sheet is already flat, it suits the problem well and is easier in terms of manufacture compared to a 3D printed version. Holes for the bearings and the hinges on the ends of the bar could be included in the cutting process. Two sheets could have been glued together to form a thicker beam in the case that the bearing was thicker than the sheet. The 3D printed version, however, would have been of designed thickness thus allowing custom fitting for the bearing. Although steel cord was considered given it's flexibility and thus ability to cater for the ranges and axes of motion of either ends of the linkage, it was shortly dismissed given that it would have only been able to provide support in tension and not in compression. A fixed threaded bar, as proposed in the second concept, provides support in both tension and compression situations, therefore meeting the requirements. Threaded bar was chosen for easy fastening as well as providing the ability to adjust the extension of the linkage for fine tuning the balance of the suspension-differential system and ultimately the balance of the rover. In any case, a weighted comparison was still made in Table 3.3 since the 3D printed hinges were still compatible with the idea an acrylic sheet differential.

Comparative Analysis

Attribute	Weight	Acrylic Sheet Bar w/ Steel Cord Linkage	3D Printed Bar w/ Printed Hinge Pieces
Ease of Manufacture	5	5	3
Duration of Manufacture	3	5	2
Cost of Manufacture	4	4	3
Cost of Material	4	5	3
Weight	3	5	4
Strength	5	3	5
Mountability	5	3	5
Linkage Motion	4	0	4
Linkage Support	5	5	5
Aesthetic Accuracy	2	1	4
	Total	3.575	**3.900**

Table 3.3: Comparative analysis of the differential concepts

Wheel Hubs and Pivots

The center wheels were fixed in rotation about the z-axis and thus the mounting features of these two wheels were included in the suspension system as in the previous concept section. The front and rear wheel pairs, however, were required to rotate in order to provide steering to the rover and therefore had to accommodate for this rotation as well as actuation components for this motion. The wheel pivots were also required to be attached to the suspension system.

The concepts developed for these components followed very similar concepts to those of the suspension hinges as they offered the same principles of articulation and mounting. The final decision would therefore be in accordance to the suspension system final design.

Wheels

The wheels on *Curiosity* are signature features in aesthetics as well as, of course, in function. The wheel shape includes the characteristic curved cross-section which has benefits for terrain like that on Mars and are unique in the thinness of their outer surface or skin. It was noted that the relative strength of the wheels on the scaled model that was being developed would be required to be significantly less than on *Curiosity* and so the design choice here was based primarily on the resulting aesthetic accuracy as the wheels would serve as a great feature in which to fulfil the client requirement of the model being highly realistic.

Concepts

- **3D Printed Wheels:** The fact that the wheels had the distinct shape that they did meant that 3D printing them would render highly accurate representations in terms of their shape on those of *Curiosity*. The design would include holes and points at which shafts and/or bearings could be fitted without the need for drilling or any other type of post-produce manipulations.
- **Off-the-shelf RC Wheels:** Another option was to purchase read-made wheels and tires intended for use on radio-controlled cars. The wheels would have points for mounting by default and would offer the benefits of a rubber tire over that of plastic. Once again, this idea was borrowed from the OpenCuriosity model in ![]. The project indicated that this method was successful in function.

Discussion While the 3D printed wheels would have offered less structural robustness as ready-made wheels designed to endure high impact, gravel environments, there was no requirement for the model to traverse any faster or in any better manner than *Curiosity* and thus the added benefit of the rubber tires and strong wheels would be an over-design. Although the RC wheels would have had points at which shafts could be fitted, and potentially even bearings already installed, this would impose limitations on the size of the shafts chosen. In the case of the 3D printed wheels, holes sizes could have been chosen to work with materials and bearings that were available.

Comparative Analysis

Attribute	Weight	3D Printed Wheels	Off-the-shelf RC Wheels
Ease of Manufacture	3	3	5
Duration of Manufacture	4	2	5
Cost of Manufacture	4	3	2
Cost of Material	4	3	2
Weight	1	5	2
Traction	3	3	5
Terrain Suitability	3	4	5
Mount-type Flexibility	4	5	2
Aesthetic Accuracy	5	5	1
	Total	**3.613**	3.097

Table 3.4: Comparative analysis of the wheel and tire concepts

Neck and Head

The mast of the model required rotation about the z-axis and a joint about which an additional axis of rotation was possible to result in two degrees of freedom for the head component. *Curiosity* employed a hinge mounted to the bottom of a second, smaller box structure, the head, which rotated to provide camera pitch. The pitch actuation mechanism was situated atop a second mechanism which provided actuation for pan-axis rotation. The size of *Curiosity* allowed smart fitting of the motors in-line with the mast shaft and embedded in the head mount hinge, however, the scaled model was not able to accommodate for motors in this way. ![Put this in the detailed design, concept designs are meant to be somewhat ideal!] It was decided that it be acceptable for the mast assembly to be out of proportion, visually, in order to be able to fit the required actuation (the component choices of which were largely based on availability).

Concepts

- **Aluminium Tube Mast with 3D Printed Fittings:** This concept made use of aluminium tubing (the same material as in the suspension system concepts) which would be set and fastened into the body. A 3D printed plug with mounting points for the camera-pitch actuation mechanism would have been designed to fit into the top of the tube, as can be seen in Figure 3.2. Camera-pan actuation would have been built into the inside of the body into which the mast tube would extend. The height of the head could therefore have been adjusted since the tube was free-moving with respect to the rover deck.

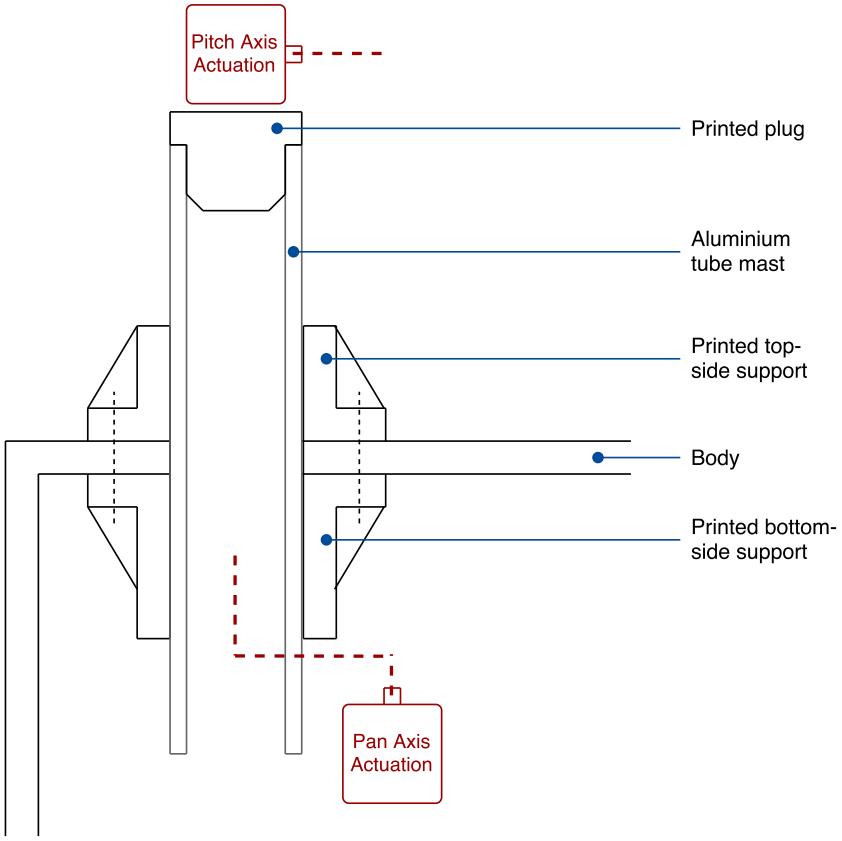


Figure 3.2: Conceptual diagram of a section view of the body and aluminium mast assembly

- **Full 3D Printed Assembly:** As opposed to making use of the aluminium tube, this concept employs 3D printing for the full assembly which would be mounted to the top of the rover body with no portion of it extending below the deck. The camera-pan actuation would be the same as in the previous concept but the camera-pitch actuation mechanism would be brought above the level of the rover deck. Further, an actuation mechanism that combines both axes of motion could ave been developed to reduce the spatial footprint that it might have incurred. The width of the base of the mast, at the mounting point, would be increased to provide structural support.

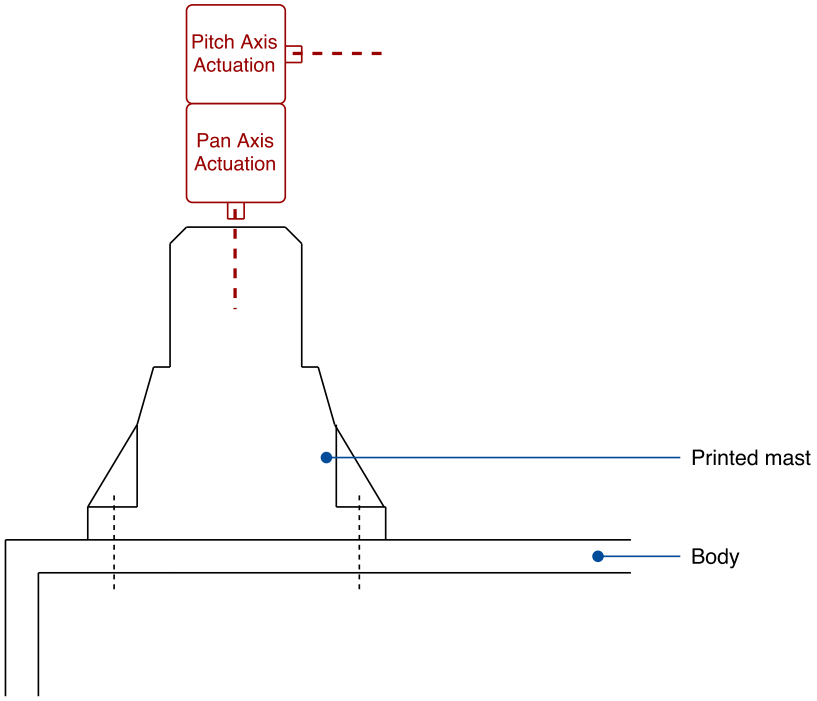


Figure 3.3: Conceptual diagram of a section view of the body and printed mast assembly

Discussion

Positioning one of the two component actuation mechanisms inside the body had benefits for the spatial footprint above the rover deck, however, the implications of taking up more space inside the body were far greater given the intended use of that area. Having the entire tube mast rotate within the opening hole from below the deck might have increased the torque requirements on the actuation mechanism, dependant on the nature of the opening. The intended use of the opening was to provide structural support and any fit tolerance introduced to improve the torque requirements of the actuation would negatively impact the effectiveness of the hole feature in its support function. This could have been solved with a bearings, however, that would have incurred addition room for mounting.

Comparative Analysis

Attribute	Weight	Aluminium Tube w/ 3D Printed Fittings	Full 3D Printed Assembly
Ease of Manufacture	3	2	4
Duration of Manufacture	4	4	3
Cost of Manufacture	4	3	3
Cost of Material	4	4	2
Weight	4	3	4
Proportion	3	4	3
Body External Footprint	3	4	3
Body Internal Footprint	5	1	5
Strength	5	5	4
	Total	3.200	**3.514**

Table 3.5: Comparative analysis of the wheel and tire concepts

Actuation

All components requiring actuation mechanisms have been covered in the above concepts. Consisting of only rotational motion requirements, the mechanisms were split into two categories based on the desired order of output motion (translated from the desired order of input control signal), namely angular position and angular velocity. Position actuation was required for rotating the wheels about their z-axis pivot for turning or steering and to provide panning and pitching motion to the head subsystem with the camera. Both of the functions involved setting a desired angular position and having the mechanism hold that position during operation. On the other hand, driving the wheels of the rover was better thought as involving an output velocity. *Curiosity* made use of high-ratio motors for all types of rotational actuation to ensure robustness of the design, higher torque outputs and to achieve precise control of each of the driven features which was acceptable in that high-speed performance was not a targeted requirement.

Concepts

- **High Torque DC Motors:** Using high torque DC motors would have been the most accurate replication of the actuation as used on *Curiosity*, as each of the mechanisms had high-ratio gearboxes attached to the brushless motors. The DC motors would have been controlled by means of PWM which would have resulted in an output angular velocity. For this reason, a control loop mechanism would have to be employed for positional motor control in the case of the subsystems that required it.
- **Analog RC Servos:** A candidate alternative to using high torque DC motors that was considered was servos, intended for radio-control vehicle use, into which a high-torque gearbox was already built. An example of this type of motor is shown in Figure 3.4. The motors operated using a pulse signal of which the width translated to a specific position as a percentage of the motor's rated angular range.

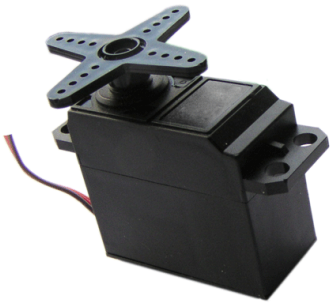


Figure 3.4: Image of an example of an RC servo motor [45]

Discussion

The two candidate actuation devices were in fact similar in that they both offered high torque output, however, the inclusion of a built-in analog position control system differentiated the servos from the DC motors. If the DC motors were to be used, the central system would had to provide an external control system to implement position control for the head and mast subsystem as well as the pivoting of the wheels. Further, this would have required output state capture by means of an encoder or an analog-to-digital converter adding complexity to the system. The fact that the servos had this control functionality built in meant this solution would have greatly reduced the incurred complexity of the actuation of the rover as a whole, as all that would have been required is for the system to provide a power rail and PWM signals.

The choice of actuation mechanisms was highly dependant on the chosen combination of subsystem concepts, specifically that of the power supply, the central control system and those that needed actuation themselves. No weighted comparison was performed for the above concepts as the choice was heavily affected by these subsystems.

Central Control System

The central control system was a critical component not only to the rover, but to the conceptual design process as it was an enabler/disabler of many of the candidate solutions. Discussed here are the electronic hardware comparisons made with respect to the central control system. The software design process took on a secondary priority approach and as such, the hardware choices were driving of the design (not without consideration of implications in the software system). As will be mentioned in full in the detailed design section, it was intended to follow a COTS design approach as far as possible given the time-frame of the project as well as the notion of keeping the design open to others who might be familiar with the hardware components chosen with respect to the aim of open sourcing. As far as education and outreach is concerned, familiar hardware is well suited to helping users and those involved in the project learn the principles of a rover design.

Conceptual candidate systems included popular, small, single-board computers, sized appropriately with the intention of fitting the system in the body of the model. Note that the lower-level device class suited to deeper embedded software applications was considered and would have proved suitable if it was not for the video streaming requirements.

It was anticipated that the video feed would require on-rover encoding and compression and thus imposing the need for a better performing device capable of running a high-level operating system. The requirements for this system, which included wireless communication and embedded interfaces, were kept in mind when performing the comparative analysis. Notable specifications in accordance with the requirements are shown for each of the boards.

Concepts

- **Raspberry Pi Model B:** The Raspberry Pi was a credit-card sized single-board computer which was developed with the intention of aiding computer-science education. It made use of a well-performing CPU as well as an on-chip GPU making it suitable for low-end, media-based computational applications. Raspberry Pi computers have a very large online community from which vast resources were available.

  Notable specifications (for the 3rd generation model):

  - **CPU:** 1.2 GHz 64-bit ARM Cortex A53 (Broadcom BCM 2837 SoC)
  - **Memory:** 1 GB
  - **Storage:** None, microSD Card Slot
  - **GPIOs:** 40 pins,
  - **Network Connectivity:** Bluetooth 4.1 and Bluetooth Low Energy, 100 Mb Ethernet, 2.4 GHz wireless
  - **Other External Interfaces:** 4x USB 2.0, Camera Serial Interface (CSI)

- **Orange Pi:** The Orange Pi, an open source variant to the Raspberry Pi, was considered as it offered much the same capabilities as the Rasberry Pi. It was able to run many open source operating systems such as Debian and Ubuntu.

  Notable specifications (for the Plus model):

  - **CPU:** 1 GHz 64-bit ARM Cortex A7 (AllWinner H3 SoC)
  - **Memory:** 1 GB
  - **Storage:** None, microSD Card Slot, SATA 2.0 Connector
  - **GPIOs:** 40 pin header,
  - **Network Connectivity:** 1 Gb Ethernet, 2.4 GHz wireless
  - **Other External Interfaces:** 4x USB 2.0, Camera Serial Interface (CSI)

- **Beaglebone Green Wireless:** The Beaglebone Green is another small board as part of the Beaglebone device family, a range of single board computers that have been developed to bridge the gap between embedded electronics and computers. The green version is better suited for embedded applications compared to that of the black version and was the only Beaglebone device that had wireless connection capabilities.

  Notable specifications [46]:

  - **CPU:** 1 GHz 32-bit ARM Cortex A8 (TI Sitara AM3358)
  - **Memory:** 512 MB
  - **Storage:** 4 GB eMMC

  - **GPIOs:** 65 pins,
  - **Network Connectivity:** Bluetooth 4.1, Bluetooth Low Energy, 2.4 GHz wireless
  - **Other External Interfaces:** 4x USB 2.0

- **Intel Edison:** The Intel Edison is less of a single board computer and more of a complete system on chip mounted to a small board intended for use in the Internet-of-Things industry as well as for mobile and wearable products. The tiny module can further be mounted to a breakout board which provides USB interfaces and a GPIO through-hole grid.

  Notable specifications ![cite]:

  - **CPU:** 400 Mhz Intel Quark x86 (Intel Atom)
  - **Memory:** 1 GB
  - **Storage:** 4 GB eMMC
  - **GPIOs:** 28 pins,
  - **Network Connectivity:** Bluetooth 4, 2.4 GHz wireless
  - **Other External Interfaces:** 1x USB 2.0, 1x USB Serial (UART), as provided by the breakout board

- **Intel Edison w/ Arduino Breakout Expansion:** The Intel Edison had available a second breakout board which was developed to make the SoC compatible with the large variety of Arduino-compatible modules and add-ons.

- **Intel Galileo Gen 2:** The Intel Galileo is a development board that better aligns with the single-board computer principle, compared to the Edison. The processor and board are fixed and allows for connection of Arduino-compatible hardware as well as supports a range of other interfaces.

  Notable specifications ![cite]:

  - **CPU:** 400 Mhz Intel Quark x86 (Intel Pentium)
  - **Memory:** 256 MB
  - **Storage:** None, SD Card Slot
  - **Network Connectivity:** 1 Gb Ethernet Port
  - **Other External Interfaces:** 3x USB 2.0, 1x USB Serial (UART)

Discussion:

After careful research into each of the above candidate products, it was decided that all of the boards were suitable for the central computing system of the rover. All of the devices were capable of running high-level operating systems as well as had some means of connecting to a video capture device as well as providing hardware interfaces that might have been required. However, caution was taken to choose a device that would not be over-powered for the application nor provide breakouts and interfaces that would have been left unused. At this point in the design process, it was difficult to determine the exact computational requirements and so the design choice was made based on anticipative measures. It must also be noted that the choice was largely influenced by availability and cost of the devices.

Attribute	Weight	R-Pi 3	Orange Pi	Beaglebone Green Wireless	Intel Edison	Intel Edison w/ Arduino Breakout	Intel Galileo Gen 2
Cost	5	4	4	3	2	2	1
Weight	3	4	3	4	5	5	4
Availability	5	3	1	2	5	4	5
Size	4	4	3	4	5	4	3
Wireless Support	5	5	5	5	5	5	0
Provision for Video Capture	5	5	5	4	1	3	3
Suitability of Processing Power	3	3	3	3	5	5	3
Add-on Compatibility	4	3	3	2	1	5	5
Power Consumption	3	1	1	3	5	4	2
	Total	3.703	3.243	3.351	3.622	**4.000**	2.811

Table 3.6: Comparative analysis of the central control system concepts

Camera

An important item in the list of requirements and specifications was the capture of a video stream to broadcast to the connected clients. The camera was required to be above VGA resolution (640×480 pixels) and have a means of connecting to the chosen central computing system. The available camera modules were categorised by connector type, listed below.

Concepts

- **CSI Compatible Webcam:** Many of the central computing system candidates provided support for a Camera Serial Interface (CSI) connected camera for video capture. CSI is a camera interface standard maintained by the Mobile Industry

Processor Interface Alliance (MIPI Alliance) that is at its 3rd stage of revision at the time of writing. An example of a CSI camera was the R-Pi Cam, produced specifically for use with a Raspberry Pi.

- **USB Compatible Webcam:** The majority of external webcams were USB connected, allowing them to be easily connected to a laptop or computer. The USB webcams that were investigated as being candidate devices varied in their class of drivers which was a potential issue for compatibility with the central control system, specifically the operating system that would be used. The chosen device, if of type USB, would have had to have been a USB Video Class (UVC) compliant device due to the fact that UVC devices are driverless and thus are compatible with a far greater range of host computers and operating systems.
- **I^{2}C Webcam:** Since the central control system would be capable of allowing serial connections, cameras that were I^{2}C connected were considered.

Discussion

The performance benefit of candidate cameras was not possible to determine based on their interface type, but rather the manufacturer and the designed specifications. It was decided that the camera be chosen based on compatibility with the central control system and availability of such a device.

Proximity

![Fill out]

3.2.2 Final Design Choice

At this point, the ideas and concepts in Section 3.2.1 had been explored in detail and final choices for each of the subcomponents had been made. This section indicates the outcomes of these choices as well as gives a brief overview of the final concept to be developed. The subcomponent concepts in Section 3.2.1, shown visually in Figure 3.5, that included weighted comparisons were finalised primarily by the outcome of those comparisons (i.e. the concept that achieved the highest score, denoted in the Tables 3.1 through 3.6 by bold numbers) and the few that did not follow the same structure of analysis were chosen by compatibility with relevant subcomponents.

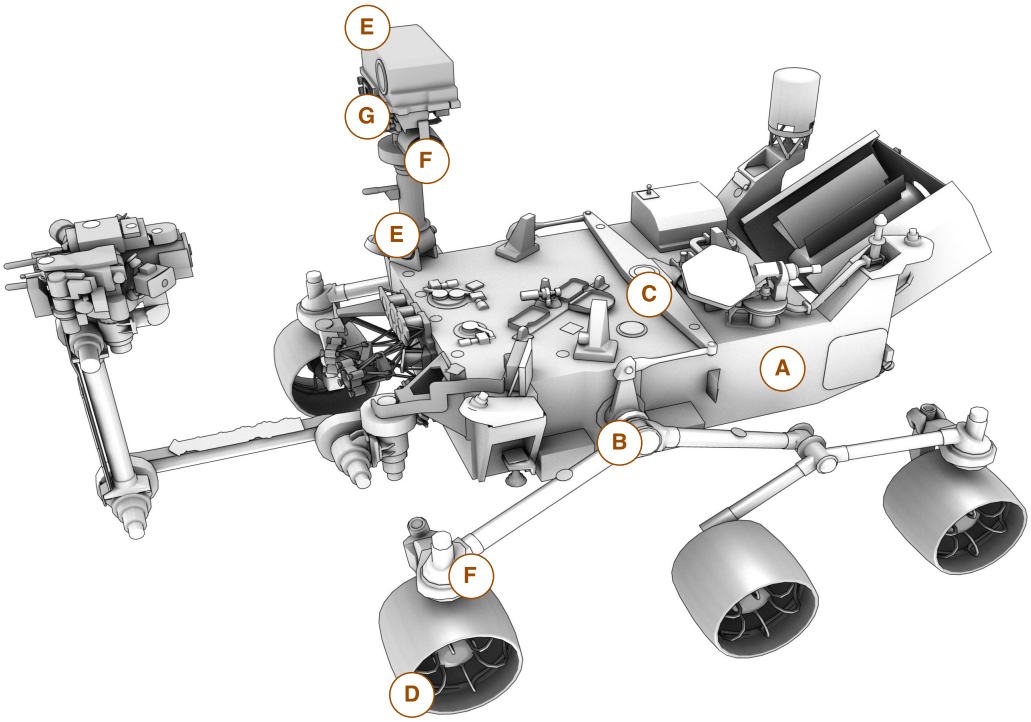


Figure 3.5: Adapted render of a model of the rover indicating the subsystems considered in the conceptual development [47]

A. It was decided that the body be made from acrylic sheet panels glued together to form the box structure required. The panel design, cutting and assembly was considered easier compared to the manufacturing of carbon fibre and 3D printing as well as being strong enough to provide the central structural support.

B. The suspension system was to be constructed from 3D printed joints and aluminium tubing for the superior total weight of the assembly and the concept's compatibility with fittings to be made for mounting the wheels. The fact that the joints were to be 3D printed meant that it would become easy to model and print the arbitrary angles of that of the suspension design without the manufacture limitations imposed by other methods and materials. The joints allowed for the design to maintain an accurate representation of that on *Curiosity* without sacrificing function and meant that close fit parts such as bearings could be easily incorporated into the design without imposing constraints on the choice of those types of parts.

C. A full 3D printed bar with printed hinges was to be used for the differential system due to the acrylic sheet and steel cord not being suitable for the required function. As mentioned in the concept's discussion, the 3D printed differential bar meant that a bearing could be correctly mounted for the motion required. The hinge on the differential bar side would be connected to the suspension side hinge by a threaded bar. As will be discussed, the connecting threaded bar could be used to adjust the distance between the hinges thus providing the ability to finely balance the rover during assembly.

D. The wheels were to be 3D printed which would allow for superior aesthetic accuracy

and custom fit design for bearings and actuation. The 3D printed wheels would be lighter than bought wheels and was deemed worth the incurred manufacture time and cost.

E. The head and neck (mast) was to be fully 3D printed to make it more suitable to being mounted to the top of the rover deck without any portion of it extending into the body of the rover, taking up space required for the electronics. 3D printing meant the ability to accommodate for the chosen means of actuation. The head would be designed to consist of two parts so that access to the internals of the head was possible. This sub-assembly was then to be mounted to parts connected to the actuation and further parts for mounting on the rover deck.

F. Choice of actuation settled upon the use of RC servos. The servos were to be of a suitable size, preferably "sub-micro"[1], and compatible in interface with the central control system. The servos chosen were highly geared and were to provide the required torque for actuation of the head and wheels. The servos would have standard mounting holes making incorporation into the design an easier process.

G. The camera was to be a USB (UVC-compatible) webcam chosen based on availability. The camera would be suitable for use with the chosen central control system and significantly lower in cost compared to the other concept options. It was decided that a webcam be physically altered to be suitable for incorporation into the design of the head.

Not shown in Figure 3.5 is the central control system, which was to be mounted on the inside of the body structure. The Intel Edison with the Arduino breakout expansion was chosen due to availability of the product as well as its compatibility with add-on hardware, the advantage of which was in the increased development process as a result. The Intel Edison was regarded as being better suited to the performance and computational requirements over the other boards and provided the necessary hardware interfaces for the primary functions and associated hardware for the system throughout.

As discussed in the functional analysis, *Curiosity's* robotic arm subsystem was not included in the model due to time and cost constraints on the project. The design processed aimed to make the addition of this type of a feature possible in potential future work on the project.

[1] A "sub-micro" servo motor is one of a range of standard sized servos for robotics and radio-controlled vehicle applications

3.3 Vehicle Design and Development

Finalisation of the chosen concepts for the development of the rover meant that the project could progress to the detailed design stages. It was discussed in Section 3.2.1 that the mechanical design of the vehicle would be driving of the project, specifically the software aspects, and thus the report will deal with the mechanical and electronic detailed design first.

3.3.1 Mechanical Design

The mechanical design was initiated by planning the basic layout of mechanical subsystems and components and from that point developing each system further. In this section, the choice of scale (and hence dimensions) is followed by the overall plan of mechanical layout. Subsystems that were peripheral to the body structure are then covered, which include the suspension, differential and mast subsystems. Finally, the design of the body structure is described as well as detail surrounding the model as a whole.

Scale and Dimensions

Replication of *Curiosity* on an aesthetic level was a project aim to increase the viewers' and users' sense of familiarity with the model, a way of promoting the engaging experience. The replication process identified that proportion was an effective and achievable starting point and it was decided that as many of the parts as possible be based off the dimensions of the corresponding part on *Curiosity*, brought down to scale by a predetermined factor. Scale was constrained primarily by the cost and manufacturing time of parts that were required to be 3D printed; the larger the model, the greater the amount of material required and the longer it would take to print it. The print bed size was also a constraint in this regard. More details pertaining to the use of 3D printing facilities can be found in Section **??**.

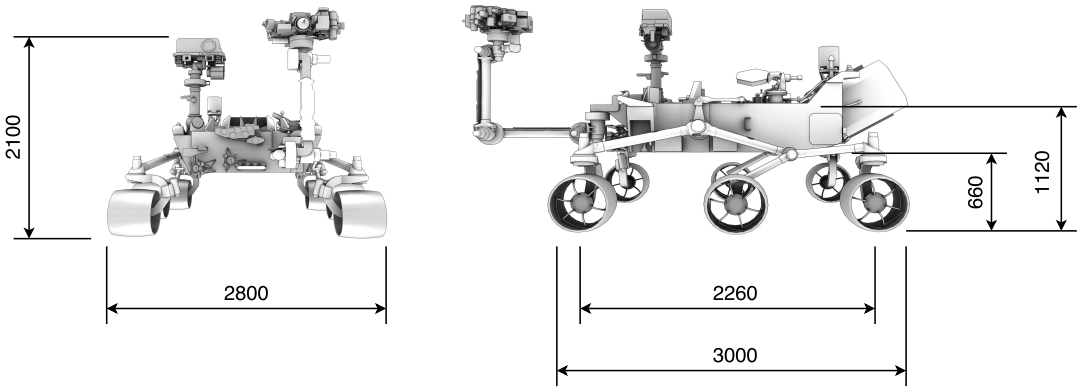


Figure 3.6: Diagram indicating the external dimensions of *Curiosity* in millimetres [47]

The dimensions shown in Figure 3.6 were retrieved from [17] and [48] as guideline, external dimensions and no other dimensions were available that might have provided a more detailed insight into each of the subsystems and components. A 3D model of a small

scale, static, printable model of *Curiosity* published by NASA was found and used for extraction of proportion of the individual components [49]. While this model was not ideal in terms of representative accuracy, it provided the much needed basis on which to generate the designs of parts with an acceptable level of similarity. From these details, it was decided that the rover be built at a 1:10 scale, making the total length and width of the model, including the wheels, 300 mm and 280 mm respectively. This scale took into account allowing the model enough space to be used within the typical exhibit size outlined in the problem definition as well as 3D printing capabilities. The scale ensured that anticipated electronic internals would fit into the body structure and that mounting of bearings and the chosen standard size of servo motors was reasonable.

The chosen scale gave rise to a set of external dimensions for each of the subsystems, highlighted in Table 3.7. The guideline full-scale dimensions in Figure 3.6 were used against the 3D model in [49], here-onwards referred to as the reference model, to derive the scale of this reference. This scale was then used to obtain external, high-level dimensions for all of the subsystems and components ensuring that they remained in proportion. The dimensions were used for positioning of components and aided in the management of space allocation thereof. There were certain cases whereby external factors influenced component dimensions to result in these components not abiding by spacial allocations as well as proportion, the cases of which are dealt with within their respective sections.

The reference model, which was in Standard Tessellation Language (STL) format, was imported into a CAD package with millimetre units and evaluated in that state to result in a scale of 1:1.7037 (reference model as to *Curiosity*). The dimensions shown in Table 3.7 are of priority components and subsystems only, omitting details related to components that were intended to serve aesthetic purposes only (such as a mock-up of the RTG).

Layout Plan

The dimensions in Table 3.7 were used to construct a plan of the layout of subsystems to be used for further detailed design. The plan can be seen in Figure 3.7 along with selected dimensions from the table.

Standard Features

Prior to designing and developing the components and parts in detail, standard feature dimensions and sizes were set up to maintain a level of consistency throughout the process. This included features such as holes, walls and fasteners and served as a general guideline allowing for variation of these parameters if required. Table 3.8 highlights these feature details.

Suspension

The collection of joints, pivots, struts and wheels was collectively referred to as the suspension system, one positioned on either side of the body structure. Each side of the suspension system included two fixed link mechanisms, the “rocker” and the “bogie” as

Feature	Dimension	Code	Value (mm)
Rover	Height	A1	210
	Width	A2	280
	Depth	A3	300
Body	Height	B1	46
	Width	B2	136
	Depth	B3	190
	Ground Clearance	B4	66
Suspension	Height	C1	122
	Width	C2	72
	Depth	C3	271
	Wheel Pitch	C4	119
	Center-pivot to Body Front	C5	71
Wheel	Diameter	D1	50
	Depth	D2	40
Mast	Height	E1	98
	Width	E2	62
	Depth	E3	50
	Position from font deck edge	E4	28
	Position from right deck edge	E5	28
Head	Height	F1	30
	Width	F2	60
	Depth	F3	50

Table 3.7: Table showing the external dimensions of components and subsystems obtained from the reference model

Feature	Size	Comments
Fasteners	M2 - M3	Full pan-slotted screw, hex nut and flat washer stack. M3 as first priority dropped down to M2 if the part dimensions or circumstances did not allow
Holes	M2 - M3	The holes were to match the fasteners in terms of size
Hole Clearances	0.5 mm	The clearance was made to be over a standard fit clearance in anticipation of spread of the surfaces of the part during the 3D printing process
Wall Thicknesses	Major: 5mm Minor: 3mm	Major walls were used where significant structural integrity was required, whereas minor walls were used both for less significant areas or areas where available space was a constraint

Table 3.8: Table indicating standards for common features across the entire design

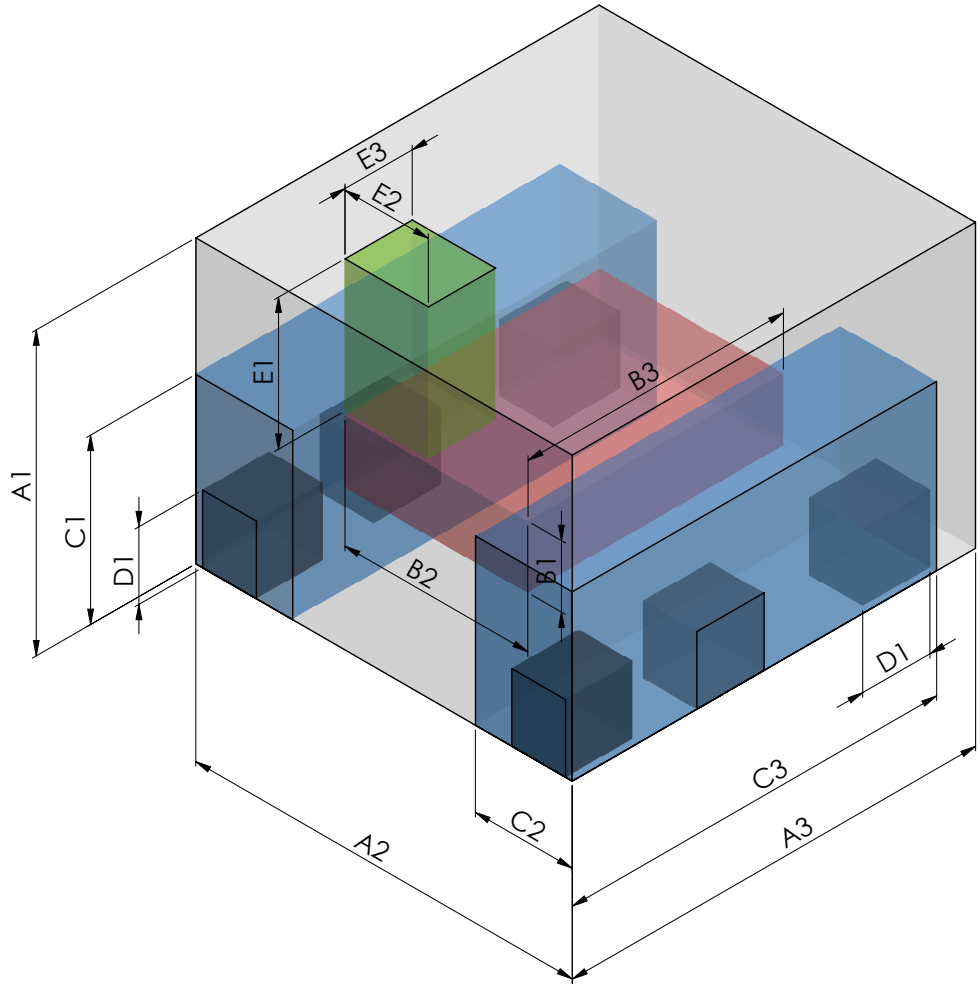


Figure 3.7: Isometric view of the 3D layout plan with a subset of the external dimensions shown

part of the Rocker-bogie principle as highlighted in Section 2.2.3. During the conceptual design phase, it was decided that this mechanism be constructed by connecting aluminium tube pieces to 3D printed joints, on the ends of which the pivots and struts could be attached for the wheels.

The design started with the identification of the components required and a map of how they would be fitted together which included the names of each part for identification throughout the process. Figure 3.8 shows this mapping, which can be considered as a lower-level conceptual plan of the system.

It was noted that the ratios of the tubing or beams on *Curiosity* (and rocker-bogie mechanisms in general) were important since it determined the system's range of motion, affecting the rover's ability to traverse the terrain. The reference model in [49] was not able to provide this kind of detail, and so the angles and positions of the centre-points of joints and axes of pivots and wheels had to be constructed from reference images of the rover. What was known was the distance of the wheels away from the side of the rover

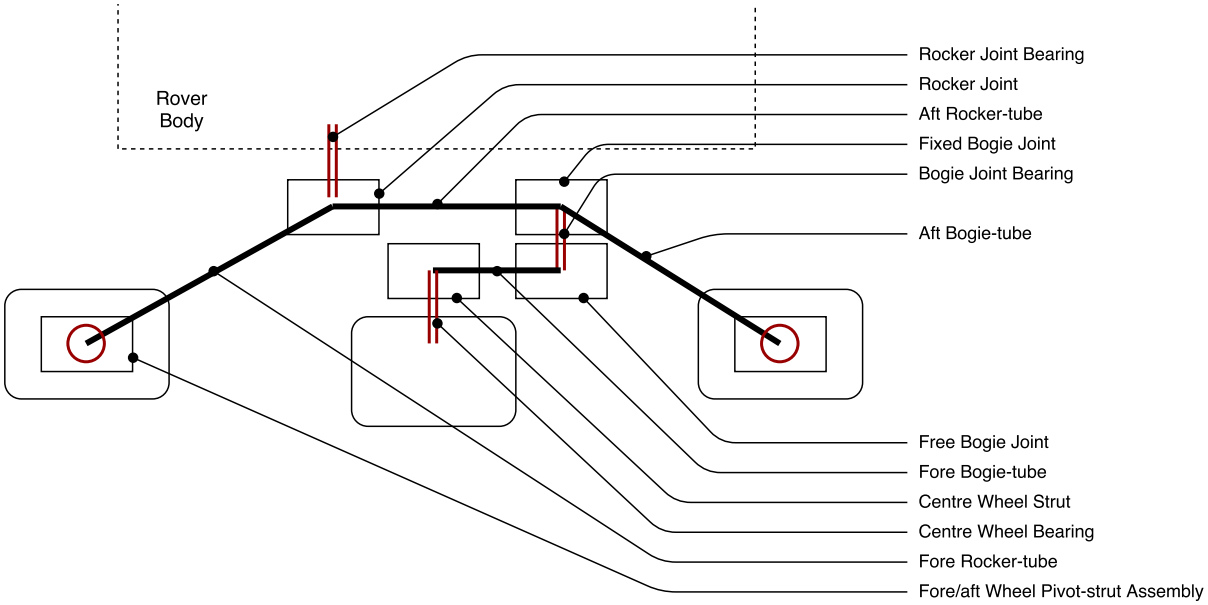
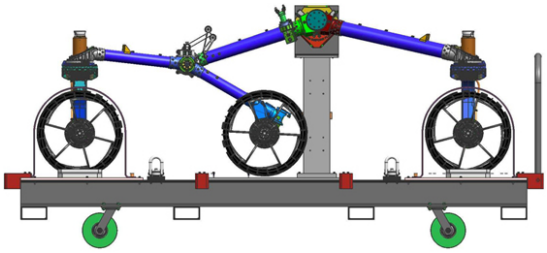
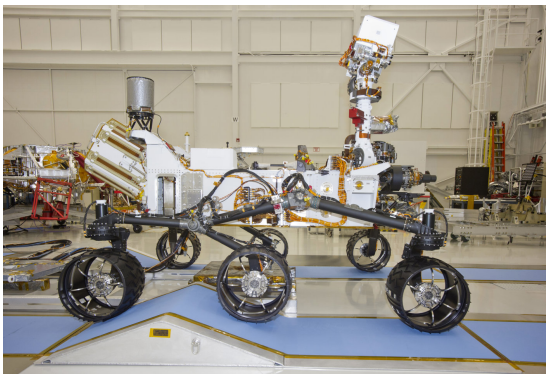


Figure 3.8: Schematic diagram of a top view of the suspension system

body, that the pivot centre-points were directly above the centres of the wheels and that the aft rocker-tube and the fore bogie-tube were parallel to the body of the rover in the x-y plane. Figure 3.9 includes the images that were used to find these details.



(a)



(b)

Figure 3.9: The two images used for obtaining the positions of joints and pivots of the suspension system in 3D space. [50] and [51] respectively.

A skeleton layout generated from the positional data obtained is shown in Figure 3.10 where a 2D sketch was created as a starting point from which a 3D sketch of the layout was formed. The skeleton sketch was rigid in the position whereby all three wheels were at rest on a flat, horizontal surface parallel to the rover body.

The joints, pivots and struts were then developed around the axes and centre-points in the sketch, a work-flow typical of the CAD package used.

Joints

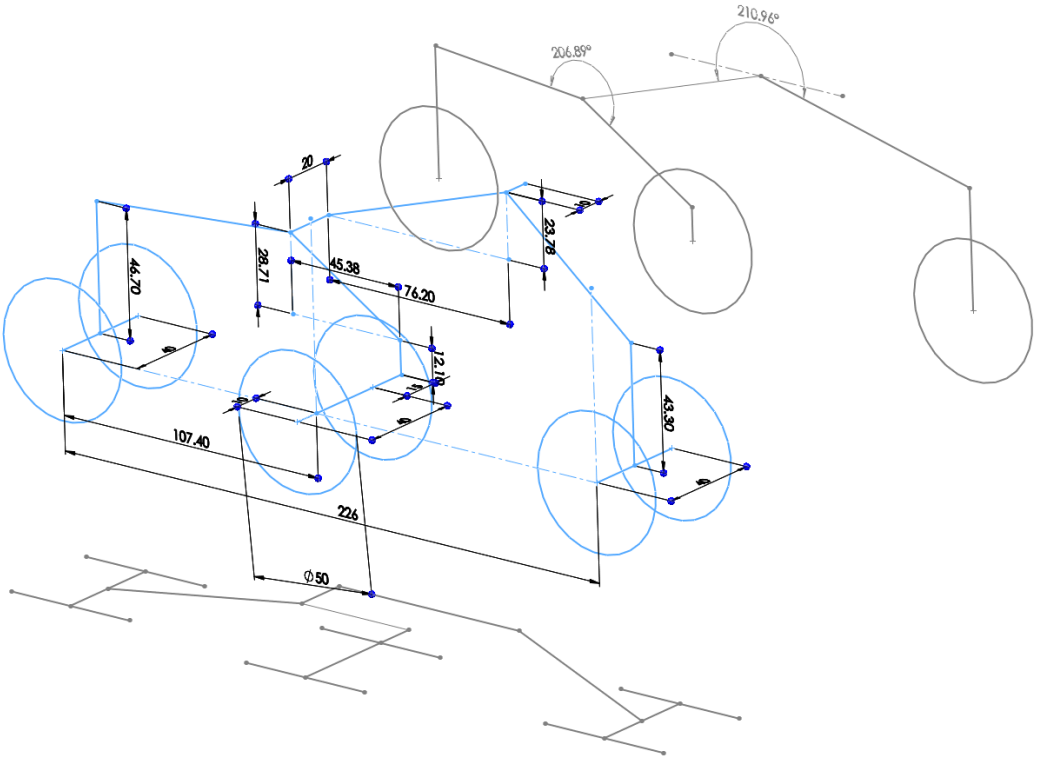


Figure 3.10: Isometric view of the 3D sketch of the generated suspension system skeleton used for positioning of designed parts

One side of the suspension system required three different joints: the rocker joint and fixed and free bogie joints. The bogie joints were the simpler of the three and provided a means for the bogie to pivot about one of the ends of the rocker. The fixed joint was fixed in rotation on the end of the rocker while the free joint rotated about a point on the free joint. The rocker joint allowed for the entire mechanism to pivot about a point on the side of the rover body as well as allow attachment of the differential bar in order to keep the two sides of the suspension system coordinated.

It was decided that the rotation of the joints be aided by use of bearings mounted on aluminium shafts. The aluminium would not add excessive amounts of weight to the system but still provide the rigidity required between the joints. Bearings were then required to be fixed into the rocker and free bogie joints with a shaft extending from the side of the body for the rocker joint bearings and another shaft mounted in the fixed bogie joint. Various methods of mounting bearings and shafts into 3D printed parts were considered, however, the most commonly employed technique involved press-fitting both types of components. At this point it was clear that printing the parts from a material which allowed for a certain amount of flexibility (i.e. less brittleness) would suit press-fitting bearings and shafts and reduce the chances of parts cracking when doing so. The press-fit holes and bores were introduced into the design after modelling the joints.

The joints were to host the aluminium tubes and thus a way of mounting them was required. Aluminium tubing dimensions were decided upon based on availability, cost,

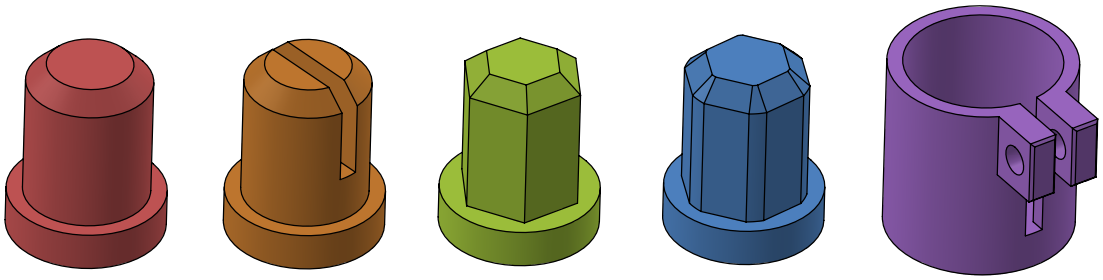


Figure 3.11: Render of the plug concepts considered for the attachment of aluminium shafts onto joints and pivots

weight and strength and the aim of keeping the suspension system in proportion was a contributing factor in this design choice. The size of aluminium was chosen to be a standard extrude of 15.88 mm outside diameter with a 1.62 mm wall thickness (making the internal diameter 12.64 mm). Two methods of fixing the tubes to the joints were considered: the first of which was to design a plug onto which the tube could be pressed and fastened and the other involved a bracket into which the tube would be placed and the bracket could have been tightened using a clamp or nut and bolt. The plug concept was chosen over the bracket due to size constraints which would have been exceeded if the latter were used. The plugs took advantage of the fact that the aluminium was tubular and minimised use of space outside of the diameter of the tube. However, using a plug might have introduced a weakness into the design in that cross-axis forces (bending moments) on the plug could damage, if not, tear the plug from the joint part. This would not have been the case with a bracket where these types of forces would have translated into forces parallel to the plug main axis. Despite the possible weakness, the plugs were used and care was taken to ensure that typical use would not affect the part in this way. Another reason for not using a bracket was due to anticipation of the plastic material used for the printing possibly deforming when tightened with the suggested fastenings. Plastic, among most other materials, offers greater robustness when compressed (as in an internal plug feature) as opposed to if it is put under tension [52].

Multiple plug shapes were considered, as a range of which are shown in Figure 3.11. The aim was to have the plug not require glue, as aluminium is not well suited to being glued using adhesives that work well with plastic parts. An ideal plug was one where just the press-fitting process was satisfactory in order to obtain a rigid attachment.

Chosen was the plug that was hexagonal in shape but had edges which were filleted to match the inside surface of the aluminium tube. This was a hybridisation of the cylindrical plug, which introduced a very low window of tolerance in the manufacture of the joints, and the simple hexagonal plug. The filleted edges increased the contact surface area with the inside of the tube, improving the effectiveness of the fit, whilst allowing for greater manufacturing tolerances. In case the plugs proved to be lacking the required strength, a hole could have been drilled down the centre of the plug and a steel rod could

be glued into place to strengthen the joint, specifically at the intersection of the middle body and the plug. A nut and bolt was added to the plug-tube assembly as in Figure 3.12 to improve the fitting and prevent rotation of the tube around the axis of the plug.

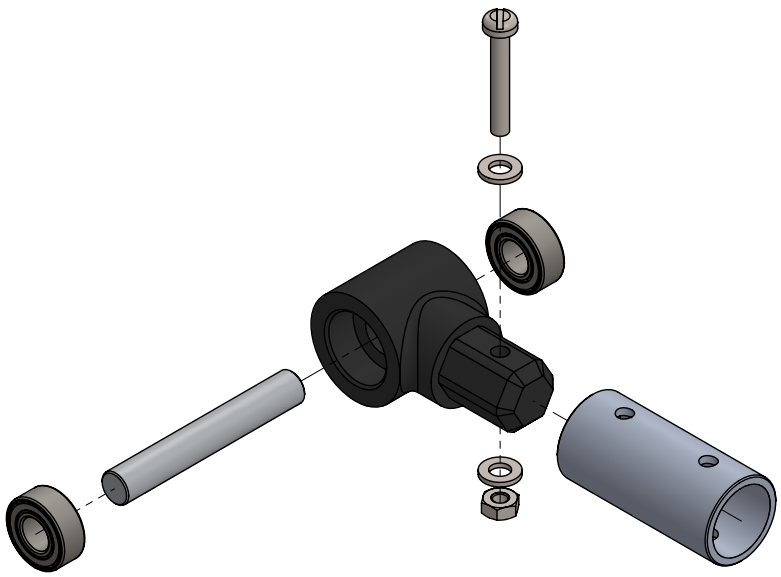


Figure 3.12: An isometric exploded view of the plug-tube-bearing assembly concept for the suspension system

An extension to the rocker joint was made in the form of a "fin" feature to allow for the connection of the differential system. The feature was added "in-place" in the 3D model after the differential has been added to the assembly so that correct alignment was ensured.

Finally, features for the press-fitting of bearings were added to the joints that required them. Bearings were chosen at this point to be of dimensions shown in Figure 3.13 in which the shaft diameter is also shown. A single bearing alone was not suitable to provide support against bending torques brought about when the shafts were to be put under load, thus each free-moving joint had two bearings on either extremity. A hole through the centre of the bearing bores of ⌀8 mm was included to allow the shafts to extend to the outwards facing bearings. The final designs for each of the three joints are shown in Figures 3.14, 3.16 and 3.15.

Wheels

Six wheels were to be designed, four of which were driven by the sub-micro servos (front and rear wheels) and the two centre wheels were to be mounted onto fixed shafts with bearings. All six of the wheels on *Curiosity* were actuated to ensure robustness of the driving mechanisms in a wider range of terrain types and traversal situations. It was also a feature of redundancy in that if one of the motors failed, the rover was capable of continuing operation. However, it was decided that for this model only the front and rear wheels would be actuated given the reduced power to weight ratio compared to that

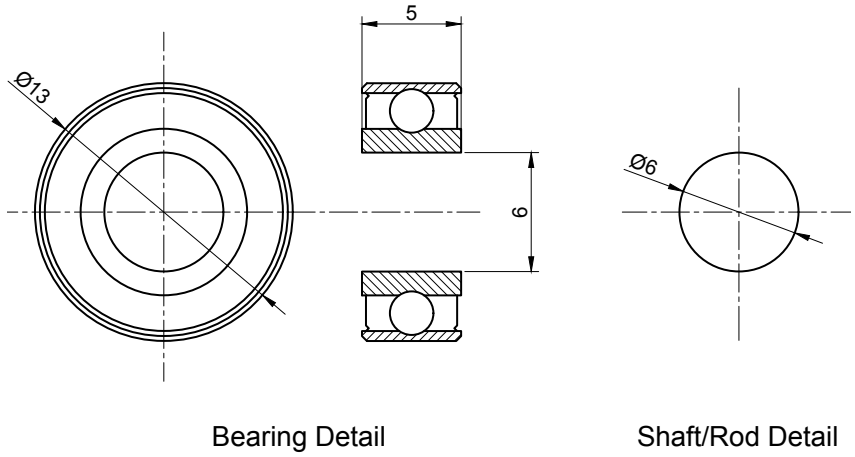
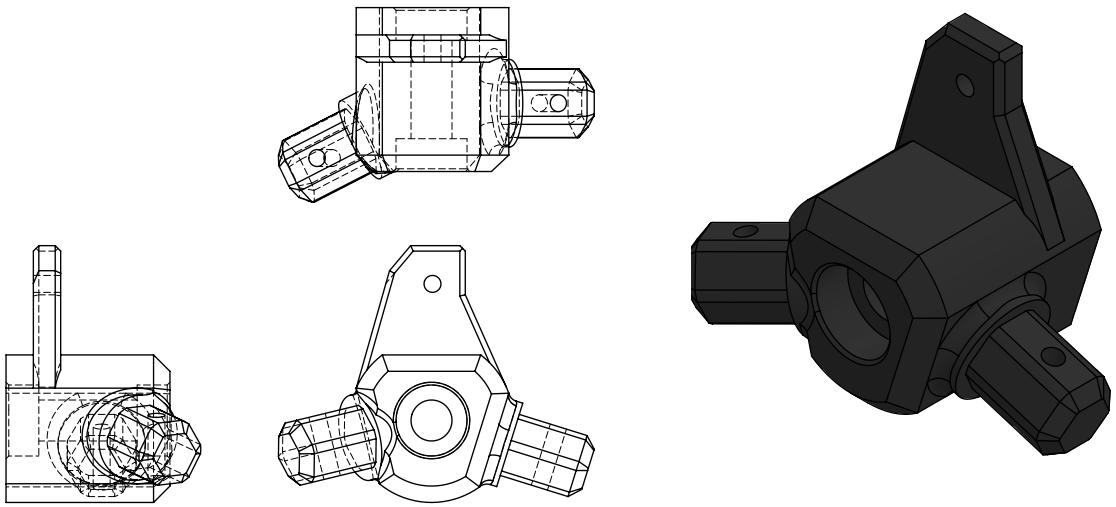


Figure 3.13: Detail of the bearings and shaft chosen for the entire design

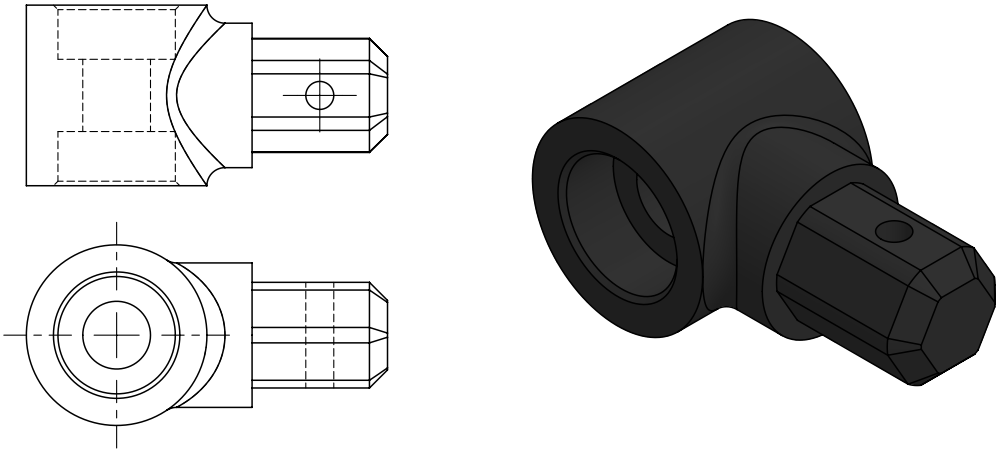


(a) (b)

Figure 3.14: Detailed drawings of the rocker joint component for one side of the suspension system

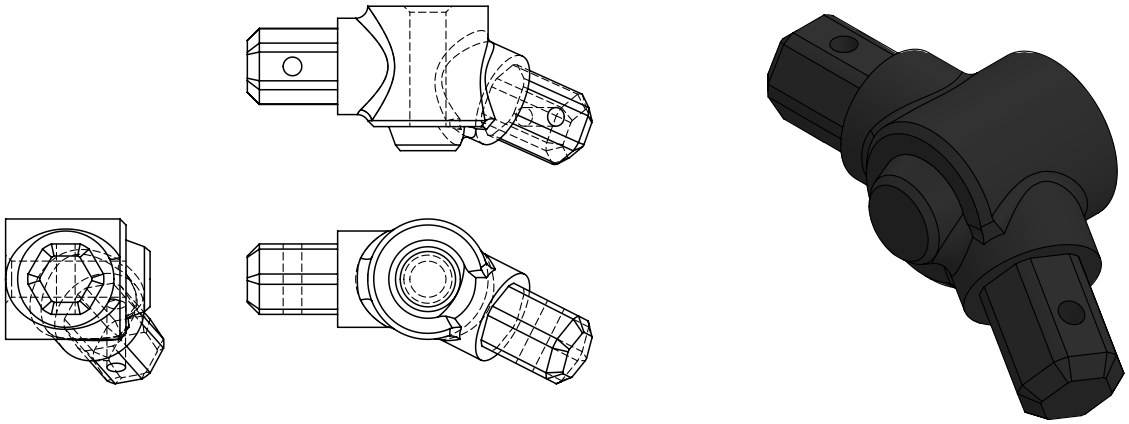
of *Curiosity*. Redundancy was not an issue worth the resultant extra servos and the incurred control complexity.

As mentioned in Section 3.2.1, the wheels were a great opportunity to make use of the aesthetic accuracy of additive manufacturing thus the wheel was modelled so as to replicate the cross-sectional curve of the outer shell of the wheel. Included were the tread patterns for traction as well as the morse-code emboss which read "JPL", used on *Curiosity* to acquire optical estimates of the distance travelled by the rover. Spokes and a centre cylindrical core was added to the inside of the outer shell, keeping the wheel as a single piece.



(a) (b)

Figure 3.15: Detailed drawings of the free bogie joint component for one side of the suspension system



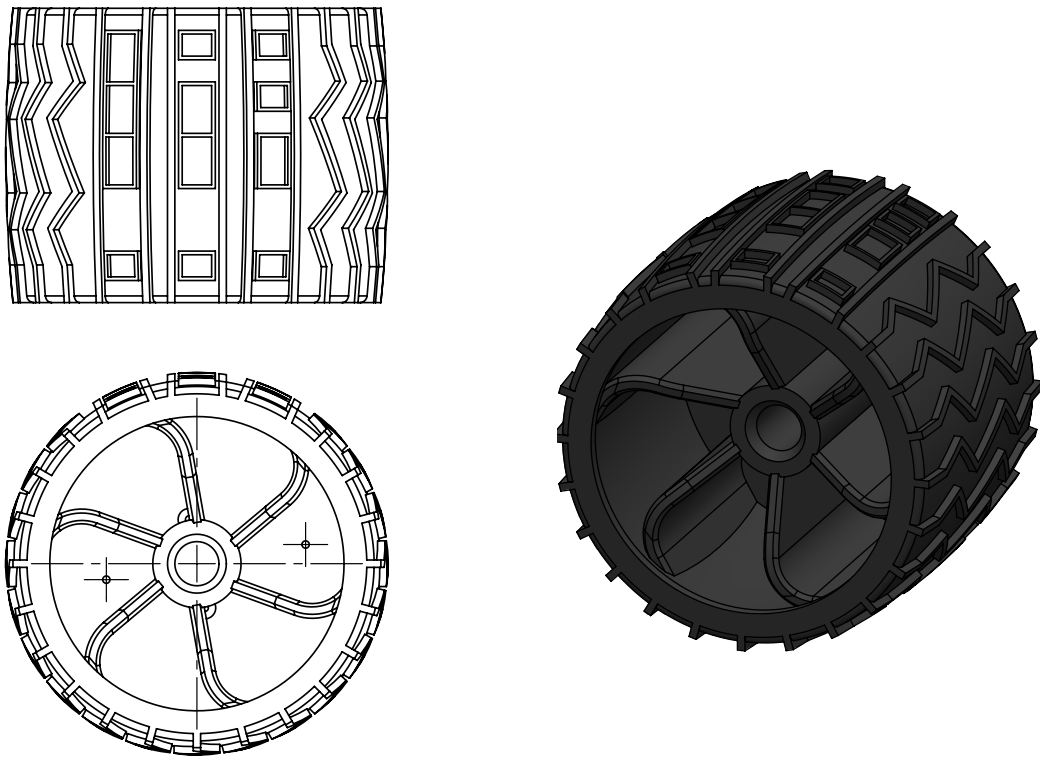
(a) (b)

Figure 3.16: Detailed drawings of the fixed bogie joint component for one side of the suspension system

The sub-micro servos were accompanied by servo horns that fitted onto the shaft of the servo. The horns were cross-shaped, a layout of which was taken advantage to provide support in the cross-axis plane. The horns were measured and holes were added to the four wheels concerned so that they could be mounted directly to the driving servos. The need for bearings on this assembly was countered by an estimate of the forces developed due to the rover's weight and it was anticipated that the servos would be capable of taking the estimated load without damage or wear. This also provisioned for easy replacement

of wheels and or servos should one of them be damaged.

The same bearing bores and centre hole as on the joint components was added to the cylindrical core of the centre wheels for mounting to the aluminium shafts. The press-fitting of bearings into the wheels was to be of the same nature as that of the joints. Figures 3.17 and 3.18 show the outer and centre wheel details.



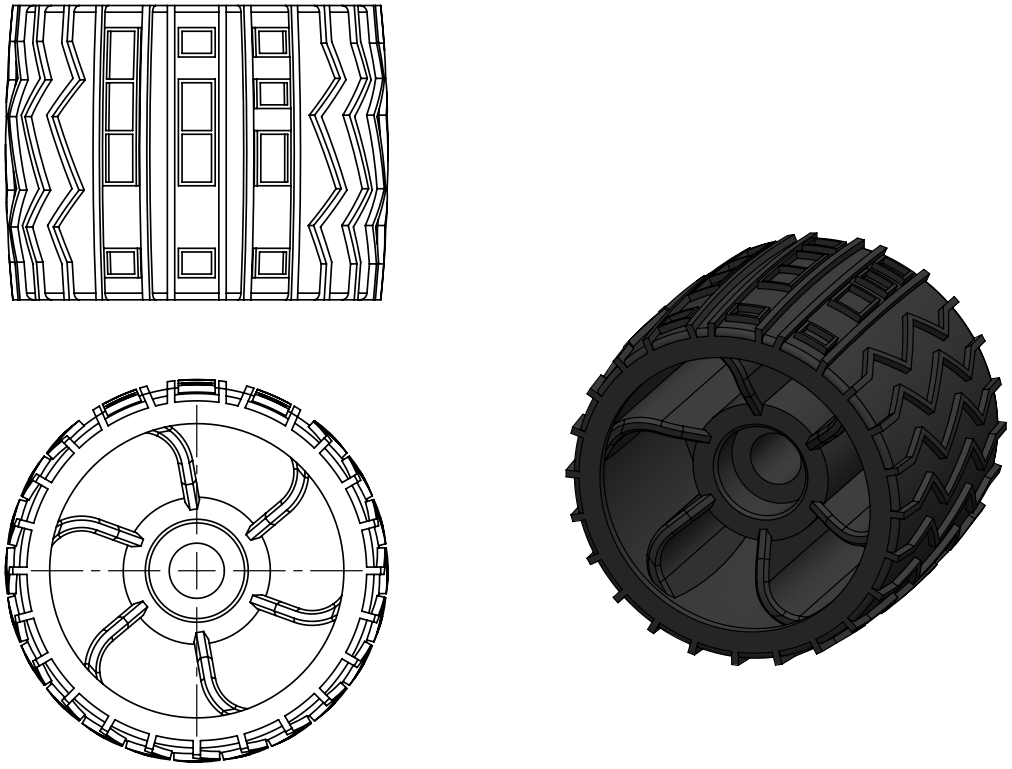
(a) (b)

Figure 3.17: Detailed drawings of the outer wheels

Pivots

Turning the wheel-strut assembly involved the pivot component which was required to allow for mounting of a sub-micro servo (steering servo) and to be attached to the ends of the fore rocker-tubes and aft bogie-tubes. The same concept for attaching to the aluminium tube as in the case of the joints was applied to the pivots thus the design consisted of a L-shaped extrusion with the plug extending from one of the outside flat surfaces. The other surface had a rectangular cut-out the size of the servo body, mounting holes for the servo and ribs for supporting the L-shaped extrusion. Figure 3.19 shows the pivot component detail for the front wheel assembly.

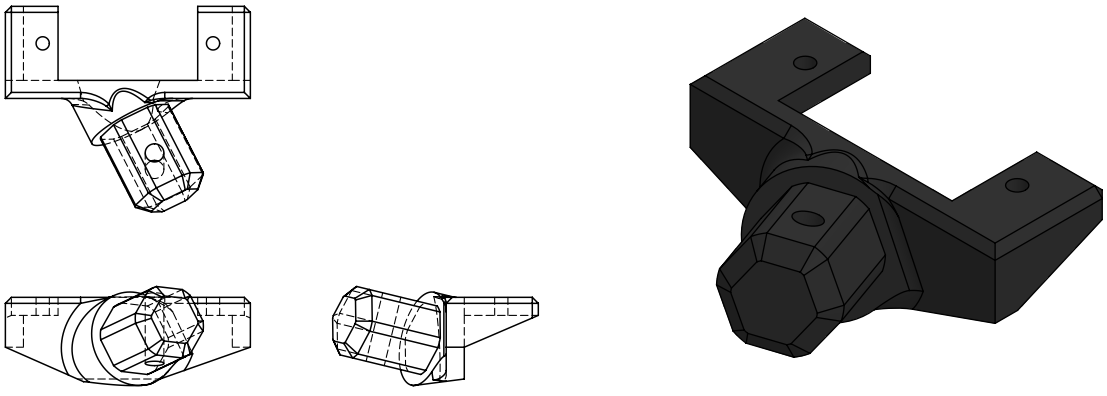
The front and rear pivots differed slightly due to the angle of entry of the rocker and bogie tubes towards the centre-points of the wheel assembly were different. Managing the differences in angles was aided by use of the 3D skeleton sketch. Again, the front and rear pivots from the left hand side were mirrored to produce parts for the right hand side



(a) (b)

Figure 3.18: Detailed drawings of the centre wheels

suspension assembly.



(a) (b)

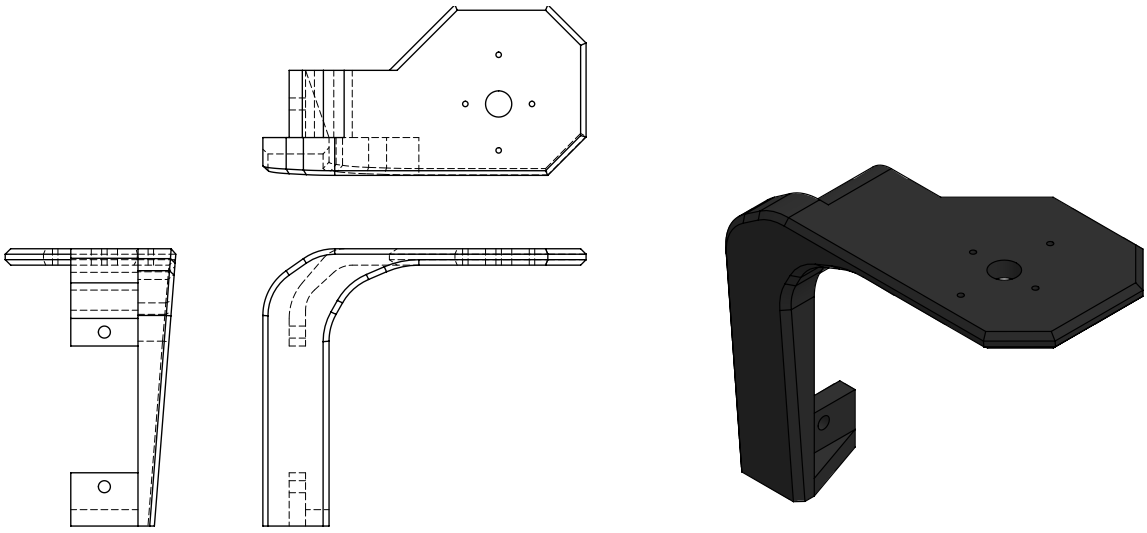
Figure 3.19: Detailed drawings of the wheel pivot component for one corner of the suspension system

Struts

Curiosity had and arcing "strut" for each wheel which curved from above the wheel,

underneath the pivot motor, over the side and into the inwards facing threshold of the hollow of the wheel. The strut was attached to the drive motor on the inside of the wheel, as close to the centre of the wheel for balance and minimisation of bending stress. The sub-micro sized servos would not fit inside the wheels of the model at its chosen scale, thus they had to be mounted on the outside of the wheel. The strut was required to provide a place to mount the driving servo and to be mounted to the servo horns of the steering servo.

The strut was designed to maintain the curved appearance as on *Curiosity* but to take into account the strength of the component. A flat platform above the wheel included holes for mounting of the steering servo's horn and this curved downwards into the strut section of the component. The strut curve consisted of a flat section to offer strength against bending in the typical direction as loaded (the x-axis) with a rib type extrusion from the rear facing side of the strut for increased support and tabs on which to mount the driving servo. The design was then mirrored for the rest of the corners and the final detail can be seen in Figure 3.20.

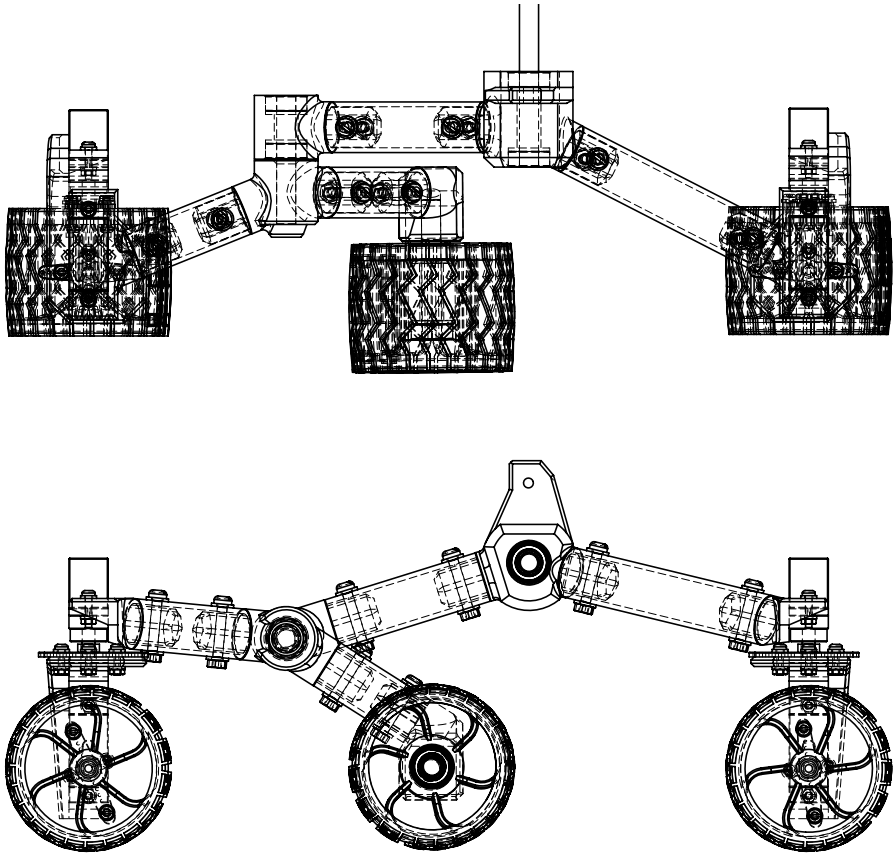


(a) (b)

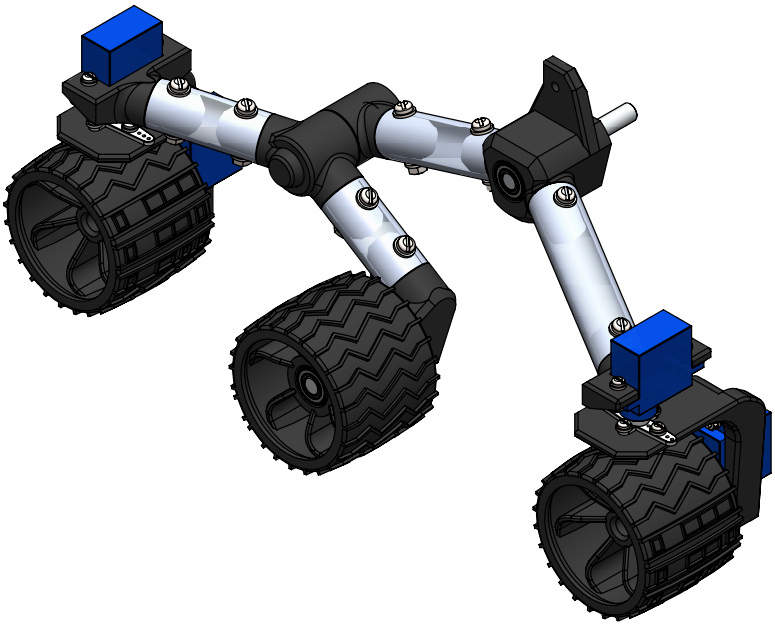
Figure 3.20: Detailed drawings of the wheel pivot component for one corner of the suspension system

Final Sub-assembly

Having designed the parts around the skeleton sketch, where they were kept fixed to preserve the angle details linking them, another assembly was created into which the parts were re-added and mated in a manner more typical of the way that they would be assembled. The assembly allowed simulation of the movement and this functionality was used to analyse the assembly for interferences and to ensure that the structure moved the way it was required. Figure 3.22 shows the linkages positioned as if the subsystem was navigating over an obstacle. Each individual part was then mirrored and assembled to form the opposite side of the suspension system. The CAD package ensured that the mirrored parts were dynamically updated should one of the original parts have changed.



(a)



(b)

Figure 3.21: Detailed drawings of the working, dynamic assembly one side of the suspension system

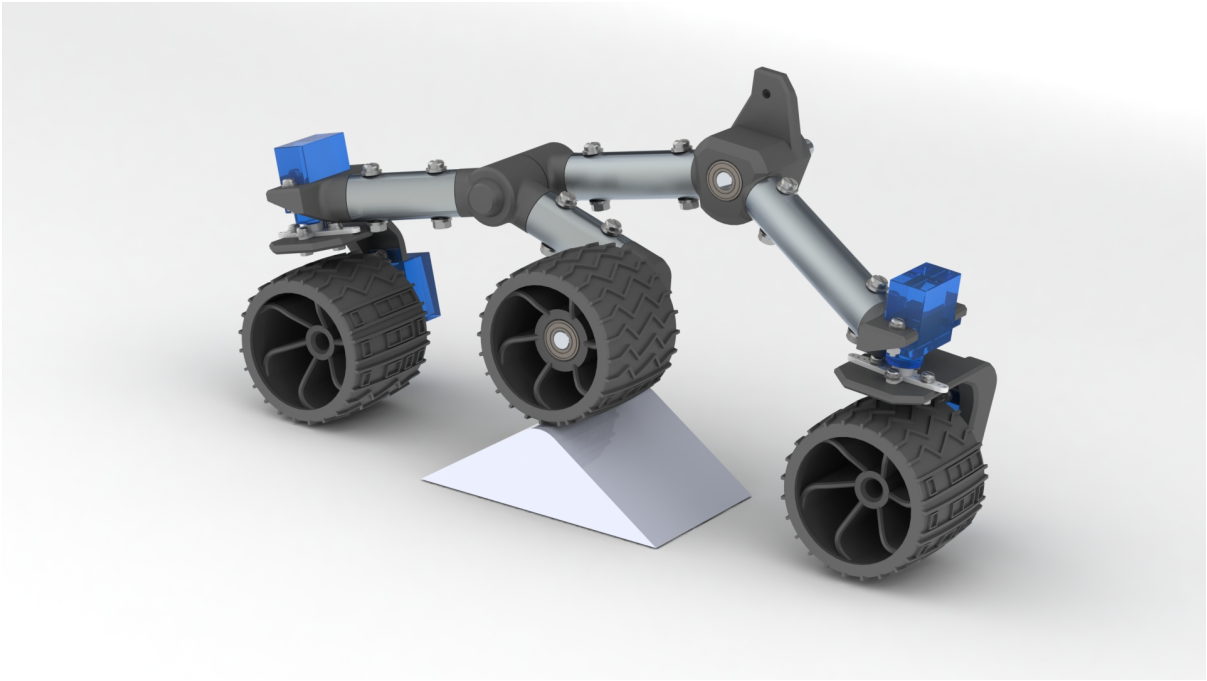


Figure 3.22: Render of one side of the suspension system navigating over an obstacle

Differential

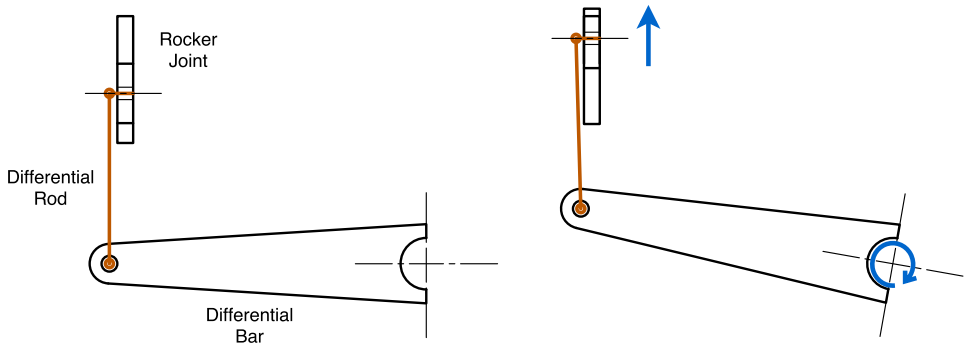
Design of the differential involved a partially completed design of the body structure which contained details of the mounting points of the suspension system and thus the position of the connection point on the rocker joint, as well as the centre point of articulation of the differential bar as acquired from the reference model. The concept chosen for this subsystem included a 3D printed differential bar, printed hinges and threaded-bar (rods) attached to the hinges to link the differential to the rocker joint. The need for hinges was to provide the rods with the degrees of freedom required taking into the account the arced motions of an end of the differential bar in the x-y-plane and rocker joint in the x-z-plane. The articulation of the differential system and the resulting motion of the rods is depicted in Figure 3.23, each of the two views showing two positions in the motion.

Differential Bar

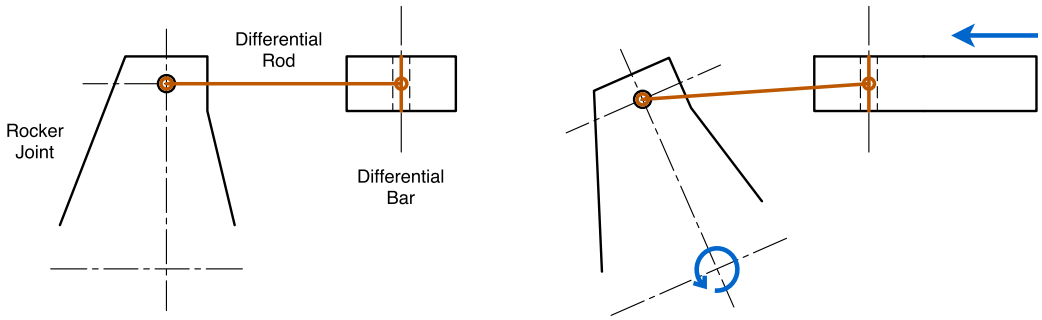
The design of the differential bar stemmed from the partially completed rover deck, specifically with respect to the width of the body and therefore the span of the rocker joints' differential connection points. The bar took on the tapered shape as on *Curiosity* with the wider section in the centre and narrower ends. A hole was added for the addition of a bearing in the centre of the bar and holes for the hinges added to the ends. The bearing was to be mounted to a short shaft which was to be fixed into the deck of the rover. Figure 3.24 shows the detail of the differential bar.

Hinges

Hinges were designed for each end of the rods, one end on the rocker joint side and the other for the differential bar. The hinges allowed rotation about specific axes, the y-

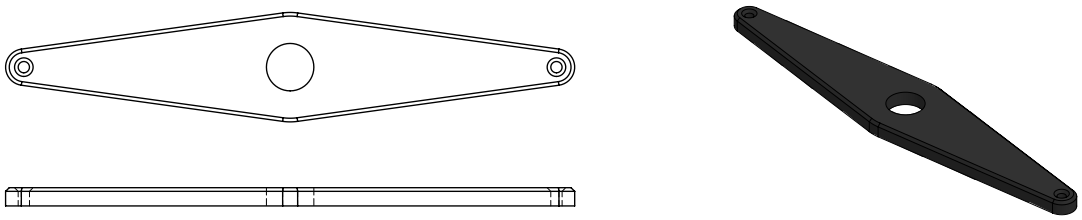


(a) Top view



(b) Side view

Figure 3.23: Diagram showing the motion of the differential and the resulting motion of the differential rod



(a) (b)

Figure 3.24: Detailed drawings of the differential bar component in the differential system

and z-axes on the bar side and the y-axis on the joint side. Figure 3.23(a) demonstrates that the bar also rotates about the z-axis which would require the hinge at the rocker joint to allow for this motion. However, this was not included in the design after confirming through calculation that the amount by which the bar would have rotated in this direction

at the maximum angle of articulation of the differential bar ($\approx 9°$) did not justify the resulting added complexity in the hinge design.

The single axis hinge on the rocker joint side amounted to a right-angled piece with a hole in one of the flat sections for fastening it to the rocker joint and another hole on the other flat section for fastening of the threaded-bar rod component. Slotted bolts and nuts were used for the joint hole with washers between the joint and the hinge part to minimise the friction between the parts despite small degree of motion that would be effected on this coupling. For the rod, two nuts on either side of the piece were used to fix the threaded-bar.

The two-axis hinge on the differential-bar end included a part with two tabs with holes, one above and one below the end of the bar and a flat section extending perpendicular to the axis made between the two tab-holes. The additional section included a third hole to allow attachment of the second part responsible for mounting the rod. The second component included a blind hole into which the rod could be place and a cutout halfway down the length of the hole to allow for a nut to be pushed into place. This meant that the rod could be screwed into the piece, the purpose of which included easy assembly and breakdown if required as well as allowing the extension of the rod to be adjusted. Therefore, the differential system could be adjusted to balance the rover body about the rocker joint pivot points.

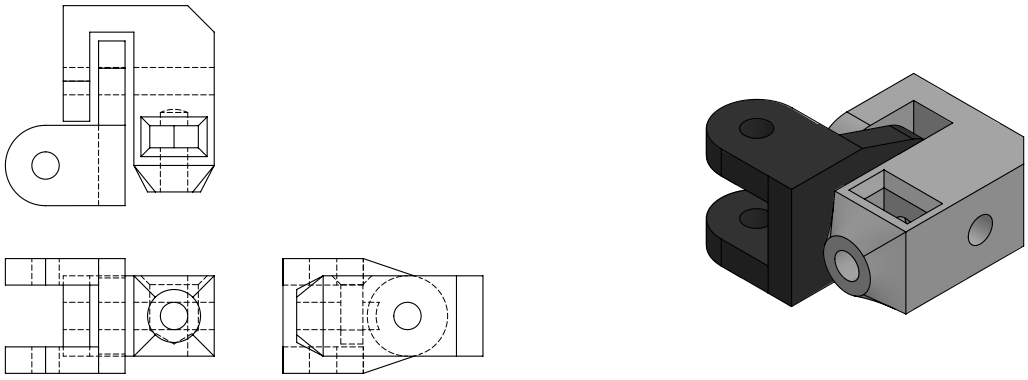
Detailed drawings of both hinge assemblies can be seen in Figures 3.25 and 3.26.

(a) (b)

Figure 3.25: Detailed drawings of the single axis hinge component in the differential system

It must be noted that other hinge techniques were considered for this subsystem, one of which included a ball and ring-socket joint which would have provided the required axes of rotation in a single coupling (for each end). The idea was not used for the design due to unavailability of the joints, particularly of the size required. The hinge technique was simpler in design and leant itself well to the chosen manufacture method.

Final Sub-assembly



(a) (b)

Figure 3.26: Detailed drawings of the two-axis hinge component in the differential system (without fasteners)

Figure 3.27 shows detail of the differential sub-assembly as generated in the CAD package used and was analysed for interferences and functional integrity.

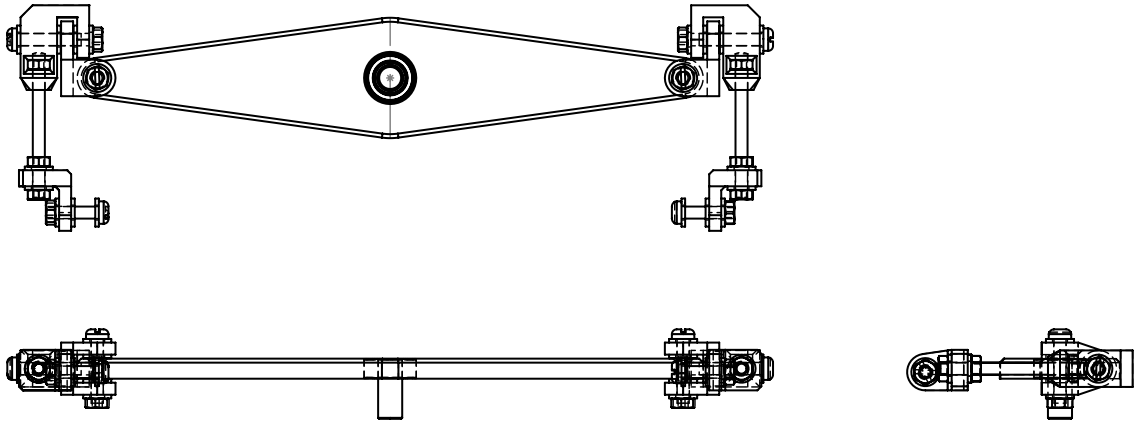
Head and Neck

It was decided during the conceptual design phase that both the neck and the head assembly be 3D printed and mounted to the top of the rover deck. Since there were two axes of rotation of the head required, as well as at least one piece to facilitate mounting the assembly, three separate components in total were required (a minimum). The assembly also aimed to minimise the amount of movement that components that facilitated mounting of the servos were subjected to for friendlier cable routing. At the same time, the ease of 3D printing such an assembly had an effect on the approach taken when designing the structure and configuration of the components as well as the components themselves. As a result, the configuration designed consisted of a static mount piece, a middle section, referred to as the neck, which allowed mounting of both servos for the two axes of rotation and a head sub-assembly to mount the camera and the ultrasonic sensor.

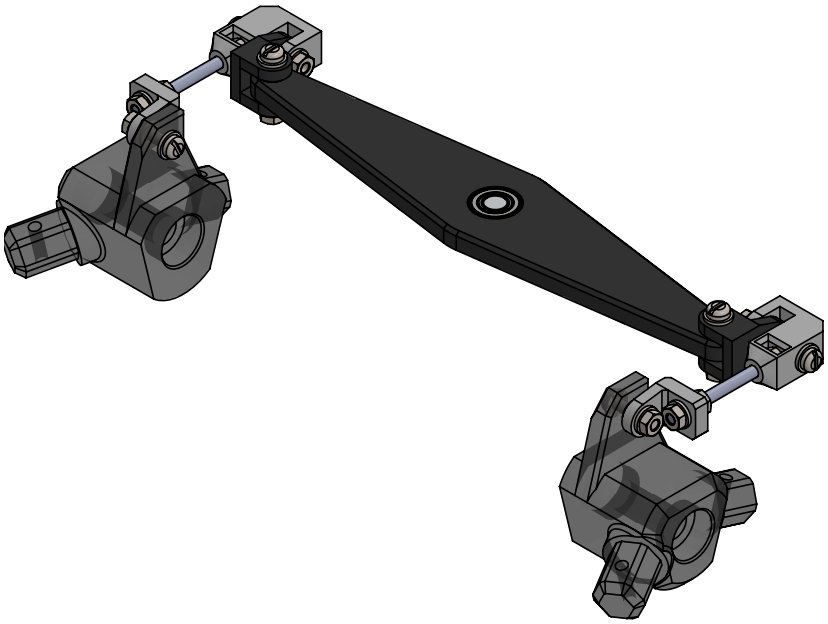
Neck Mount

The neck mount piece provided the support for the neck and head assembly in its entirety. Besides including holes for mounting to the deck of the rover, the component featured a small blind hole at the top center surface into which the shaft of the pan-axis servo could fit and be fastened to. A smaller hole right through the center allowed the servo shaft screw to be tightened so that the shaft did not spin in its mount.

Figure 3.28 shows the detail of the neck mount component.



(a)

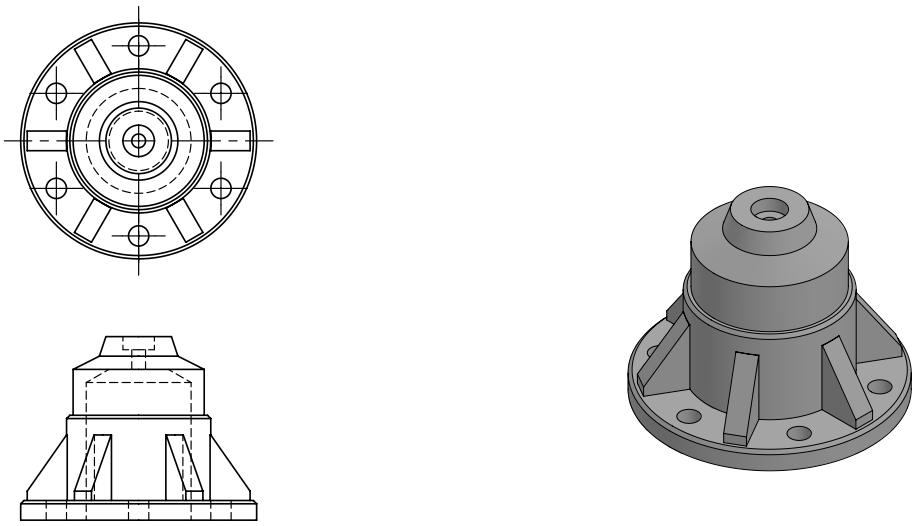


(b)

Figure 3.27: Detailed drawings of the working, dynamic assembly of the differential system (included in the isometric view in (b) are the rocker joints as part of the suspension system)

Neck Hinge and Actuation

The center piece in the neck-head assembly, referred to as the neck hinge, was designed to facilitate the mounting of two servos: one for pan movement and the other for pitch



(a) (b)

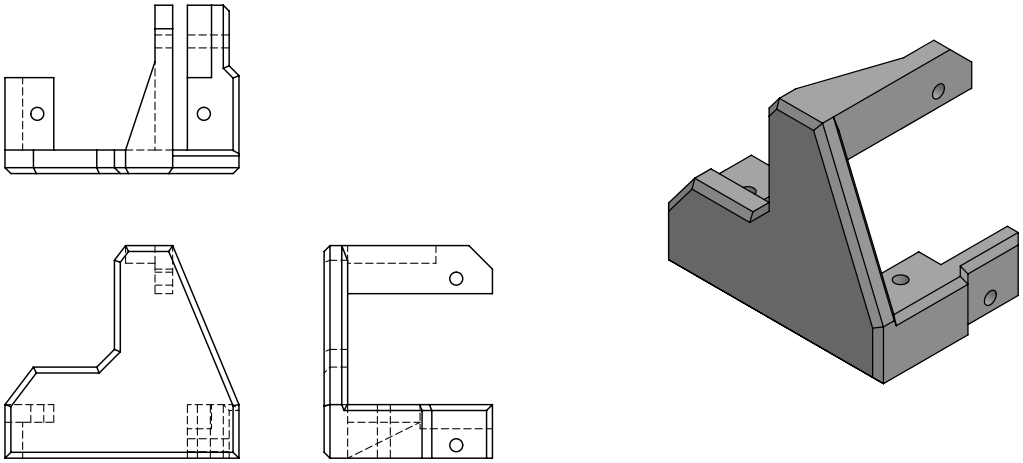
Figure 3.28: Detailed drawings of the neck mount component in the head and neck sub-assembly

movement. As discussed, the pan-axis servo which by definition was facing downwards towards the rover deck surface would fit into hole in the top of the neck mount and suspend the rest of the neck-head assembly. Fitted into the neck hinge component was the pitch-axis servo which was displaced from the pan-axis servo by 90 degrees. Multiple configurations involving the placement of the servos were considered and one which had the smallest volumetric footprint as well as required the least amount of 3D printing material was chosen for obvious reasons. The configuration had the servos positioned flat against each other with support material positioned at the servo mounting holes, the assembly of which can be seen in Figure 3.30. Figure 3.29 shows the detailed drawing of the neck hinge component without the servos.

Head

The head sub-assembly provided an enclosure for the web camera as well as a surface onto which the head ultrasonic sensor was mounted. As mentioned, the very least number of components required for the two degree of freedom assembly was three, however, the head sub-assembly required two additional pieces. These components were added to reduce the complexity of the components and therefore improve on 3D printing time and material use.

The head enclosure consisted of two components: a flat base and a box-like canopy. The canopy included an arched cut-out which allowed the camera, mounted on the inside, to be exposed to the outside world. The canopy was designed as a separate piece so that it could be removed to install and access the camera. As a minimisation of the footprint



(a) (b)

Figure 3.29: Detailed drawings of the neck hinge component in the head and neck sub-assembly

required by the fastening of the canopy to the base component, inset neodymium magnets were placed at all four corners of the canopy and base components such that the canopy clicked onto the base. Additionally, the canopy was designed to be as photo-realistic as possible to the head on *Curiosity* as this was one of the rover's recognisable and characteristic components from an aesthetic perspective.

The second extra piece included in the head sub-assembly was a right-angled bracket designed to be mounted to the shaft of the pitch-axis servo using a cross-shaped servo horn and fastened to the bottom of the head base component.

Figures 3.31, 3.32 and 3.33 show detailed views of the three head components.

Final Sub-assembly

The final sub-assembly of the neck and head components, including the servos is shown in Figure 3.34.

Body

The body was a significant part of the rover, designed as a mounting platform for most of the rover's other mechanical and electronic components. The conceptual analysis of the proposed ideas for the assembly of the body settled on the use of acrylic sheet. The body was rectangularly box-shaped and, for the purpose of access to the electronics which were to be mounted internally, the bottom was left open. This meant that five separate panels were designed to be cut from the acrylic sheet and later glued together to form the

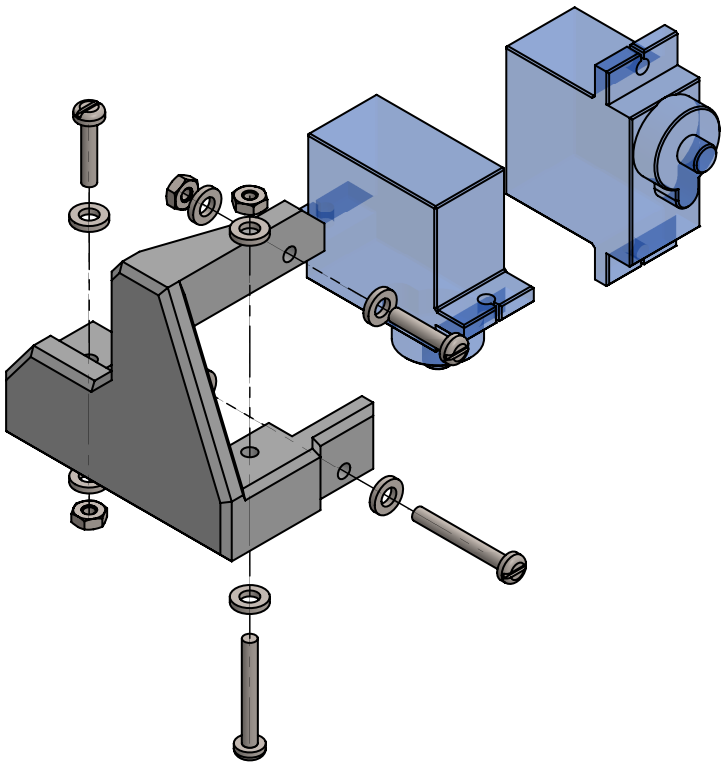
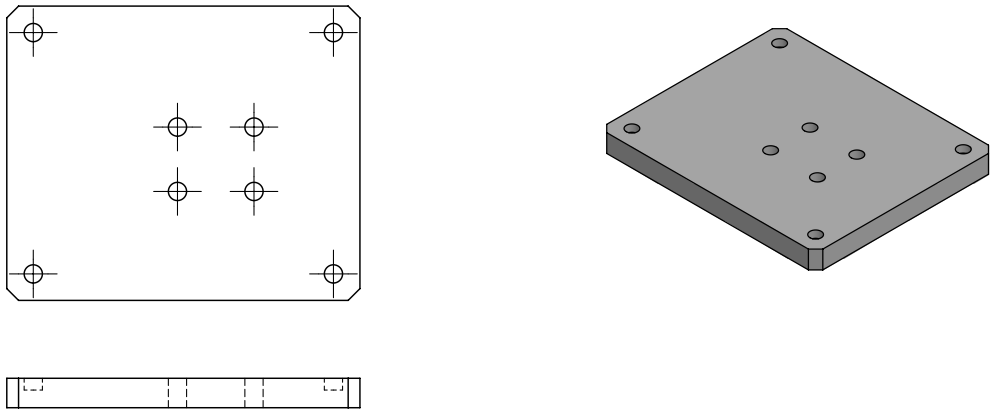


Figure 3.30: Exploded isometric view of the neck hinge and servo assembly

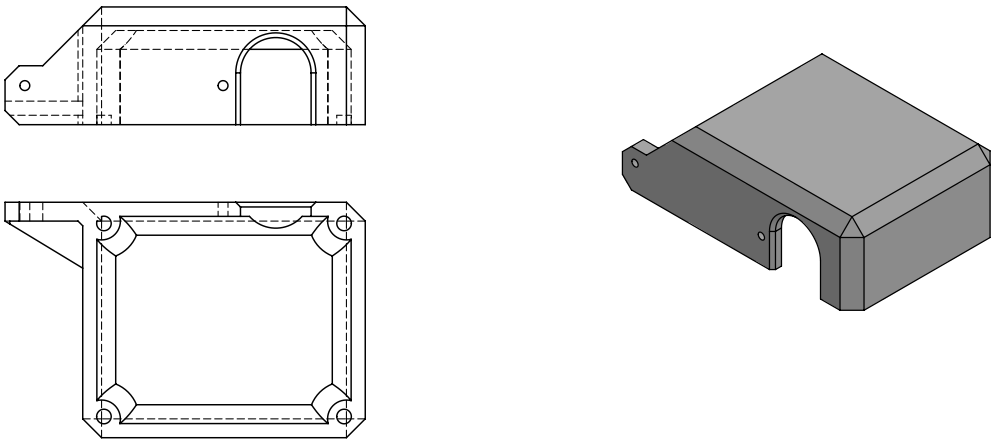
structure. Once finalising the dimensions of the panels, the required mount points were collated and added to the design. Mounting requirements for the body were as follows:

- **Deck/Top Panel:**
  - Neck and head assembly
  - Differential bar midpoint shaft
  - Aesthetic detail components
- **Left and Right Panel:**
  - Suspension rocker hinge shaft
  - Bottom cover magnet mount pieces
  - Aesthetic detail components
- **Rear Panel:**
  - Rear ultrasonic sensor
  - Aesthetic detail components
- **Front Panel:**
  - Front ultrasonic sensor



(a) (b)

Figure 3.31: Detailed drawings of the head base component in the head and neck sub-assembly

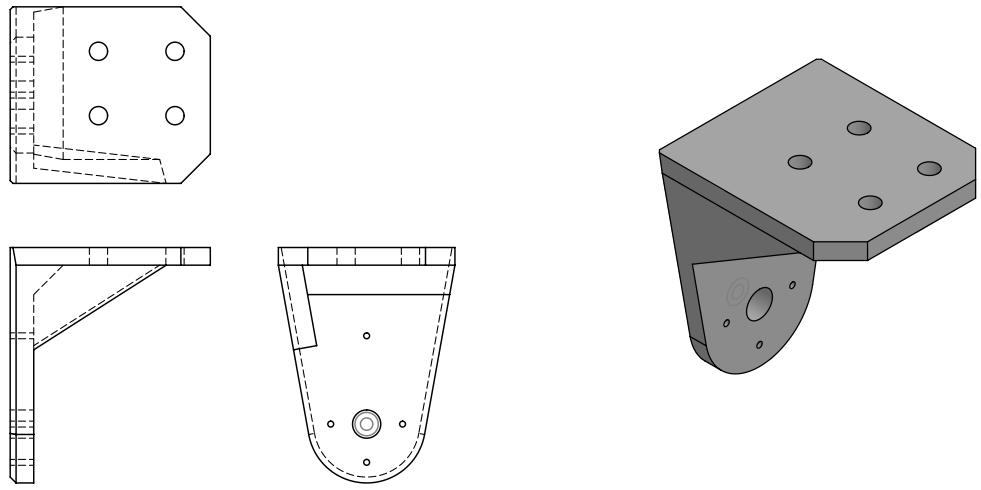


(a) (b)

Figure 3.32: Detailed drawings of the head canopy component in the head and neck sub-assembly

- Aesthetic detail components

Aesthetic detail components were those designed to add realism to the rover body and are discussed further in Section 3.3.1. A series of slots were added to the edges of each of the panels which fit into the those of adjacent panels for structural integrity and positioning



(a) (b)

Figure 3.33: Detailed drawings of the right bracket component in the head and neck sub-assembly

accuracy whilst glueing. Figure 3.35 shows the detailed drawings of each of the panels, the same drawings used for laser cutting the 3mm acrylic sheet.

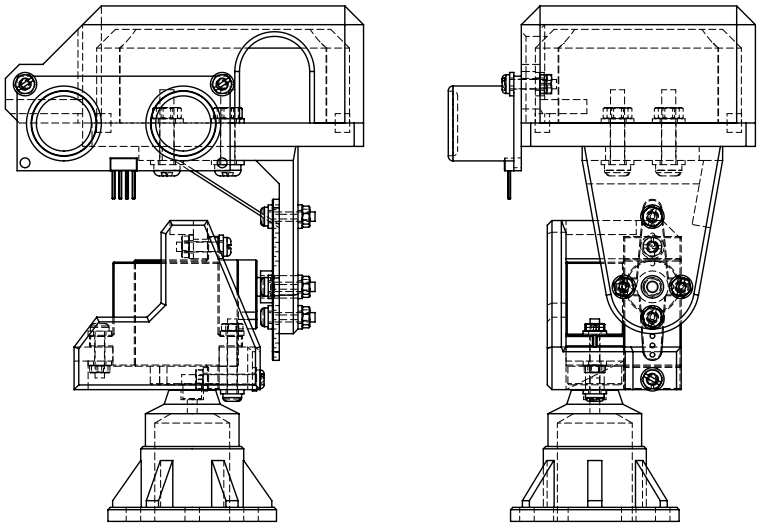
Aesthetic Details

To achieve the realism desired for the rover model, a number of extra components were designed to be 3D printed and mounted to the rover body component. The design and construction of the body made this suitable and a satisfactory level of realism was possible due to the 3D printing manufacture of the components. The following aesthetic detail components were designed, the drawings of which are not included:

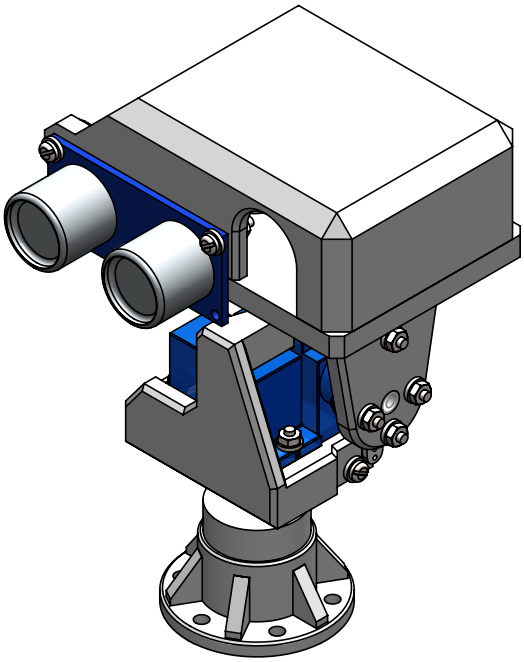
- MMRTG
- Rear shoulders
- Fore shoulders
- Four rover deck components
- Observation tray
- Panel detail pieces for the front, left and right panels

3.3.2 Electrical Design

The electrical component of the rover involved supplying power to the RCE board, actuation, sensors and camera and included the electrical interfaces between these components.



(a)



(b)

Figure 3.34: Detailed drawings of the working, dynamic assembly of the head and neck system

This section covers the design of these interfaces, the supply of power and the detailed choice of electrical hardware and components for this part of the project. Due to project time constraints the driving principle behind all electrical hardware was COTS design and the intention of turning towards complete and built components and systems before falling back onto systems built from first principles as a part of the project. This was also beneficial for individuals wanting to get involved as part of the open sourcing of

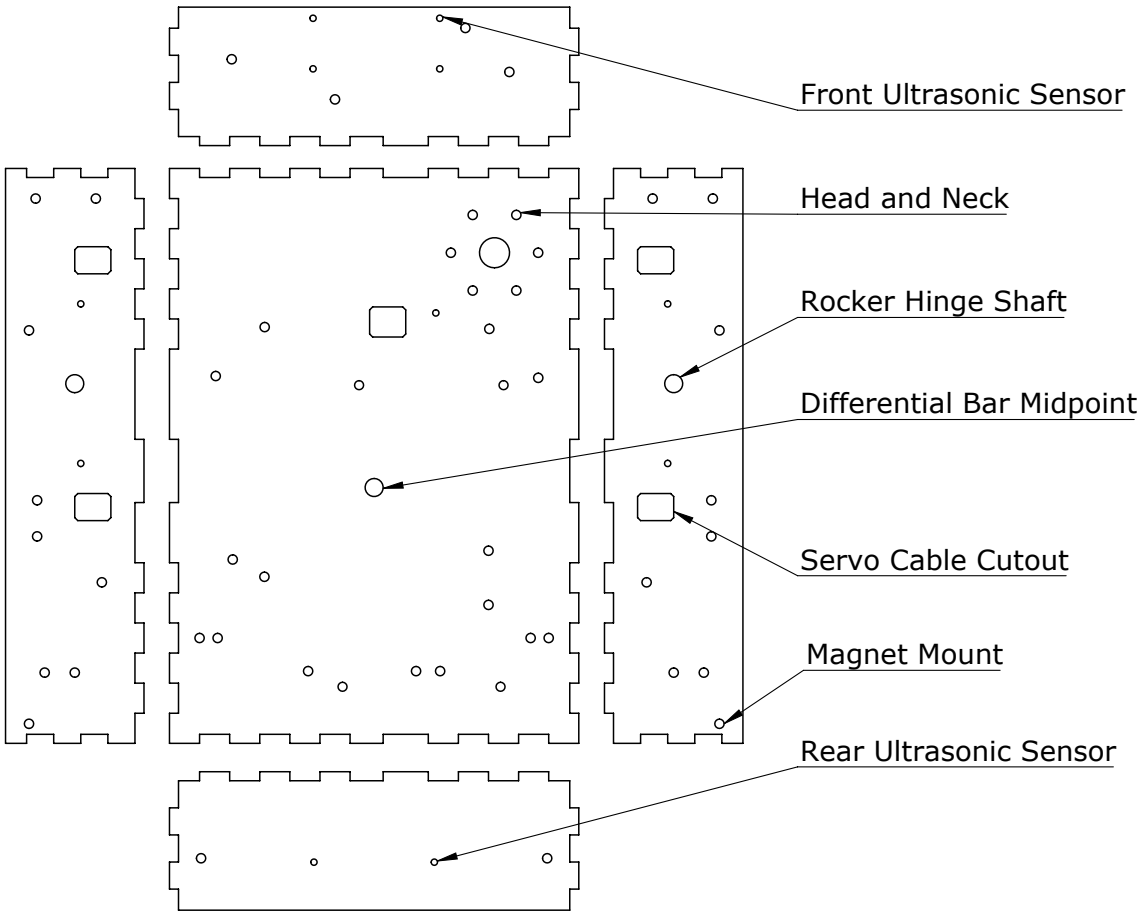


Figure 3.35: Drawings of the five acrylic sheet panels including labels indicating the mount points

the project. Including as many COTS components that are available to as many people as possible improved the accessibility of the project and the notion of education and outreach as a whole.

Actuation

The device for position- and velocity-based actuation during the conceptual design phase was servos of the sub-micro size and after investigation into the availability of multiple models of the servo type, the Hextronics HXT900 servo was chosen. The servo, a 9g, plastic-geared component requiring a 3V-6V supply, offered 180 degrees of movement at a maximum of 1.6kg per cm of torque. The range of motion meant that is was suitable for the steering of the four corner wheels as well as pan and pitch actuation of the head above the neck sub-assembly. However, the decision to use the servos for driving the four corner wheels meant that the servos required continuous rotation and not the limit range. The continuous motion was also required to be controlled from the perspective of velocity and not position. It was found that servos designed for continuous rotation were available but were not suitable for the project budget. An alternative solution was found, whereby

the range-limited HXT900s could be altered to result in velocity controlled continuous servos. These were used and the alterations involved are discussed in Section 3.5.

Standard communication applied for the HXT900 servo component, an analog-type device. The servo required power and ground connections and a third line for a PWM signal. The pulse width determined, in the case of the range-limited servos, position, and velocity in the case of the modified continuous servos. This signal was to originate from the Intel Edison board, however, the low level implementation of PWM signal generation on the device had significant limitations which meant that an alternative means had to be found. Additionally, the Intel Edison Arduino breakout board only offered six PWM signal pins and the rover actuation system required ten. The solution was to use a 16 channel PWM extension board which was compatible with the header and pin layout of the Intel Edison board, allowing for flexible control of many servo components. The PWM extension, manufactured by Adafruit, interfaced with the Intel Edison by means of a two-wire I^2C connection which meant that the rest of the Intel Edison board's pins were available for use elsewhere. The PWM extension also included short-circuit and reverse-polarity protection for increased system robustness as well as a breadboard section which was used for multiple other electrical subsystems discussed further.

Sensors

The three HC-SR04 ultrasonic proximity sensors provided elementary spatial awareness to the rover, specifically to the control system as part of the RCE. The sensors, mounted to the front and rear body panels and to the front of the head, operated at a 5V supply and communicated via an analog interface similar to that of the servos. Other than power and ground connections, the sensor had two other pins, one for input trigger signals and the other a range measurement output signal. The sensor required the trigger pin to be held high, at TTL level, for a minimum of 10μS to signal to the on board devices that a measurement was desired. The sensor then emitted a series of 40kHz sonic bursts and measured the time it took for the six bursts to arrive back at the sensor audio interface. The sensor then generated a signal of a single pulse over the output line, the length of which was proportional to the range measured. The pulse duration varied from 116μS to 23.2ms and the RCE was required to time the length of this pulse in order to obtain the measurement.

However, as discussed in this report as part of the software design phase, the software runtime (as well as the operating system) did not cater for signal input timing of the required accuracy. An I^2C interface-able backpack device was available which converted the analog pulses to digital signals easily read in by the Intel Edison board. Due to local unavailability of this device, a fallback solution was to design and develop a pulse to analog voltage converter which could be fed into the ADC device on the Intel Edison board. The design involved triggering the sensor at a specified interval passing the output pulse through an RC filter circuit to result in a transient analog voltage signal representative of the average period of the pulses over a designed time-constant. This also decoupled the RCE trigger timing from the timing required for reading the resulting measurement signal therefore the voltage level from the pulse to voltage converter could be sampled at a point which was most suitable. The RC filter included included an LM358 operational

amplifier in a unity-gain configuration as a buffer for the input pin of the Intel Edison board. Due to its simplicity, the schematic is not shown.

Power

The crux of the electrical component of the rover was the supply of power to all the electrical devices. As per the specifications, the rover was required to be self-powered and this was achieved by means of a battery to be mounted to the inside of the rover body. The battery, a three-cell, lithium-polymer type with a capacity of 1000mAh, was chosen based on its popularity in the remote-control vehicle industry and featured a high current output capability (low internal resistance), a good supply choice for servo motors which draw noisy current and induce high voltages due to their nature of operation. Operating between 12.4V and 11.3V, the battery was compatible with a wide range of balancing chargers.

In order to design the power supply network, a table (Table 3.9) of voltage requirements and approximate current usages was drawn up and totalled to verify the suitability of the battery as well as determine if additional electronics was required.

Device	Voltage Requirement (V)	Current Usage (A)
10× HXT900 Servo Motors	3-6	2.5
Intel Edison Arduino Board	7-15	0.3
3× HC-SR04 Ultrasonic Sensors	5	0.045
Web Camera	5 (USB)	Max 0.5 (USB)
3× Pulse to Analog Converters	5 (TTL)	0.05
	TOTAL	**3.445**

Table 3.9: Table showing the voltage supply requirements and approximate current usage of each of the electrical devices

It must be noted that the Intel Edison board included on-board 5V and 3.3V regulators (for low current internal and external supply) which left the servo and Intel Edison board requirements to be satisfied. This was solved by the inclusion of a switch mode buck converter device (which made use of the XL4015[1]) with voltage and current adjustments and a maximum current supply of 5A. Since the Intel Edison board required it the least 7V, and only one output voltage could be provided to the electrical network, the servos were supplied 1V over their maximum. Investigation into the effects of oversupply on the HXT900 as well as thorough testing at this voltage indicated that the supply was suitable. Thus, the buck converter was adjusted to supply 7V, a step down from the voltage provided by the battery. The boost converter module brought several benefits to the design including, short-circuit protection, current limiting, elementary power filtering and an indication of the presence of power. Therefore, the current supply budget was easily accommodated by the battery chosen and suitable for the buck converter at the worst case scenario of all devices drawing maximum current simultaneously.

[1]XL4015: a 5A, 180KHz, 36V Buck DC to DC converter

The HC-SR04 sensors and pulse to analog converters were supplied from the Intel Edison board supply and the camera via the USB port supply.

Overall Electrical Schematic

Once the electrical sub-systems had been designed and the hardware choices made, an plan of the entire electrical system was drawn up as shown in Figure 3.36 and included power supply and communication interfaces.

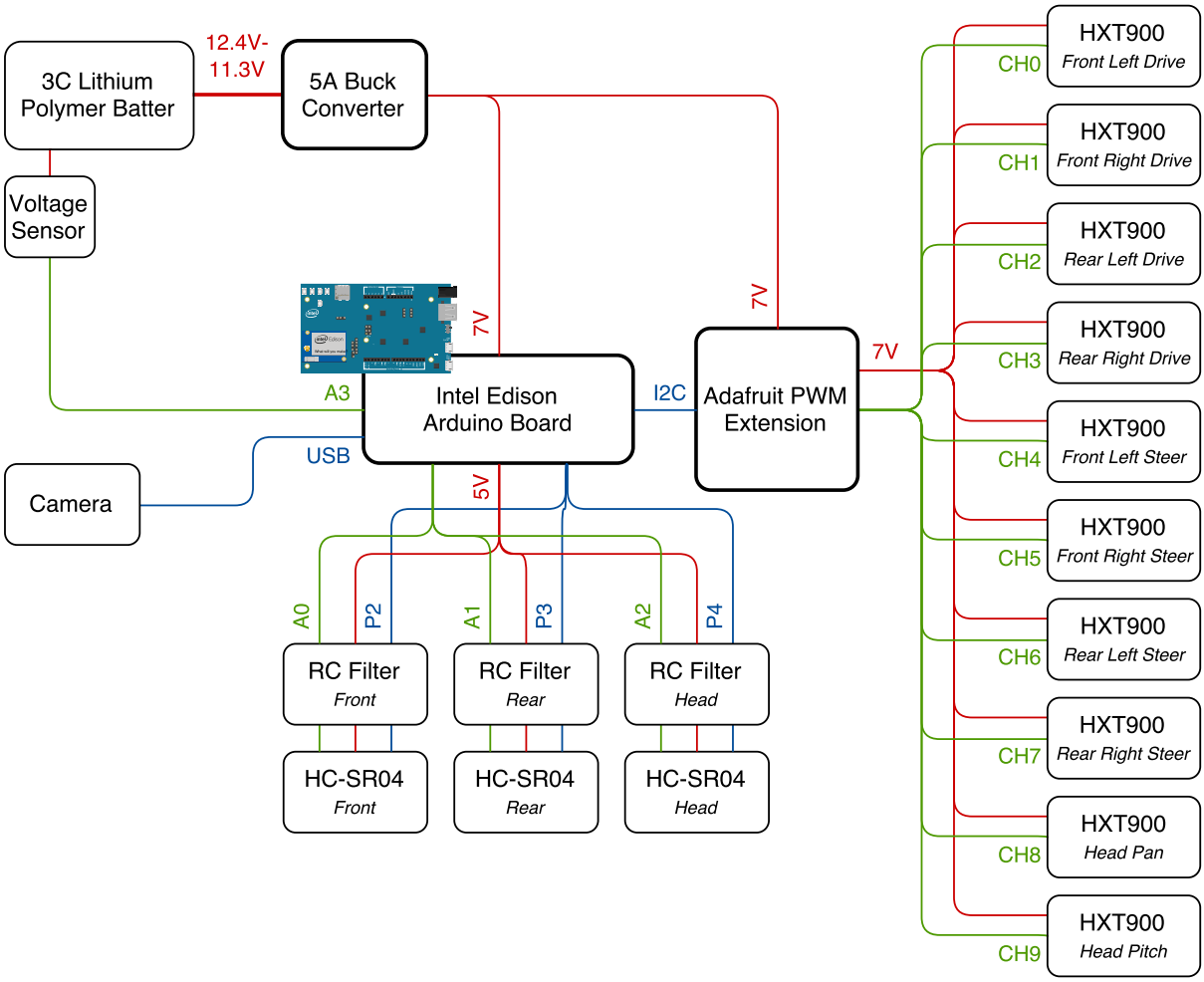


Figure 3.36: Simplified schematic plan of the entire electrical system including power supply and interfaces.

3.4 Software Design

Following the design of the mechanical and electrical aspects of the rover was that of the software as hosted by the RCE as well as for user control and interaction. The section covers the design process undertaken before and during the development of the software for all aspects of the project. The software design and development processes were highly iterative as many of the design choices had to be made after realisation of the technology capabilities and conversely, many of the technology choices were made based on what was required on a more functional and detailed level at design time. The two processes are separated as far as possible in this report for a friendlier structure.

The design stage initiated with an overview of the requirements, descriptions of the technologies that were chosen for development and a plan of the designed structures of each of the subsystems as categorised in the overview.

3.4.1 Systems Overview in Context

An attempt was made to mimic the communications and software structures and patterns used for MSL (and other similar missions) and as such, the software system for this project resembles the three component structure consisting of a system on the rover, the software employed for the collection of relay satellites and ground-based DSN hardware, and the client front-end software as used by the MSL team. Figure 3.1 already highlights the basic breakdown of the software systems. As discussed in Section 3.1.4, the three major software subsystems by name included the Rover Compute Element (RCE) and the Robot Sequencing and Visualisation Program (RSVP) divided further into the Server and the Client.

The RCE was hosted by the computational hardware on the rover (the Intel Edison board, here-onwards referred to as the RCE board) and required control of hardware, sensory and digital input and communication with the RSVP Server in order to fulfil the end requirements. This imposed the need upon the RCE to be able to read from and control the hardware inputs and outputs on the RCE board. For communication, the RCE was to make use of the on-chip WiFi module to send and receive data which would make up the commands, telemetry and video stream.

The RSVP Server was a standalone server process serving as the middle-man between the RSVP Client and the RCE with respect to telemetry, control and video data. The Server established connections with the RCE for data message and video stream type communications and was required to act as a broadcast node to allow for multiple RSVP Client instances to receive the data messages and the video stream. Details on the network topologies, specifically those of the video broadcasting component, are discussed in the structure plans to follow. The RSVP server also hosted the RSVP Client web application to be served and coordinated control access by the active clients.

The RSVP Client as served by the RSVP Server was a web application which was required to facilitate user input for control of the rover, display the video feed from the RCE

as broadcast by the Server as well as provide telemetry received via data messages. As described further in the following design sections, the RSVP Client consisted of a front-end component visible in the browser or any web-view as well as an underlying system managing the flow of data through the application and handling the communication between it and the RSVP Server. State and user input data was to be sent to and received by the Server and the video data received to be displayed in the video element within the interface.

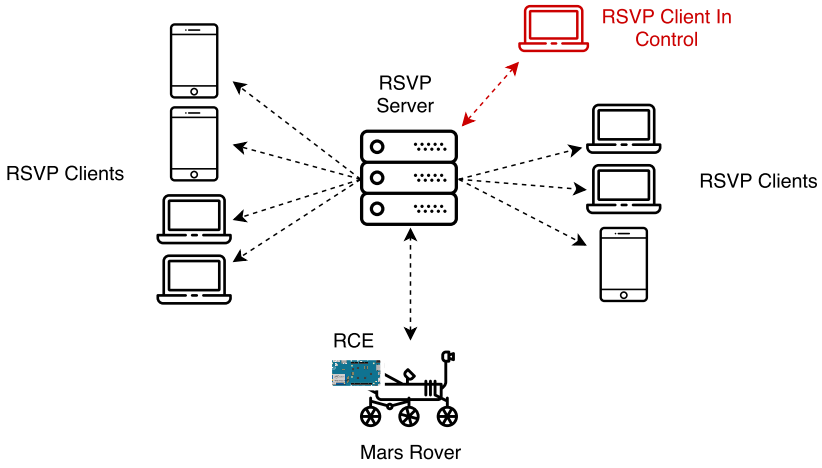


Figure 3.37: Diagramatic depiction of a typical use scenario of the software system.

An overview of the entire software system on a very high level is shown in Figure 3.37, a typical scenario involving the operational rover, the RSVP server in communication with the RCE and multiple RSVP Clients connected to the Server. The intention was to have the system be capable of allowing multiple devices on the network to connect to and interact with the system and one of the devices to have rover control access. The RSVP Server would manage the connected Clients and decide which of them would be able to control the rover. The principle behind the choice of which Client becomes the controlling Client is discussed later on.

3.4.2 Plan of Structure

In order to effectively coordinate the design and development of the software system as a whole, structural plans of the system and the subsystems within were constructed. The plans also aided the realisation of requirements and, further, choices of software technologies to use for each of the subsystems and components. The high level structure of the three subsystems and their interconnectedness was designed before constructing more detailed perspectives of each of them.

System Architectural Structure

Figure 3.38 is a high-level diagram showing a single-client scenario involving the three primary software subsystems and the means by which they were to communicate and interact. The solid arrows indicate wired connections and these exist on the RCE board for, as on the left, hardware communication and interfacing with sensory devices such

as the camera. The RCE uses the wireless module to transmit video data and message data to the Server via the wireless module on the RCE board. The RSVP Server then offers the application source for the RSVP Client to be requested and loaded. Control, telemetry and video data is then sent and received between the RSVP Client and Server.

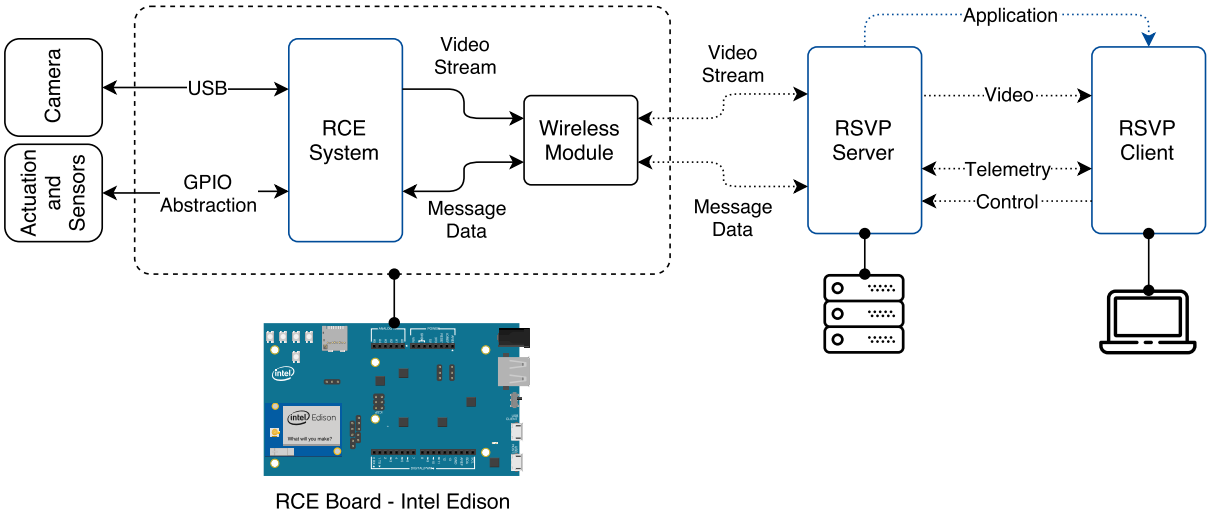


Figure 3.38: Diagram outlining the basic software system component structure.

As mentioned previously, the RSVP Server's primary role was to relay data between the RSVP Client and the RCE. Employing this structure for the flow of data allowed for many RSVP Client instances to exist and interface with the system. Not shown in Figure 3.38 is the case where there are N RSVP Clients on the network whereby one client would be in control of the rover, using the control, telemetry and video communication channels, and $N - 1$ Clients would be receiving telemetry and video data only. Throughout the software design process, as much of the computational burden anticipated to exist in the system was offloaded onto the RSVP Server simply due to the fact that the RCE had the most severe computational performance limitations. The RSVP Server could be scaled to provide any required performance and thus could absorb functional components that were computationally taxing, an example of which is handling a large number of client connections that all require video and telemetry data. Communication specifics between all of the subsystems in Figure 3.38 will be discussed in the design sections that follow as well as in the development sections.

It can be seen in Figure 3.38 that data transmission between all components of the system are grouped into two main channels: video data and message data. The separation of the two is made clearer in Section 3.4.4 which discusses the broadcast topology intended for use for the video data and how the data and the characteristics of the required communication naturally differ from that of telemetry and control data. Splitting of the two flows of data also allowed for finer control over the transmission in terms of setup sequences and during operation. As such, the message data channel between the Server and a Client was split on the basis of function.

3.4.3 Technology Choices

Common Platform Flavour

The fact that the RCE board's computer supported the use of Linux as an operating system presented an opportunity to have all aspects of the software system follow suit in this regard, specifically the RSVP Server. The Server was intended to be flexible in form-factor to facilitate the open-sourcing secondary objective of the project and for the purpose of the system developed for this report, it was chosen to be a PC capable of running a distribution of Linux. Most distributions of Linux offer the stability and performance required by server-type processes and applications which explains the popularity of the operating system in network and internet service industries [53] and hence the decision to employ such as the operating system for the Server. The choice between heterogeneous and homogeneous architectures for the software system resolved to a trade-off between the advantage of technological flexibility as in the heterogeneous case and development simplicity in the other. Technological flexibility would have benefited the design in allowing choice of technologies that would better suit each of the subject components, perhaps to optimise resource consumption (storage, computation or even power) or improve compatibility with associated hardware and other components. Due to the limited time-frame of the project as well as there being no specification related to such optimisations, the benefit of the ease of learning and design brought by a homogeneous system weighed in greater than flexibility, together with the abundance of packages and tools available for a Linux-based platform. Using Linux also supported the free, open-source software ideology making it more readily available to those who intend to be involved.

JavaScript

With both the RCE and RSVP Server being based on a Linux platform as well as the RSVP Client being web-based (discussed further in Section 3.4.3), another opportunity to employ a common technology across the system was presented, specifically the use of the JavaScript language most commonly associated with websites and web applications. Using JavaScript across the stack (including the RCE) followed the architecture pattern of homogeneity, simplified the learning process before and during development of the subsystems and minimised anticipated development overhead in terms of software development environments and build tools. Using JavaScript kept the project on a modern, popular and cutting-edge trajectory, a choice backed by it being a widely recognised full-stack solution used for services such as Netflix, Paypal, Medium, Uber, Twitter and Airbnb [54] [55]. JavaScript's place in an embedded context was found to be increasingly fitting, especially in scenarios where an internet-connected stack could benefit from the seamlessness of technologies as a result. Given that the Intel Edison was a device which comfortably bridged between a embedded environment dedicated to hardware alone and a connected computer with no direct hardware-interface relevance at all, JavaScript provided a good balance between the two areas of software and kept available interoperability with software and tools that could cover the extremes if required. Other benefits include the use of JSON as a well recognised data format, untyped syntax and the semantic depth provided by the object-oriented nature of the language.

JavaScript itself was not the platform choice in entirety and due to the fact that it is an interpreted language, the project required a choice of runtime environment that would be able to be used on both the RCE and the RSVP server. It was found that the most popular server-side runtime for JavaScript projects was Node.js, an asynchronous, event driven, single thread environment that utilises the V8 JavaScript Engine. It was identified that the structure of the stack of this project resembled a typical IoT stack (involving a front-end, back-end and connected devices) and Node.js was the most popular environment for applications that required real-time data synchronisation and connectivity between devices with a growth in usage unlike any other JavaScript runtime or framework [56].

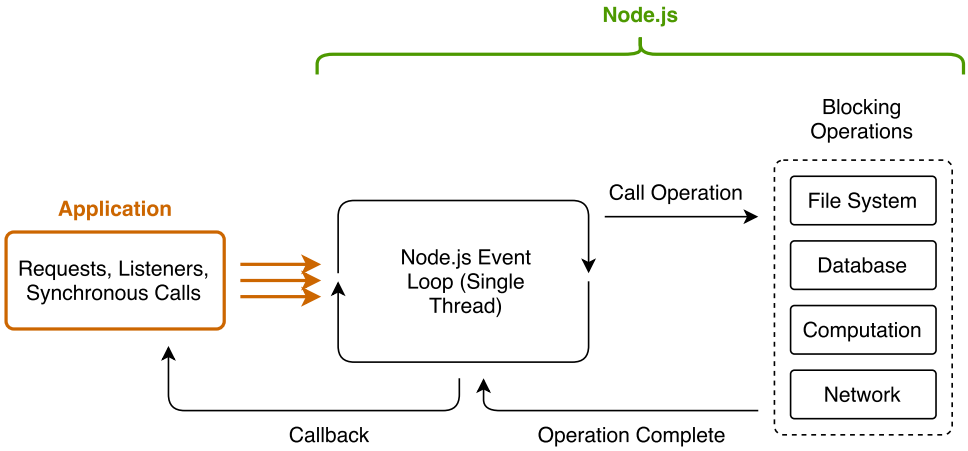


Figure 3.39: Diagram showing the Node.js architecture's event-loop execution design (adapted from [57] and [58]).

A key advantage of the way in which Node.js runs is in the event loop architecture that it utilises, as demonstrated in Figure 3.39. Since Node.js is single threaded, in order to bring asynchronism to an application, at least one point of the software architecture should be sequential in execution, taken care of in the Node.js case with its event loop. The significance of this method of execution is in the advantages that it brings when dealing with operations that block the execution flow. In the context of server applications where requests are fully asynchronous, any operation that blocks handling of such requests will result in poor response performance and a poor end user experience. System operations are normally always blocking in operation and thus Node.js utilises a system library `libuv` to offload the blocking calls to system threads, the important point here being that the event loop need not wait for the operation to complete. This means that the left hand side of the flow of execution in Figure 3.39 (application requests) is scalable with minimised performance cost, allowing many end users to be connected to and request data from the server at any given time.

Node.js brought with it open-source resources, tools and software modules which greatly improved the development time and opened up what was possible from the perspective of both the RCE and the RSVP Server (and Client, but to a lesser extent). Many of the open-source projects used throughout the software system were chosen during development based on demand and are mentioned in-text in the design and development sections as well as in Appendix A.

A project worthy of mentioning at this point is one that was in development at the time of writing by NASA, namely Open MCT[1], a web-based mission control framework utilising Node.js as the server runtime (although the architecture design was intended to be server-agnostic). Open MCT was a great example of how an across-the-stack JavaScript implementation is well suited to a data and communication critical application. Another example of Node.js's place in the aerospace industry was the use of it by United Technologies Corporation Aerospace Systems (UTCAS) to develop a data management system for the lifecycle of EVA spacesuits, a response to a problem identified in the analysis of data which was before hosted across separate and legacy databases [59].

ECMAScript 6

Announced on the 6th of June 2015, the Sixth Edition of JavaScript, also known as ECMAScript 6 (ES6), introduced a significant range of new features to the language many of which are syntactically unique. It was decided that the project make use of the new edition in order to remain in-sync with modern modules and technologies at the time. In fact, a few features from the Seventh Edition (ES7, released on the 7th of June 2016) were included as well, both additions to the original ES5 and ES5.1 resulting in an added complexity in the build process. The changes to the build process are dealt with in the development section.

Embedded Software Platform

The RCE board's Intel Edison computer ran a default operating system: a custom version of Linux from the Yocto Linux Project maintained and compiled by Intel. The open-source Yocto Project provides the tools and resources to build Linux distributions specifically for embedded targets. Intel's distribution was a modified version of "Yocto Poky" and could be flashed onto the Intel Edison using the tools provided. After little experimentation with the distribution, it was found that many of the niceties of a Debian-based operating system were not present, the main issue being the lack of the package manager (`apt-get`) and its associated online repositories which would aid the development process. For this reason, a different distribution was chosen for the RCE, "Ubilinux", a customised version of "Debian Wheezy" which was maintained by Emutex. It was decided that due to mid-project termination of support for Ubilinux from Emutex, Ubilinux would remain the distribution used during development of the rover after which it would be replaced with the latest version of Yocto Poky if time permitted. This would provide an opportunity to build into the distribution the tools confirmed as being required for the operation of the RCE.

Since it is primarily a server runtime, Node.js itself did not have native support for embedded hardware control such as that required for the RCE. It was important to first understand how the hardware was interfaced with from the perspective of the Linux user space[2] on the Intel Edison. One of the base-level principles around which Linux was developed is the concept of the hierarchical file-system. Most importantly,

[1] Open MCT: `https://nasa.github.io/openmct/`

[2] Linux user space: memory area and separation of execution reserved for application code and some hardware drivers, which sits separate from the kernel space reserved for low level code and drivers

the file-system need not necessarily consist of files that represent viewable and editable data in the user-owned sense (documents, pictures and other media, for example) but also files that hold system data which have been allowed to propagate from the kernel space to the user space. The system files may represent various hardware components and subsystems, holding state information and allowing control of the hardware through editing of the associated files. The significance of this concept is that the Intel Edison exposes hardware pin access using a virtual file system from the kernel upwards into the user space using `sysfs`. When the `sysfs` file system is mounted, files organised and structured to represent the GPIOs and other hardware peripherals appear in the user space file hierarchy and if the user has the required permissions, it can read from and write to the files from software. A diagram showing the components of this type of hardware interface on the Intel Edison is shown in Figure 3.40.

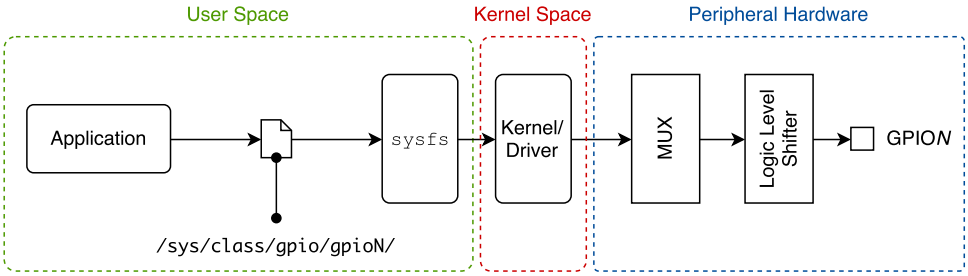


Figure 3.40: Simplified diagram showing the interface between peripheral hardware and code running in the Linux user space (adapted from [60] and [61]).

Atop this method of hardware interface was another abstraction that was available: a C/C++ library, called `mraa`, which provided common `sysfs` file operations in code allowing for better semantics amongst the large amount of multiplexing required on the Intel Edison board. `mraa` also provided JavaScript bindings so that the library could be easily utilised in the RCE software subsystem running in the Node.js environment. This was not the end of the stack, however, as multiple Node.js modules were available that resided atop the JavaScript abstraction of `mraa`. Up until this point, little variation in the platform layers was available and this final choice of module formed the primary design choice for the RCE platform. Two robotics framework modules were chosen as candidates based on popularity and compatibility with the Intel Edison (and the Arduino Breakout Expansion), namely "Johnny-five" and "Cylon.js". Both modules leveraged the object-oriented nature of JavaScript to provide easy interface with the hardware on the RCE board as well as provide the ability to remotely program and command the device on the same network. The architectures of each of the modules were compared and it was found that Cylon.js emphasised the remote scripted topology compared to code that would run local to the device to be controlled. The intention was to have the RCE be independent in operation from any other device (i.e. not be strictly reliant on another device to be online) much like the autonomy of *Curiosity* and therefore Johnny-five (hereafter referred to as J5) was chosen as the framework to use. An example snippet of how one would use J5 is shown in Snippet 3.1.

```
1 import * as five from 'johnny-five',
2
3 // Create the board instance
4 board = new five.Board();
```

```
5
6  // Setup listener to wait for board to become ready
7  board.on('ready', () => {
8    // Create an Led on pin 13
9    var led = new five.Led(13);
10
11   // Strobe the pin on/off, defaults to 100ms phases
12   led.strobe();
13 });
```

Snippet 3.1: Example initialisation of a device and an LED in J5.

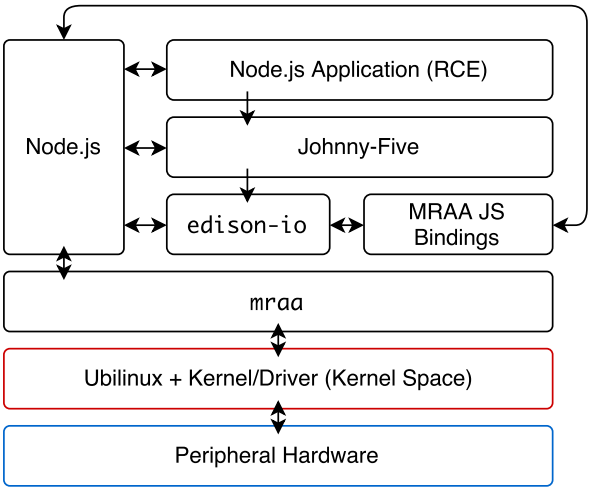


Figure 3.41: Diagram of the hardware stack designed for the RCE.

The resulting RCE hardware stack is shown in Figure 3.41 which allowed hardware exposure to the RCE software for all the specified requirements, except the camera, which was handled in a lower-level manner discussed in the development section. A further addition to the stack, `edison-io`, is shown in the diagram, which is responsible for dealing with hardware platform specifics in an attempt to keep Johnny-Five and its class and driver implementations as consistent and standard as possible. The `edison-io` project is a direct wrapper of `galileo-io`, a board support module compatible with the Intel Edison.

The Web as an Application Front-end

Up until this point, the RSVP Client has been said to be a web application without proper reasoning. The decision was made at the start of the project not without consideration of other front-end platforms. Whilst not explicitly stated in the project specifications, the intention behind the RSVP Client was for it to be a highly accessible application that need not be restricted to within the exhibition space. This follows the theme of the project being targeted at educational environments and general outreach where an application that can be accessed from multiple different devices in multiple locations would prove to be a valuable feature.

The two main categories exist with respect to base-level technologies for the front-end, native applications and web applications, where native refers to applications that are dedicated to the subject device or platform. Investigations made into the popularity of the two application types [62][63] suggested that at the time of decision, native applications were more commonly used but most likely for specific and repeated tasks or where CPU-intensive computations are required. What was also found, albeit a trivial point, was that the development time, complexity of projects and increased maintenance and skills is required for applications that are to be natively developed but also available across many different types of devices. The web, however, could be considered a universal platform which attempts to bring platform-agnosticism to web content through browsers. The browser can be seen as a technological buffer between the page content and the hardware on which the browser is running.

The project aimed to have the RSVP Client be an application that could be run on as many devices as possible, varying in operating systems, form factors and screen sizes. This objective coupled with the very short development time-frame gave the decision to make the Client a web-based application satisfactory justification. It was also deemed reasonable to make use of the already existing requirement for network-based data communications for the application itself.

Front-end Framework

The RSVP Client could have been developed from first principles or by making use of an already developed front-end JavaScript web-application framework, of which there were an abundance. Many of the frameworks available were created with both user-interface (UI) components and application mechanics in mind. For brevity, a comparison of the candidate frameworks is not shown, however the ones that were considered include AngularJS[1], ReactJS[2] and Polymer[3]. The chosen framework was Polymer for the following reasons:

- **Web-components:** Polymer makes heavy use of a particular area of the modern day web standard, the features relating to web components. Web-components allow the encapsulation of functions, features and UI artefacts to result in a component that is re-usable and maintenance-sane. This allows the developer to bring semantic structure to the application in a way which promotes understanding for people who wish to contribute to the project, for example.
- **Shadow-DOM:** Included in the principle of web-components is the Shadow-DOM feature, a mechanism of HTML encapsulation which includes style and associated methods and properties.
- **Declarative Data:** Polymer comes with a rich data management mechanism

[1]AngularJS: a framework which allows the extension of plain HTML with in-browser JavaScript dependency injection - `https://github.com/angular/angular.js`

[2]ReactJS: a Facebook developed declarative JavaScript library with a distinct DOM creation and management flow - `https://github.com/facebook/react`

[3]Polymer: a Google developed library come ecosystem which allows the creation of reusable HTML elements with style and functional encapsulation with emphasis on making close use of web standards - `https://github.com/Polymer/polymer`

allowing for the declarative construction of data flow through the application, an area of application design that can get complex and difficult to scale.

- **Polymer Elements Catalog:** Google have provided a large catalog of ready-made Polymer elements which are commonly required UI and application components for easy development. In addition to the Polymer Catalog are elements that have been developed by third parties and made available for use in an open-source manner.

The Polymer Project had also made indications towards making the framework ES6 compliant meaning the framework would remain a sustainable choice for the RSVP Client application.

Message Data Communication

Message data communication was required between the RCE and RSVP Server as well as between the RSVP Server and the RSVP Client. This data would be a structured payload which in JavaScript amounts to a JSON (JavaScript Object Notation) object, an example of which is shown in Snippet 3.2. The JSON object can be serialised to a string which, in fact, becomes irrelevant from an implementation perspective in the context of the chosen technology. This could have been achieved by leveraging the AJAX principle whereby HTTP `POST`s and `GET`s are made from a client endpoint to a server endpoint. In the case of an HTTP `POST`, the receiving endpoint may accept the transaction and do with the data what it requires. A `GET` required the receiving endpoint to send a response back, hence giving AJAX bidirectional capabilities. A big driver behind the adoption of AJAX was the fact that web pages did not have to refresh the browser window to make such a request and thus various parts of the page could update in a manner more similar to an application. This is referred to as a Single Page Application (SPA).

```
{
  name: 'Message Name',
  type: 'data',
  payload: {
    key1: 'data1',
    key2: ['data2', 'data3'],
  },
}
```

Snippet 3.2: An example of a structured data message.

However, data was required to be pushed from both endpoints, regardless of whether or not the endpoint was a "server" and AJAX did not provide such connection persistence. A technique called long-polling[1] was a popular progression from AJAX, however, a more efficient method was available. A new communication protocol was introduced in 2008 called WebSocket which combined a small portion of the HTTP protocol, specifically the initial handshake, with the use of a TCP connection to provide full duplex communication

[1]Long-polling: a method by which an HTTP connection is kept open until data is ready to be pushed as a response

with no regressions in security [64]. To open a WebSocket connection, an ordinary HTTP `GET` request is sent to a server which is ready to receive WebSocket requests, with an additional header `Upgrade: WebSocket`. The server then responds to the request and a TCP connection is set up between the two endpoint along which asynchronous, size-unrestricted messages can be sent in both directions without the overhead that HTTP requests incur. As such, this protocol was chosen as frequent and variable sized data messages were required to be sent for telemetry and control purposes.

The WebSocket API was available to use to implement such connections, however, a higher level Node.js module was also available, a widely used abstraction called Socket.io[1]. the benefits of using Socket.io included its simplified API which was kept flush with ES6 as well as the fact that it provided cross-browser fall-backs, offloading the responsibility of ensuring compatibility with different browser vendors onto the Socket.io project. The event-driven nature of the module provided the ability to react to the asynchronous pushing of data as well as event such as new connections and changes in connection status, allowing better control of the communications.

Media Streaming

One of the key requirements for the project was the streaming of a video feed from the rover to the connected Clients over the wireless connection. As discussed in the conceptual development sections, a USB compatible webcam was chosen to capture video from the rover head and provide this data to the RCE board and hence the Linux operating system. It was at this point that a method of streaming the video to the connected Clients was required and the process involved investigating a combination of broadcast topologies and tools that made the various topologies possible.

With the presence of the Server, already responsible for broadcasting telemetry data and relaying control data, video stream broadcasting to the multiple Clients was delegated to it as part of the attempt to offload as much of the computationally and resource intensive operations onto it as possible. This choice of broadcasting meant that, by some means, the RCE would have to send the video data obtained from the webcam to the server over the wireless network connection.

To initiate the process of choosing streaming technologies, research was conducted into popular and effective software tools for streaming video data, specifically for the chosen platform for the RCE and the RSVP Server, Node.js. A common design pattern among many IoT and modern embedded software developers was to use the same technology employed for the transmission of telemetry and control data, Socket.io. Since WebSocket, and hence Socket.io, uses a TCP connection, the size of the transmission data was not a limitation and this made sending large collections of video data possible. The RCE would send the video data frame-by-frame, as separate Socket.io messages, to the RSVP Server, which would simply relay the frames to each of the connected RSVP Clients in the same manner and the frames would then be played back to the user in the application front-end. However, the RCE would still require the video frames to be available as

[1] Socket.io - `https://github.com/socketio/socket.io`

images in a supported format, and due to the fact that the webcam was chosen to be UVC-compatible (i.e. not requiring drivers for operation and control), multiple Node.js modules were available to obtain the video data from it without the need for proprietary software. Some of the modules included `v4l2camera`[1], `linuxcam`[2] and `uvc-control`[3]. Figure 3.42 shows this method of video broadcast in a server to single client example (the example can be extended to having multiple RSVP Client instances all receiving frames from the server with no change).

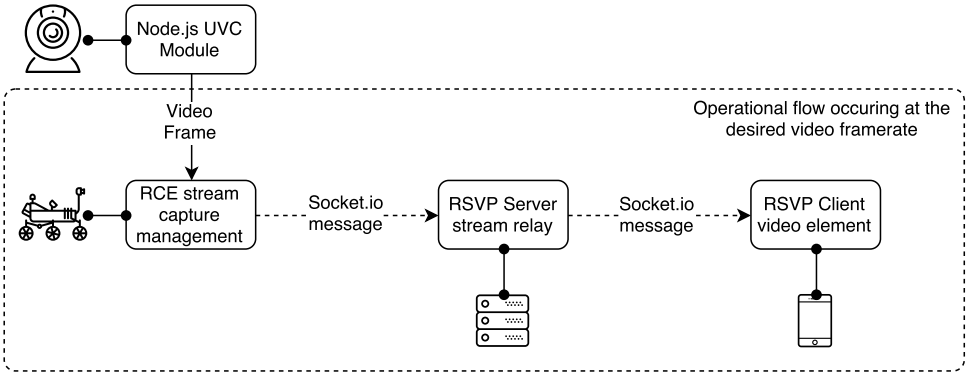


Figure 3.42: Diagram of the typical flow of data using Socket.io to stream a video feed.

Further research revealed a second streaming candidate, WebRTC, a technical specification and an open-source project comprising of APIs that can be used to implement Real Time Communications (RTC) on mobile devices and in the browser. Web-based services such as Skype, Discord, Google Hangouts and more use WebRTC for the multimedia streaming components of the applications since it provides a range of features designed a developed towards ensuring a stable communication experience despite a potentially intermittent and unreliable network connection. An important aspect of WebRTC was the fact that the API is a part of the HTML5 specification meaning cross-browser support is driven by competition of compatibility among browser vendors, a valuable dynamic for accessibility for the project. Having said that, WebRTC have provided yet another module, `webrtc-adapter`[4] to ensure compatibility of applications in the context of a highly volatile platform environment.

In the case of the WebSocket connection technique, it is apparent that there is little to no dynamically optimised transmission of video data at any point in the transport layer meaning the video feed would have been susceptible to poor network performance resulting in severe latency and potential memory issues due to bottlenecks at the sending nodes (RCE and RSVP Server). The desired robustness would have to be developed around the use of Socket.io, a task requiring extra research and development. On the other hand, the WebRTC communications architecture, which can be seen in Figure 3.43, included a video engine as part of a rich audio-visual enhancement layer which aimed to, among other objectives, protect the stream from transmission issues. As such, WebRTC was chosen as the streaming technique and the benefits that it provided to the project

[1] `v4l2camera` - https://github.com/bellbind/node-v4l2camera

[2] `linuxcam` - https://github.com/Qualphey/node-linuxcam

[3] `uvc-control`: only capable of utilising UVC commands, cannot actually capture video data - https://www.npmjs.com/package/uvc-control

[4] `webrtc-adapter` - https://github.com/webrtc/adapter/

will become apparent in the description below and further in the design sections.

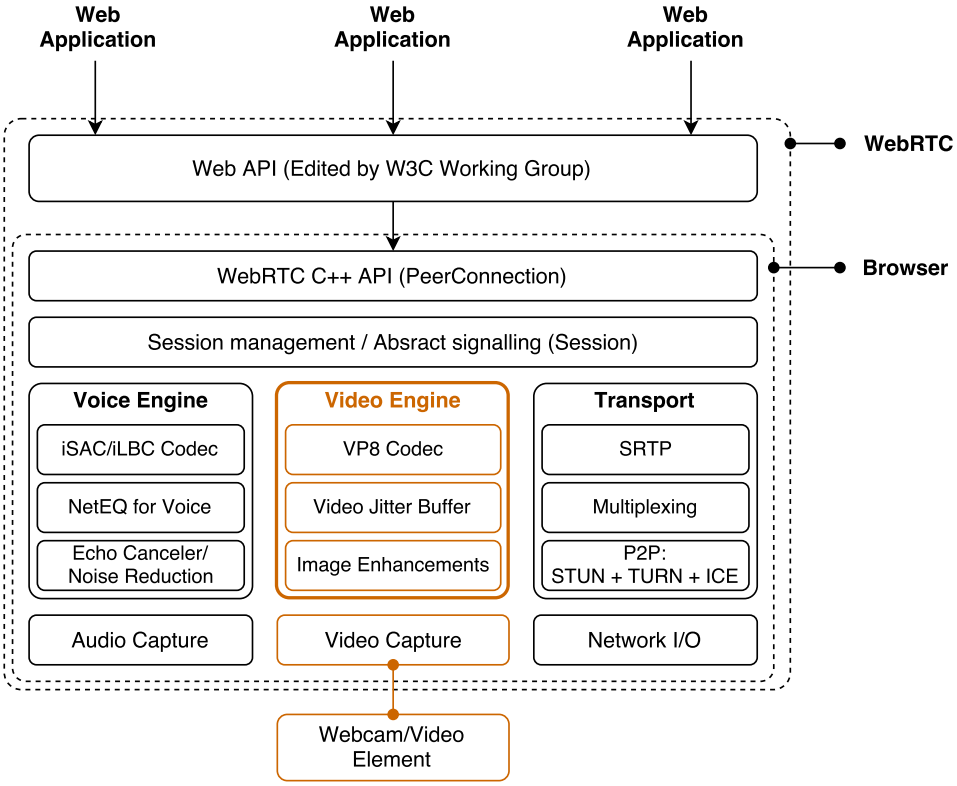
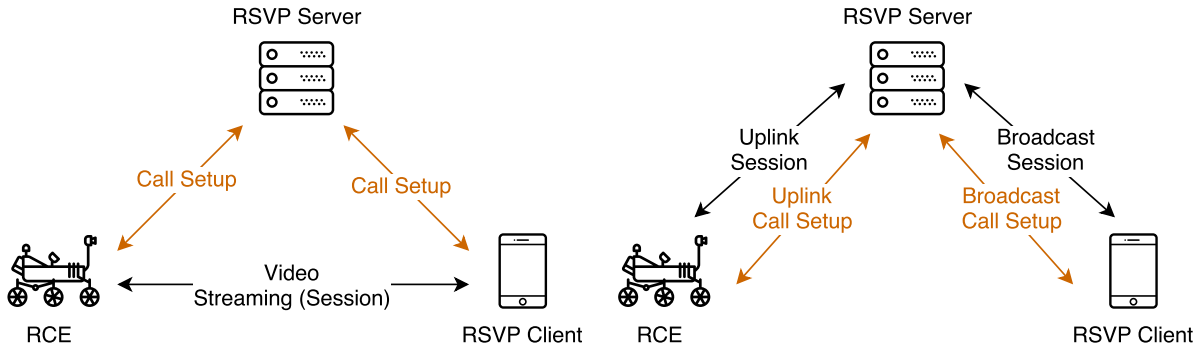


Figure 3.43: Diagram of the WebRTC architecture highlighting the video component of the streaming WebRTC provides (adapted from [65]).

As shown in Figure 3.43, WebRTC caters for both video and audio streaming, however, only the video capabilities apply to this project. In fact, WebRTC provides arbitrary data streaming by means of an `RTCDataChannel` as well and this feature was considered for the control and telemetry message data transmission. The choice remained with WebSockets due to the simplicity of the Socket.io API and the benefits that it brought considering its suitability for the associated data type and structure. The principle of operation behind WebRTC is primarily defined by the notion of the manifestation of a session between two peers by means of the `RTCPeerConnection` component. The session can be referred to as a call, and the call handshake process involves the two peers that intend to connect and a third element, a signalling server, which facilitates the initial inter-peer communication before the session is live. A more thorough description of the negotiation flow involved in using WebRTC is given in Section 3.4. The important point here, however, is the fact that the session is peer-to-peer, meaning video streaming from the rover to the RSVP Client need not involve an intermediate server component at all, apart from the signalling required at session initiation. Whilst this was appealing from the perspective of a simpler network configuration as well as a simpler server design, the project required one-to-many communication since the video streaming was a broadcast as opposed to a two-endpoint call. Having the N number of connected RSVP Clients imposing the responsibility of broadcasting the video onto the RCE would not have been suitable given the resource and computational limitations of the Intel Edison. A preferred configuration was to have the RCE transmit the video data to the RSVP Server and have the Server be responsible for the broadcast of the data. The two configurations are contrasted in Figure 3.44.



(a) Without intermediate server for video streaming (strictly peer-to-peer).

(b) With intermediate server for video streaming.

Figure 3.44: Simplified diagrams of typical WebRTC connection configurations.

As with the Socket.io abstraction for WebSockets, multiple services and modules were available for the implementation of a WebRTC communication system, all of which were free and consumed much of the work that would have been required for development consisting primarily of boilerplate software. The projects/modules considered included EasyRTC[1], SimpleWebRTC[2], Kurento[3] and PeerJS[4]. PeerJS was not suitable for the streaming-type communication required for the project and SimpleWebRTC and EasyRTC did not provide the richer media server capabilities that Kurento offered. Therefore, Kurento was chosen for the media server it included as well as the fact that it provided server-side APIs developed for Node.js together with client APIs. The Kurento media server was designed to be a common node in a broadcast network and thus suited well the use case of the project.

Detailed design of the system with the implementation of Kurento's media server will be covered in the design and development sections to follow.

3.4.4 Subsystem Design

With the primary technology choices made, all three of the software system components were designed in a functional sense, removed as far as possible from implementation and code detail. For brevity, the design (and development) descriptions that follow are written in "as-built" style with little reference to many of the design iterations and variety of design choices made along the way. Multiple solutions to a particular aspect or component design problem are discussed where significant.

[1]EasyRTC: A comprehensive JavaScript library consisting of client and server APIs developed for multiple browsers, Node.js and signalling via Socket.io - `https://github.com/priologic/easyrtc`

[2]SimpleWebRTC: A simplified abstraction of the core WebRTC components and features - `https://github.com/andyet/SimpleWebRTC`

[3]Kurento: A WebRTC media server and client APIs offering full media broadcast in multiple configurations - `http://www.kurento.org/`

[4]PeerJS: A browser-based WebRTC wrapper providing configurable peer-to-peer connections for arbitrary data transfer - `http://peerjs.com/`

RCE Design

Plan of Structure

The RCE was a multi-functional, multi-faceted software component bridging the largest gap between hardware control and communications compared to the other two components in the project. It was broken down on the basis of function and each of the subcomponents designed and developed. Multiple main functional areas were identified to be:

- **Control:** Interpretation of incoming control signals and commands, translation of such commands into hardware signals against time and control of RCE board outputs as part of a hardware-to-software abstraction layer. The control subcomponent included initialisation of hardware.
- **Sensing and Telemetry:** Interpretation of incoming hardware signals from sensors and translation of such signals into telemetry messages to be sent to the RSVP Server, including retrieving and packaging of system and hardware state telemetry data.
- **Communication:** A transport layer providing a means of communication with the RSVP Server for telemetry, control and video data transmission.
- **System Operation:** Initialisation of the RCE system as a whole including the execution of predefined system sequences.

Figure 3.45, an overall block diagram of the designed RCE structure, shows briefly the presence of the identified functional areas and how they interrelate. The RCE was ultimately governed by the Node.js application entry point and system sequences (not to be confused with RSVP control sequences). System sequences could be executed at any point during the RCE lifetime and were responsible for coordination of initialisations of hardware and software components as well as performing routines during operation. Routines included system vitality checks, routine maintenance or emergency shutdown procedures. The system sequencing mechanism allowed sequences to be executed synchronously or be driven by events (asynchronous) and allowed for pluggable and scalable sets of sequences with very little resultant impact on the general operation of the RCE. Importantly, sequences at this level were designed to only be responsible for system functions and operations and not any aspect of hardware control.

The communication transport layer provided the facilities for communication with the RSVP Server. Attached to the RCE's network I/O were three server components, an HTTP server for pre-WebSocket communication and simple requests, the WebSocket server, named the "RCEIO", and a video stream endpoint which was made separate due to the choice of streaming technology. Interface with the hardware components as shown in Figure 3.45, the camera, servos, proximity sensors and voltage input, was taken care of by the control layer, a complex subcomponent involving decoding and translation from incoming commands and signals from the RCEIO translator, a control loop acting as a filter of driving input signals from the translator and finally a hardware abstraction which partially incorporated the Johnny-Five library module.

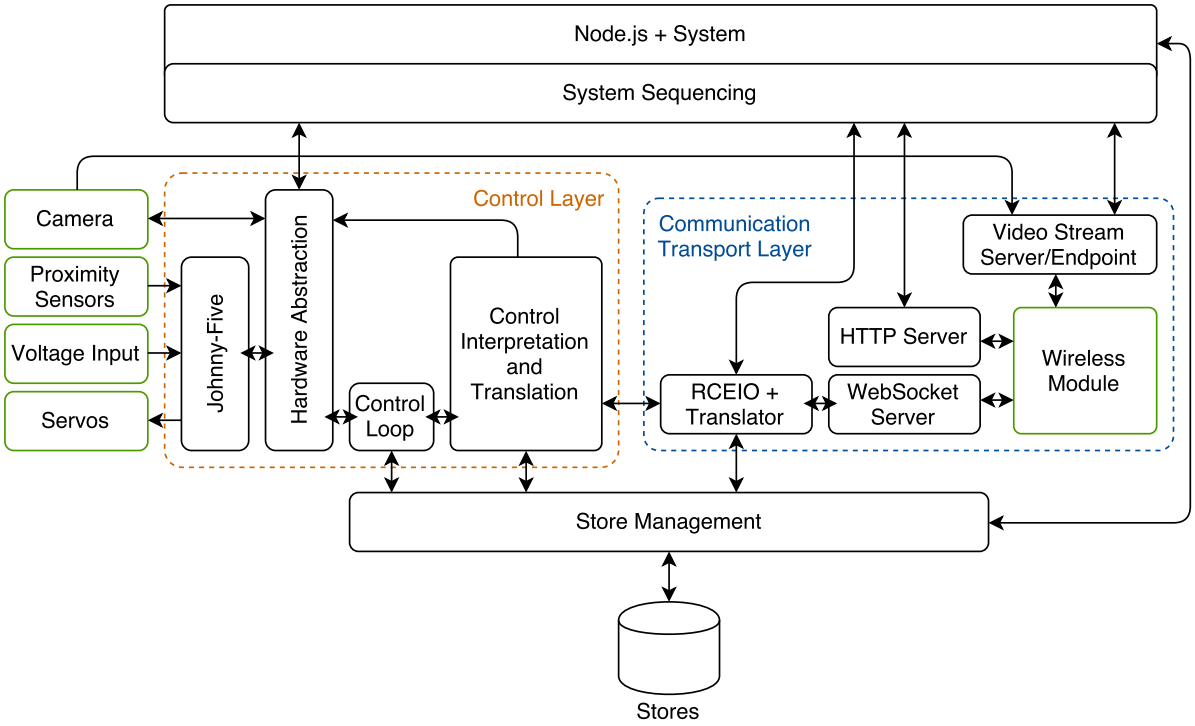


Figure 3.45: Block diagram of the designed software structure for the RCE.

Most of the above subcomponents were exposed to the data store facility as a means of communicating cross-block and cross-module as well as providing data for telemetry purposes. The reasoning behind the data store as well as its functional design are discussed as part of Section 3.4.4.

System Sequencing

The pluggable sequencing module is better depicted in Figure 3.46. Multiple sequences were designed and written to be included in the sequence library and could be called at any point during execution. Sequence operations could have been blocking in execution thus ensuring a deterministic flow of operations or asynchronous if event-driven logic and state-dependency was required. The sequence was exposed to any operation or data that was made importable in JavaScript terms throughout the project.

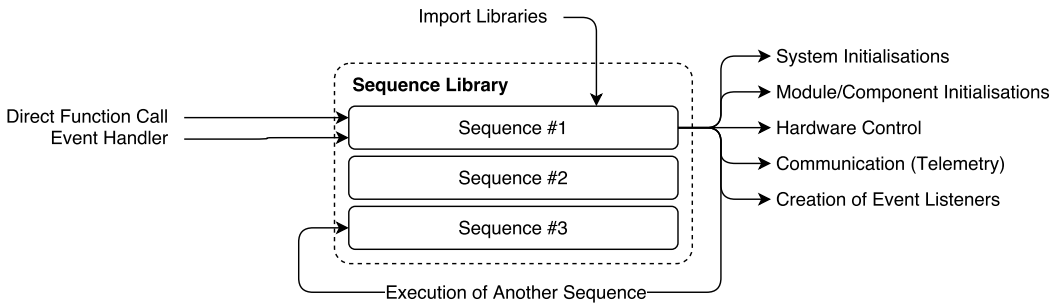


Figure 3.46: Diagram showing the functional design of the RCE's sequencing module.

An example sequence that was designed was the startup sequence, the very first set of operations that are executed when the RCE process is invoked on power up of the

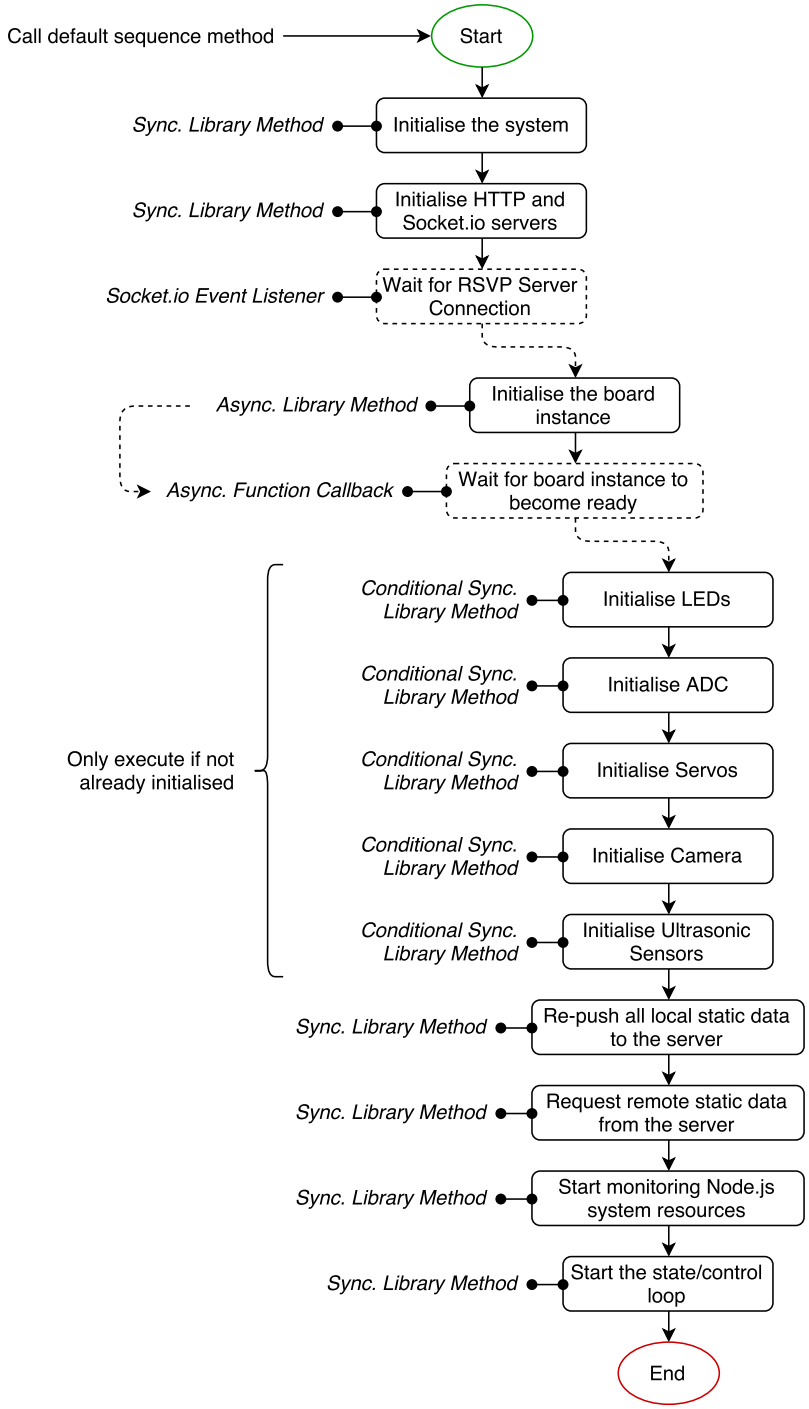


Figure 3.47: Sequential flow diagram of the designed startup sequence.

RCE board (and the rover itself). Figure 3.47 shows the execution flow of the sequence, which began with the initialisation of core system features including instantiation of Linux process monitors and registering the shutdown sequence to process events that indicated a termination of the RCE. The HTTP and RCEIO servers were then created and started after which the sequence waited for a successful incoming connection, from the RSVP Server, by means of the `connected` event emitted by the Socket.io instance itself. Upon successful connection, the Johnny-Five board instance was created, an asynchronous operation whereby the sequence resumed in the associated callback method. The rest of the sequence was strictly sequential, including initialisation of all the hardware

subcomponents (with prevention against duplicating initialisations/instances), synchronisation of data stores between the RCE's store and the RSVP Server and starting the state driver loop as part of the control aspect of the RCE.

For the purpose of this project, the following sequences were designed and implemented:

- **Startup Sequence:** As described above, the startup sequence initialised the system and hardware modules upon power up of the rover.
- **Power Down Sequence:** The power down sequence was triggered upon the termination of the Node.js RCE process which may have been brought about intentionally or in error. Regardless, the sequence ensures that hardware systems are correctly shut down and that communications with the RSVP Server are closed gracefully.
- **Self-diagnostics Sequence:** The self-diagnostics sequence was designed to be a proof of concept for the level of automation such a rover might have and included testing of all the hardware and software components, reporting back to the RSVP Client the results of the tests. It remained a proof of concept due to the lack of sensory feedback present in the design of the rover and thus the inability of the RCE to make an assessment of a particular component's state of health.
- **Emergency Shutdown Sequence:** The emergency shutdown sequence aimed to replicate the similar sequence on *Curiosity*, designed to handle bringing the rover into a safe state in the event of a detected failure in any aspect of the rover.

Hardware Abstraction

Figure 3.45 shows the control layer as being a collection of modules allowing commands from the communication layer to be passed through abstractions and translations to result in driving signals sent to the various hardware components. The required signals at the end of the control pipeline were abstracted so as to provide an easier design platform upon which the command translation and execution could be based. Two elements of abstraction were included at the end of the pipeline, one of them being that which was provided by the Johnny-Five framework for all hardware control except the camera. The principle behind Johnny-Five includes hardware components in the form of classes which are instantiated alongside a single `Board` instance often referred to as the "Control System". Each of the components utilise a "Controller" which describes the translation required from the API provided by the component to the peripheral hardware. The second layer of abstraction was custom designed to raise the level of control even further, allowing a more understandable interoperation between it and the rest of the RCE. The second level also consumed configuration and platform specific details and features with respect to the rover hardware and software design (i.e. it was not developed to be scalable or reusable). The structure of this abstraction is shown in Figure 3.48, beginning with the `Board` instance. Each of the second layer abstraction APIs contained an initialisation operation which could be executed from the startup system sequence, as well as multiple hardware specific operations as described in Section 3.6.

Command Structure and Design

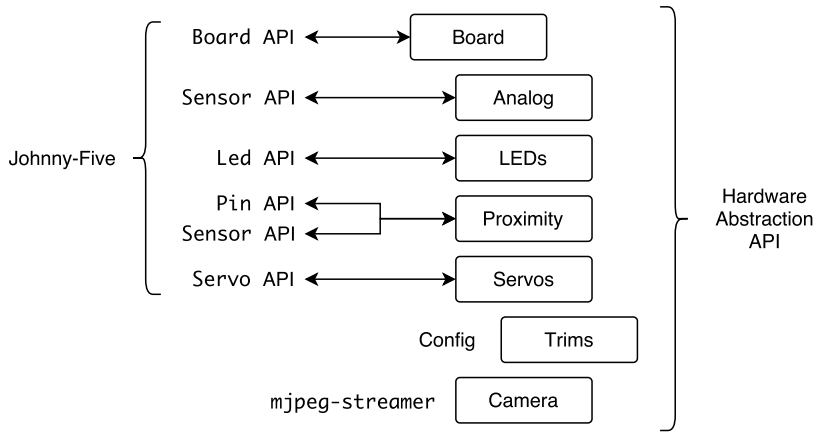


Figure 3.48: Diagram showing the structure of the two layered harware abstraction.

An internal command execution system was designed for the RCE to cater for the large number of components that could be controlled as well as to provide a clean and semantically understandable method of control for the user. The command system also aimed to be similar to the command system employed for the control of *Curiosity* (and other such rovers that utilised the RSVP) so that the control interface could be accurately portrayed. The system was designed for both interactive and RoSE style control modes as will be discussed after this sub-section.

The two primary features intended to be controlled through the use of commands was the traversal of the rover and the movement of the camera, both of which involved the servo motors used for their actuation. These two control subjects were prioritised over commands relating to secondary components and operations. Regardless of the subject, however, all commands were classified as either being low level commands to effect a single change on one hardware feature, high level commands which may involve controlling more than one component at one time, and macros which contained a sequence of operations which could either be high or low in level. The macro was then to be decoded into the respective high and low level commands and inserted into the sequence, or executed as a sequence itself.

The user may construct a sequence of commands of all types on the RSVP Client application and transmit the sequence to the RCE (via the Server) which would then be processed and executed, as described further. Therefore, the command shape and the details within were required to be human understandable and human constructable. Figure 3.49 shows the template composition of commands of each level of complexity and the inheritance of common fields by each of the types.

Command Translation and Execution

The two types of user control, interactive control and RoSE control, required to some extent separate command pipelines in that the data acquired from input elements a part of the RSVP Client application differed in shape and frequency. However, due to the command system design, both forms of control were processed to eventually comply with

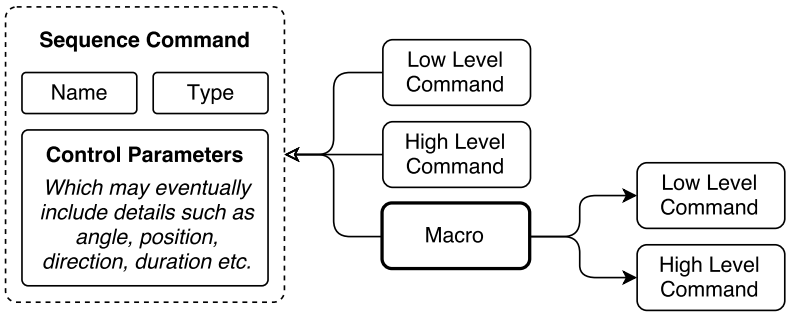


Figure 3.49: Diagram showing the types of commands, their object inheritance and macro decoding.

the standard execution process which allowed for the differing pipelines to merge for the majority of the system. More specifically, the point at which the pipelines merged required the input commands or signals to be in the form of high or low level commands with their parameters fully completed, which can be seen in Figure 3.50.

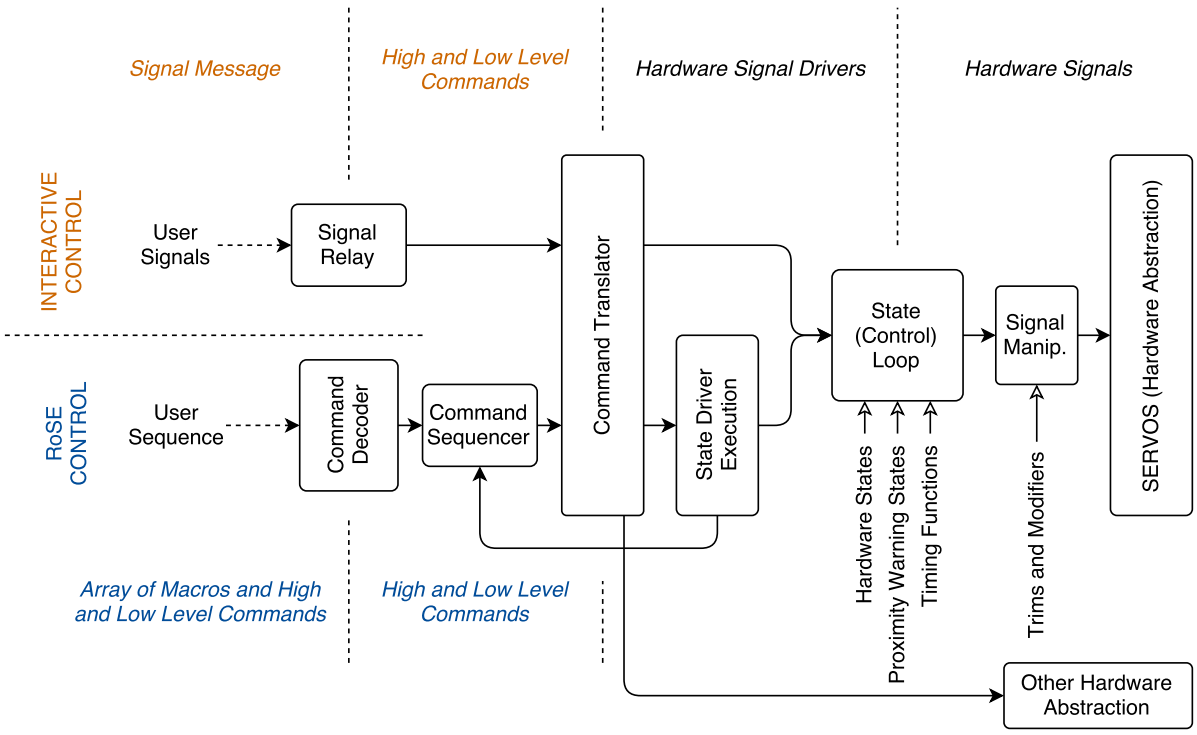


Figure 3.50: Diagram showing the flow of commands through translation and execution operations.

Beginning with the interactive type control, the signals received from the RSVP Client described the user's input with respect to the controller as part of the front-end interface. For each of the controllers, a manipulation relay was designed to convert the signals into equivalent commands analogous to a mapping of the controller to the command it represented. For example, rotating the traversal joystick forward was mapped to creating a "drive" command with a forwards velocity and no steering of the wheels. The result was a high or low level command which reached the command translator to be further

processed. For the RoSE control style, the user composes a sequence of commands and when transmitted to the RCE, is passed through the command decoder which takes macros present in the input array and constructs the equivalent sequence of high and low level commands. The command sequencer handles dispatching the sequence of commands in coordination with the execution of such commands further down the pipeline as well as provides telemetry via the communication layer indicating the state and progress of the sequence.

It is at this point that the two control type pipelines merge where the high and low level commands are translated into "state drivers". From an object-oriented software perspective, a state driver instance contained the required signal data to drive hardware outputs, the aggression with which to drive the signal output towards the setpoint signals (referred to as the "velocity") and enriching data such as the duration of the signal and the period before which the command sequencer is notified of the command having been completed, useful when the transition to the new signal setpoint did not match the duration of the hardware effect.

Figure 3.50 omits a step between the command translator and the state driver executor responsible for managing the timing of the state drivers, namely the dispatcher. The dispatch step assesses the data in the state driver and if no timing detail is included, dispatches the signal driver to the state/control loop immediately. If timing data is present and the state driver contains signals which are to be effected only for a specified duration, the state driver is sent to the state driver executor which will record the state of the hardware signals prior to driving the signals. Once the new signals are effected for the specified duration, the state driver executor drives the originally recorded signals so as to return the hardware to the previous states.

The final step in the pipeline is the state loop (also referred to as the control loop). Signal drivers contained the desired setpoints of the hardware and the velocity, which is not to be confused with signal duration. Therefore, a loop was required to interpolate between start and end signal values in real time, which was implemented by means of an interval operation taking the setpoints and the duration of the setpoint having remained the same and using these details to calculate the resulting hardware signal output. Another feature was included in the state loop which allowed specification of the interpolation curve or timing function. As such, a timing function property was included in the state driver object allowing this specification to be passed from command to state loop. Allowing the system to specify a timing function presented a simple method to effect fluid motion of the servos thus preventing damage to the servo mounts, horns, gears and the wheels themselves. It was at this point that, in the case of an obstacle warning from the proximity subsystem or an emergency shutdown, all hardware control could be blocked for safety. The signals are passed through a manipulation block used to provide servo trim configuration (separate from the offset configuration exposed by J5) and output modifiers, functions to modify the signal due to hardware characteristics, after which the hardware abstraction layer is utilised to effect signals.

For hardware other than the servos, the translation performed in the command translator block used the relevant hardware abstraction API instead of invoking the signal dispatcher,

as seen in Figure 3.50.

Communication Transport Layer

All three servers interfaced with the wireless module by default (an operating system level interface). A common pattern that was carried across from this software component to the RSVP Server and Client was the pairing of WebSocket server modules with an accompanying translator module. The endpoint-translator structure and interface is discussed in more detail in Section 3.4.4. It can be seen in Figure 3.45 that each part of the communication layer is exposed to the sequencing block, allowing them to be initialised and configured during a sequence as well as allowing for sequences to be triggered in an event-driven fashion. The RCEIO server was designed to handle message data of both control and telemetry types. It was decided that the implementation of RCEIO be a server endpoint and not a client endpoint so that the connection process on the RCE side remained as static as possible (i.e. requiring little configuration and logic). This left much of the network resolving and connection procedure up to the RSVP Server, a software component far more accessible to an operator if such access was required.

It was decided that the streaming instance remain separate from the data communication since the choice of WebRTC as the media streaming technology. However, after small-scale proof of concept development and tests, it was found that WebRTC, including but not specifically Kurento, were tools more suited to a web browser environment. WebRTC primarily makes use of the browser-camera interface provided by browser vendors and fair but above-par support for a Node.js environment for anything other than the server. The RCE required a lower level of control for the web camera and this included the process of obtaining video data from it. As such, an approach rawer in operation to that of a WebRTC endpoint was to use a Linux tool which streamed a compressed but mostly unaltered feed directly from the camera, of which there were many as discussed in Section 3.6, and did so by means of a Real Time Streaming Protocol (RTSP) server to which another entity on the network would connect on a separate port. As shown in Figure 3.51, the Node.js RCE process (in the Linux operating system sense of the word) spawns a new process, referred to as a "child process", which would invoke the streaming server tool and essentially run parallel to the RCE. This was beneficial to the design in that video streaming operations would not block the execution of operations as part of the RCE which would have negatively impacted critical functionality such as the carrying out of hardware control.

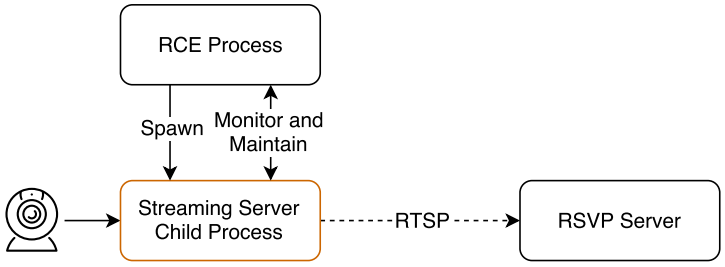


Figure 3.51: Diagram showing the spawning and interaction of a separate streaming process by the RCE.

RSVP Server Design

Plan of Structure

The RSVP Server was simpler in functional composition as it primarily handled communication. As with the RCE design, the Server consisted of the Node.js platform and entry point and a system module which maintained the operation of the rest of the subcomponents. The Server consisted of four server endpoints, one of which was an HTTP server and the rest WebSockets, and one WebSocket client. All of the servers as well as the client-endpoint operated over the network interface provided by the Node.js runtime. The same data store structure used for the RCE was present in the Server design for state data and telemetry. Additionally, a connected client management module was added to the design to manage RSVP Clients in terms of their access to various parts of the system.

The Kurento Media Server is included in Figure 3.52 despite it being a separate system process on the host operating system. The interface between this Media Server and the RSVP Server is discussed below.

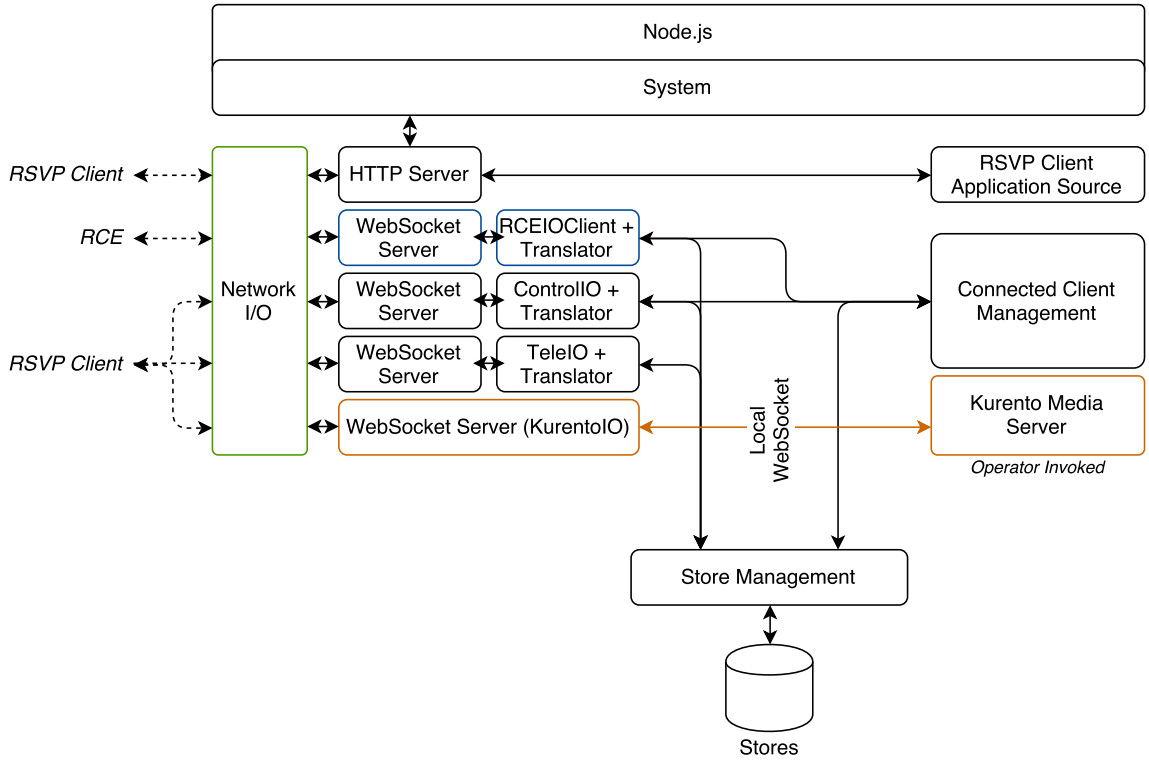


Figure 3.52: Block diagram of the designed software structure for the RSVP Server.

RCEIOClient

The RCEIOClient Websocket client-type endpoint served as the single message data communication link between the Server and the RCE, and as such, catered for control (uplink) and telemetry (primarily downlink) data. The RCEIOClient translator module routed incoming messages between the two Client-Server WebSocket channels where appropriate or stored incoming data in the data store. Conversely, messages from the two Client-Server channels intended for the RCE converged via the translator.

ControlIO and TeleIO

Communication between the RSVP Server and connected RSVP Clients was split between two WebSocket channels, the first being ControlIO. This channel was responsible for transmission of commands and control signals from the Client to the RCE, thus resulting in communication primarily in the uplink direction. Additional messages such as indication of the type control in use were communicated via this channel. Making this channel separate made it easier to securely restrict the rover control access to only one RSVP Client instance, managed by the connected client management system as discussed below. Software semantics played a part in this design choice as well.

The TeleIO client channel was used for all telemetry data from either the RCE (although not connected, via RCEIOClient), Server or the Client.

Client Management

The requirement for the control of the rover to be suitable in an educational environment or in places where many users might want to gain control created the need for management of the RSVP Client instances, specifically their allowed level of control. As such, the connected client management system was included in the Server design and utilised Socket.io's exposure of the list of connections on a particular channel. Although only one Client would have access to control communication at one time, all Clients were connected to the ControlIO channel and thus the management system used the list of Clients, each identified by a random, Socket.io-generated string ID, to keep track of disconnecting and connecting Clients. A proxy was built into the ControlIO pipeline which allowed the management system to block any attempted control communication from a Client whose ID did not match the ID of the Client allowed access.

For the purpose of this project, the basis upon which Clients were granted access to rover control encompassed the first-come first-serve principle. This meant that Clients were required to request access for control in which case their ControlIO connection ID was inserted into a queue and once the ID reached the top, access was given (with notification thereof). The Client remained in control until disconnection or if they Client opted to relinquish access. However, the system was designed to be flexible and implementing other principles of scheduling control access could be achieved with ease requiring only a change of logic in the manipulation of the queue or at the point of proxy in ControlIO.

Kurento and Media Broadcasting

The RSVP Server was responsible for the majority of the video broadcasting pipeline and thus the Kurento implementation made up a large part of the design. As discussed in previous sections, the RCE streamed the video via an RTSP connection to obtain the raw video data. The data was then passed into the implementation of the Kurento framework where it was streamed to the connected RSVP Clients via the Kurento Media Server. Figure 3.53 shows the relationships between each of the components involved in the video streaming mechanism from the RCE to the point at which RSVP Client instances connected to receive the broadcast. The diagram makes clear the difference

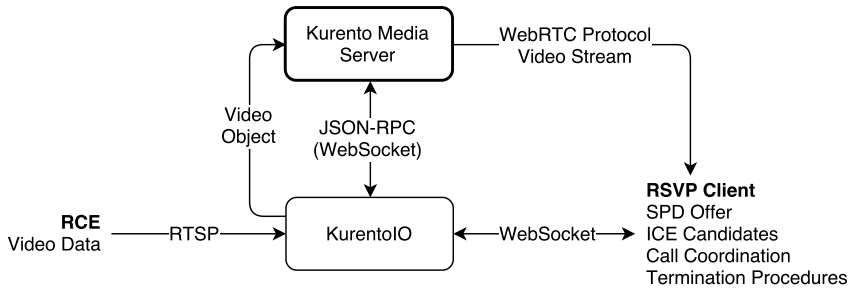


Figure 3.53: Diagram of the server-side relationships with the video stream components in the RCE-RSVP system.

in function of the KurentoIO WebSocket server and the Kurento Media Server. A key detail in the designed implementation of the broadcasting system was the fact that the Media Server ran as a separate process, as it was, in fact, a separate utility altogether. The intention behind bringing an intermediate server into the peer-to-peer principle of WebRTC communication was to improve on multimedia streaming and calling in areas such as multi-client broadcast which was a significant advantage for the nature of this project, transcoding multimedia (and the ability to offer multiple types of the same broadcast source to various different clients), recording, mixing and many other features [66]. In the context of the RSVP Server, the KurentoIO subcomponent coordinated incoming "calls" or requested connections from Client instances via its WebSocket endpoint which was easily accessible by the Clients over the network. This method of communication was utilised throughout the connection negotiation process whereby the client initiates such a negotiation with an SDP offer making the Kurento mechanism component of KurentoIO aware of the type and shape of the Client concerned. The Server responds to the offer made and this cues the Client to send multiple ICE candidates each describing supported multimedia capabilities of the Client system as well as available transport mechanisms. The Server does the same and when a pair of ICE candidates agree, the connection is established. At the same time, communication is made with the Media Server process by means of yet another WebSocket connection along which the Kurento Protocol (JSON-RPC) is used to offload the connection established onto the Media Server. From this point onwards, the RSVP Client communicates directly with the Media Server. Network change negotiations as well as disconnection procedures are also performed on the KurentoIO connection.

RSVP Client Design

Plan of Structure

The RSVP Client was a tightly integrated blend of static front-end components and JavaScript modules to turn the web view into a fully functional and responsive SPA. Figure 3.54 shows the Client application code served from the RSVP Server as well as the WebSocket client-translator pairs for the discussed channels of communication including the implementation of the client-side Kurento library. The large development portion of this software component was the UI (user interface) which will make up the majority of this subsection.

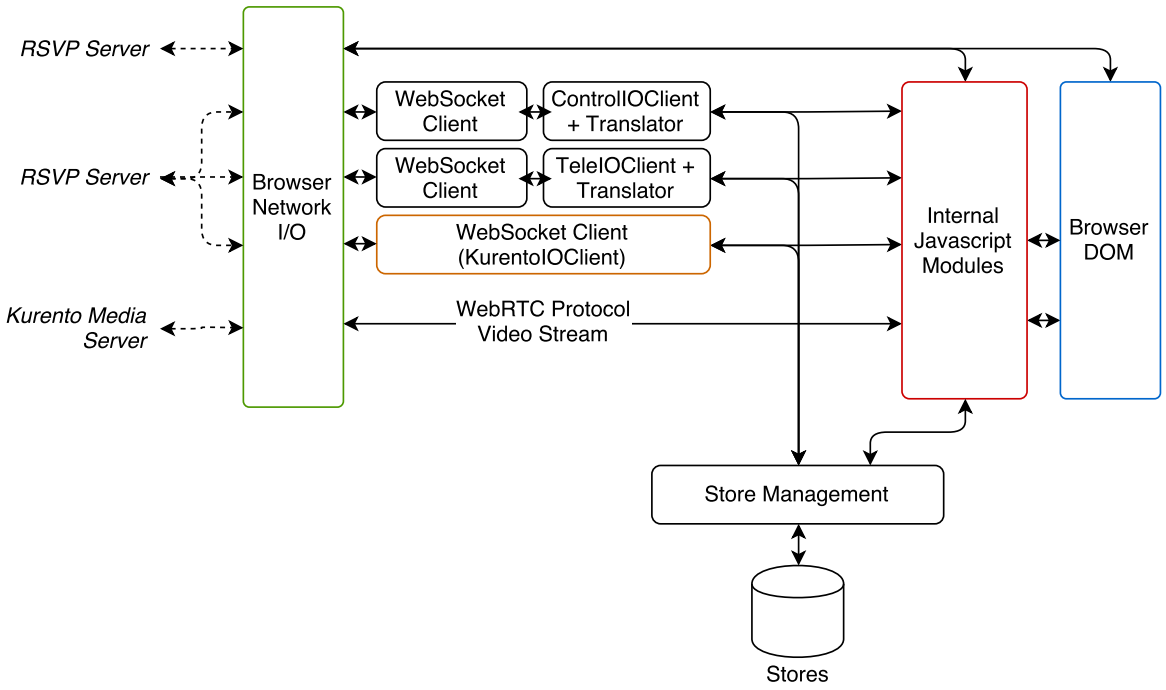


Figure 3.54: Block diagram of the designed software structure for the RSVP Server.

Layout Overview

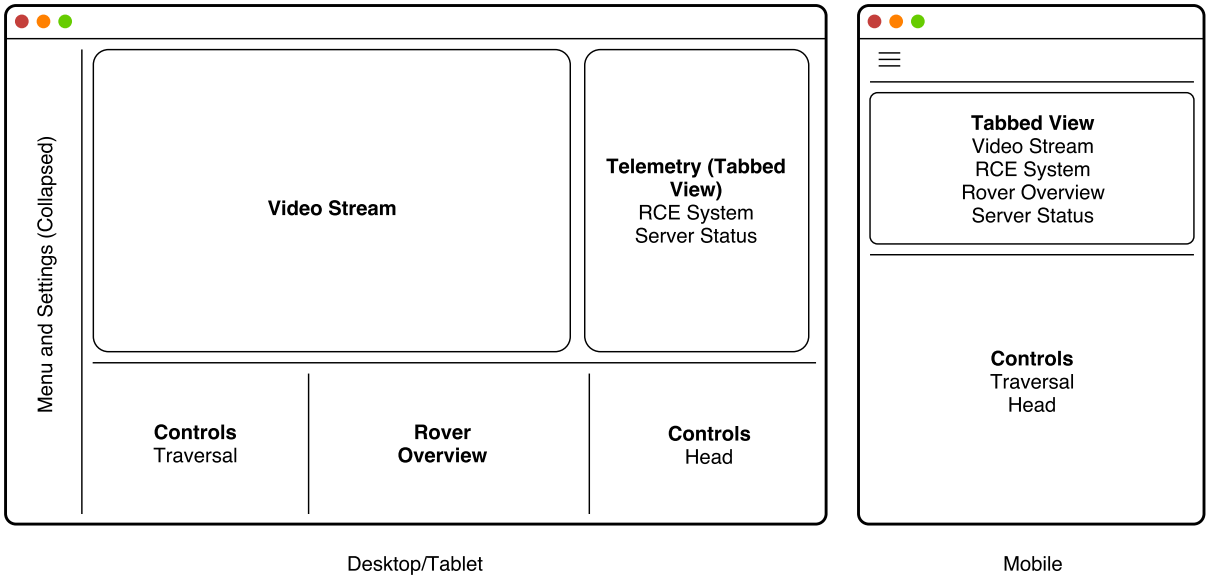
The style of layout of the UI at the highest level was in accordance with Google's Material Design specifications for applications, and consisted of an application content area with a swipeable menu panel on the left hand side. It was decided that two screens were to be developed, each one for each of the control styles. Prior to development of the UI using the technologies discussed, the two screens, menu panel, and each of the components in each of the screens were sketched to verify the validity of the layout and to ensure consistency of design throughout the development process, one prone to drift in style and design over time.

Figure 3.55 shows the general layout plan of each of the two screens as well as the mobile equivalents. The project specifications indicated the requirement for the application to be mobile compatible, thus included in every design stage made in the desktop environment was a plan of the mobile version of that particular stage.

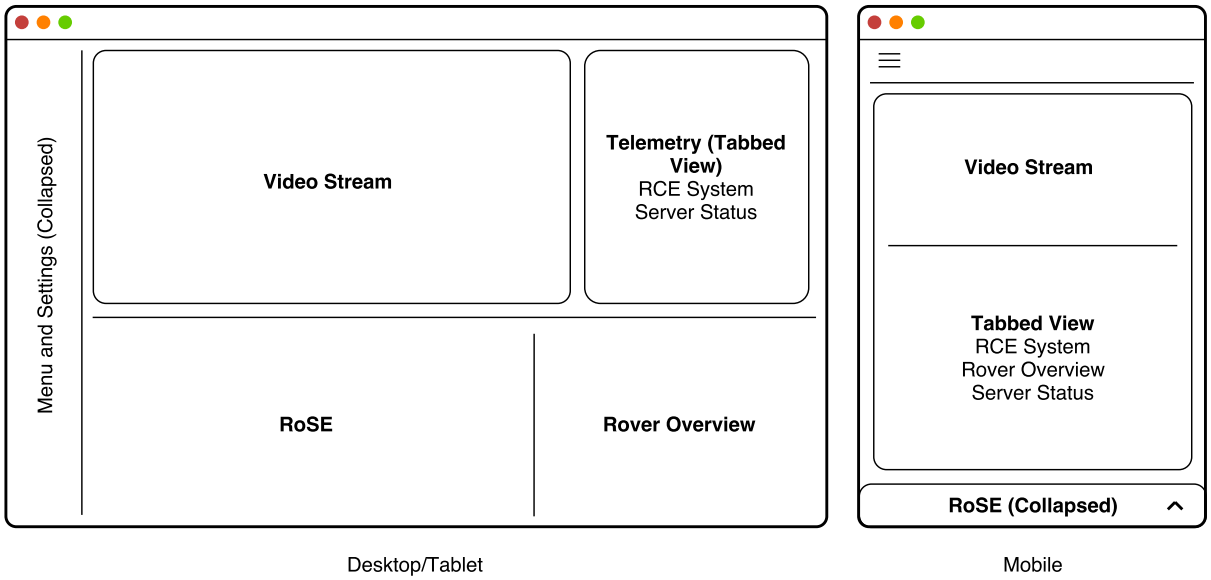
Each of the UI components shown in Figure 3.55 are discussed below.

Video Stream Component

The stream section of the UI was designed to be simply a container element for the HTML video element which was to be attached to the Kurento framework instance. During page load, an internal JavaScript module reaches into the DOM and returns the `video` element to the Kurento framework, which handles displaying the received video stream in it.



(a) Interactive control mode.



(b) RoSE control mode.

Figure 3.55: Layout plan of the RSVP Client User Interface for both methods of control.

Rover Controls (Interactive)

The rover controls component, a mockup of which is included as Figure 3.56, was split into two sections, each positioned on the left- and right-hand bottom corners of the content area of both the desktop and mobile configurations. The left hand subcomponent holds the controls for rover traversal, which consist of an interactive two-dimensional joystick and two buttons for rotating the rover about it's central axis. The y-axis of the joystick controls the forwards and reverse velocity of the rover and the x-axis of the joystick controls the amount by which the rover will turn given some component of forwards or reverse velocity (in other words, the amount by which the wheels are rotated along the steering axis). The right component controls the angular position of the head about its pan and pitch axes and is sticky in behaviour allowing the user to position the head and

have it remain in that position. A button above the head joystick allows the user to center the head upon pressing it.

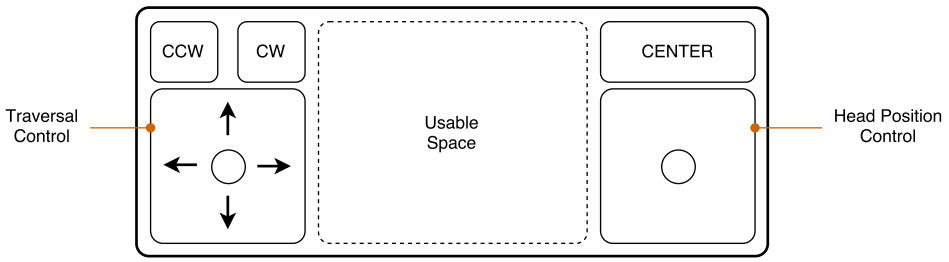


Figure 3.56: Diagrammatic mockup of the interactive controls component of the UI.

Rover Overview Component

The rover overview component of the UI was designed to be a highly engaging graphical representation of the rover as a means of displaying telemetry from the rover's control software subcomponents. The design of the rover did not include feedback sensory hardware, thus the data used in this overview element was that of the signals used to drive the hardware. The data still gives an accurate estimation of the hardware state of the rover. The component consisted of an SVG graphic of the top view of the rover with wheels and a head that could dynamically rotate in-view. Included in the graphic were indicators for the proximity sensor data including whether or not they were triggering proximity/obstacle warnings. On either side of the component, sections for each of the hardware components displayed textual telemetry of their state, as seen in the mockup in Figure 3.57. The overview component was placed in the usable space (as in Figure 3.56) between both control element sections.

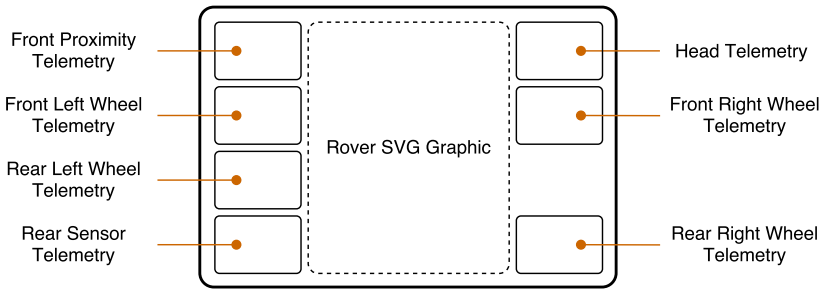


Figure 3.57: Diagrammatic mockup of the rover overview component of the UI.

Rover Sequence Editor Component

The RoSE component replaced the rover controls component when switching from the interactive style of control to the RoSE style. The component was designed to allow the user to compose a sequence of commands, view and edit the sequence and transmit the sequence to the rover for execution. Further, the component included controls for the control of execution of the sequence as well as reported back the status and progress of the sequence execution.

Figure 3.58 shows the list of command items in the main area of the component. A new command could be created by clicking on the "plus" icon on the bottom bar section

of the component, which would show a dialog box allowing the user to choose which command to create and afterwards a series of controls for entering the parameters of the command. Each command item contained the name of the command and a summary of the parameters that were entered into the new command dialog box. On the left of the command name, an icon showed the type of command (low-level, high-level or macro) and on the far right of the item an edit button to bring up the dialog box to edit the parameters of the command and a delete button to remove the command from the sequence. The items in the list could be reordered by dragging the item to a new location in the sequence.

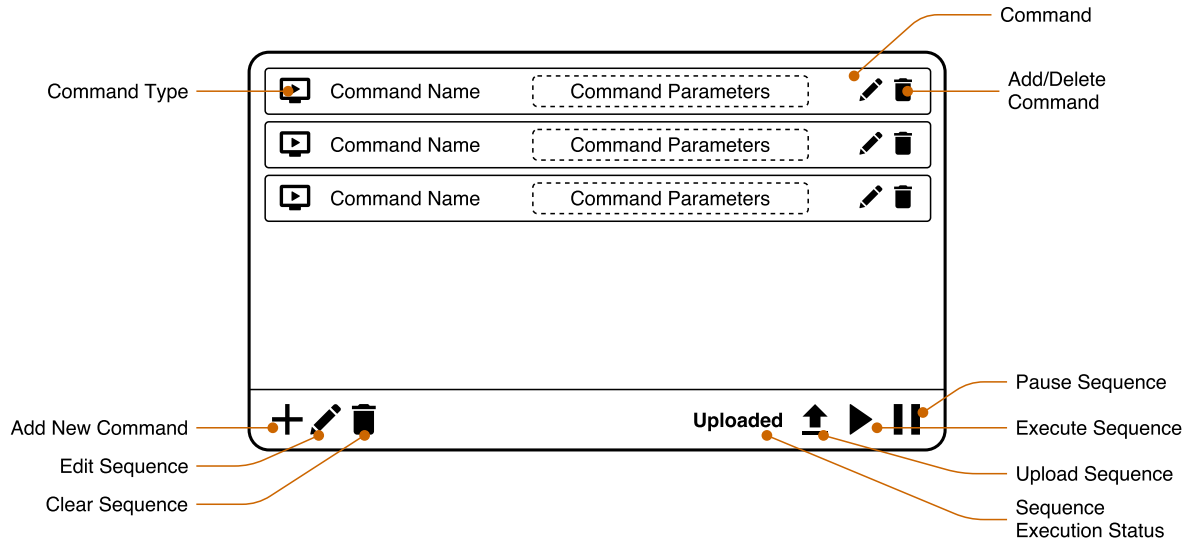


Figure 3.58: Diagrammatic mockup of the RoSE-style sequence editor component of the UI.

The sequence may be uploaded when complete, and executed if the RCE acknowledged the new sequence. The controls for uploading and controlling the execution of the uploaded sequence were placed on the right hand side of the bottom bar section. To simulate the strict procedure when creating, editing and executing sequences on the RSVP for *Curiosity*, the sequence editor component only allowed the sequence to be executed if all new changes had been uploaded and disallowed editing if a sequence was uploaded until the edit button on the left hand side of the bottom bar section was pressed to put the component back into editing mode.

During execution, the RCE indicated the command that was being executed in real time and the sequence editor component highlighted that command to show the user the progress of execution.

RCE System View

To the left of the video stream container in the desktop layout (for both interactive and RoSE control configurations shown in Figure 3.54) was a tabbed view which contained two views, one of which was the RCE system view component. Telemetry related to the RCE system itself and various subsystem status data was shown in this view. The top section of the view consisted of four dial components to show the CPU and RAM usage of

both the RCE Node.js process and the separate video stream utility process as discussed in Section 3.4.4. Below the dials was a list of the hardware control software components showing the initialisation and online status of each. Figure 3.59 shows the component design detail.

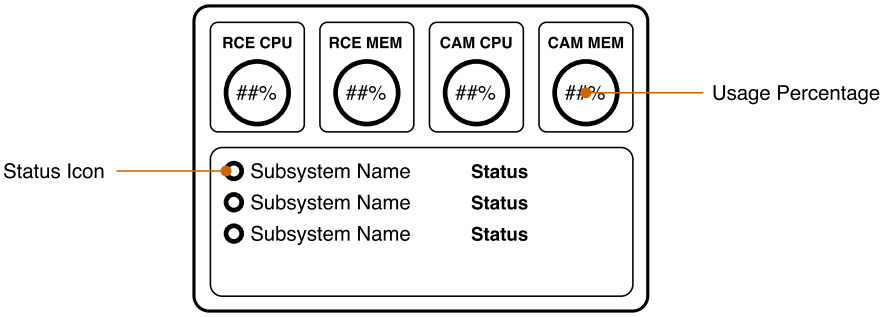


Figure 3.59: Diagrammatic mockup of the RCE system view component of tabbed telemetry container in the UI.

Server Status View

The server status view was another page in the tabbed view as used for the RCE system view. This component was similar to the RCE system view, however, it showed data related to the RSVP Server. This was designed to give the user more insight into the real time operation of the server as well as be a helpful tool in the monitoring of the server for maintainers of the RSVP system.

The server status view consisted of the status of the server instance, that of the Kurento Media Server instance as well as how many RSVP Client instances were connected to the server at that point in time.

Menu Component

As mentioned, the application included a side panel that was designed to serve as a menu. Since the application did not require many pages or views, the menu was more functional than navigational, a classic menu use case. The menu allowed the user to perform the following operations:

- **Start Stream:** The "Start Stream" button allowed the user to initiate the WebRTC call in an attempt to begin to receive the video stream. When the application was initially loaded, the stream started automatically. In the event that that user decided to stop the stream, or the stream disconnected due to a network error, the user could make use of this button to resume the stream.
- **Request Control:** The "Request Control" button allowed the user to place their instance of the RSVP Client in the queue to gain access to control of the rover.
- **Control Dropdown:** The "Control" dropdown element revealed two controls related to the style of control in use. The first control was a switch element which switched the control style between "interactive" and "RoSE". The second control allowed the user to select the level of difficulty of the simulation of control.

- **Settings Dropdown:** The "Settings" dropdown element revealed a set of settings controls which included:
  - RCE endpoint detail configuration (IP address)
  - a button to open the trim edit dialog
  - a button to reset the RCE if required (and the user has access to this command)

Common Software Components Design

Developing the full-stack in JavaScript meant that many of the subcomponents and features could share design patterns, technology choices and code implementations without the abruptness of having to design for differing language and/or platforms. It was quickly identified that many of these subcomponents were required in all three of the software components and as such, are discussed in this section.

Synchronised Data Store

During the design process for each of the software components, the need for a mechanism to reliably store global state data was realised. Operations across all modules within each of the components relied on the state of many other modules and thus a well designed central store for this state data was recognised as being beneficial to the design. More importantly, however, was the need for the communication of this state data between each of the components. An example of such was the need for the RSVP Server to be aware of the initialisation status of the camera on the rover. Once the camera was initialised, that state change was required to be communicated across the network to indicate to the RSVP Server to initiate a connection with the camera streaming port on the RCE and to offer the stream to the Kurento Media Server.

A further important detail in the data store requirement was for it to be asynchronously notifying. The nature of Node.js and JavaScript in the browser was one that was event-driven and this applied to data changes as well. The alternative to having event-based data change notifications was for points in the software components that require knowledge of changes in data to poll the central store of data at some predefined frequency and compare the data in search of differences. Since Node.js is single threaded, this would have had significantly negative impacts on the performance of the software components, especially in the context of this project where state data was prevalent.

The opportunity to use the data store module for easy telemetric broadcast was also recognised during the design process. Data in the form of telemetry was able to be saved to the store and the configuration of the store structure would determine the channels along which such data would be broadcast, all of which would happen asynchronously.

Operations that were required from the data store API included the ability to:

- set data,
- read data,

- register callback functions that would be executed upon the change of data at a particular path (i.e. listen to data changes),
- listen to data changes at any level below a particular path in the data structure,
- specify whether or not the data at a particular path was to be broadcast over a chosen WebSocket channel upon change,
- receive data from store event emitters on other software components, across the network,
- request that already existing data be broadcast from one software component to another, and
- re-notify all appropriate listeners and WebSocket channel clients of already existing data.

The module was designed to include a data store class definition which incorporated the required methods and event emitter features to provide the required functionality. For each of the software components, classes were instantiated with the required fields and notification channels and exposed to the rest of the modules in each system. Figure 3.60 shows the internal structure of one such data store and the use of the API in a typical read, set and notify scenario.

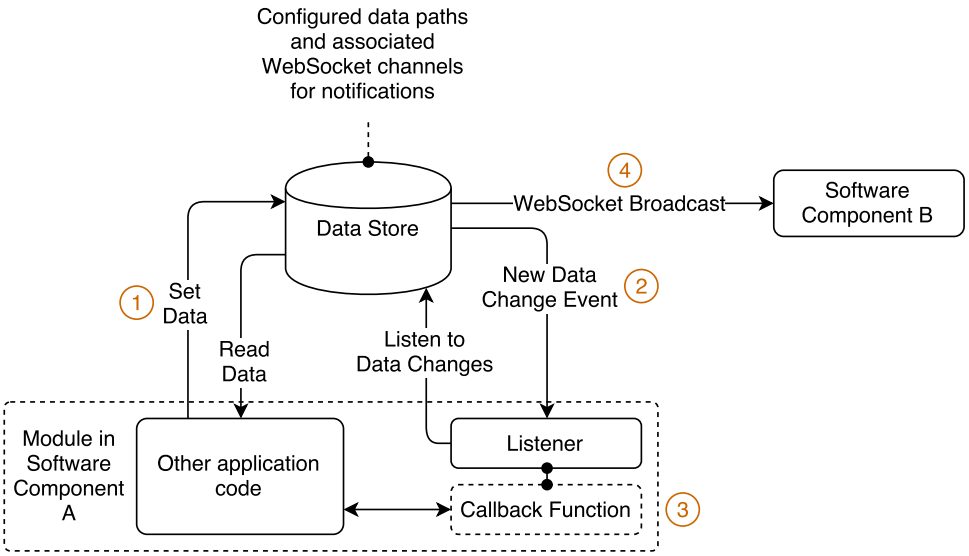


Figure 3.60: Diagram showing a functional sequence of typical interface between a data store instance and application code.

Software component A and in Figure 3.60 could each be either of the RCE, RSVP Server or RSVP Client software components. Component A creates a listener on a particular path's change of data with an associated callback function. Code elsewhere in the component sets new data to the store at that path at point **1** using the data store's `set` API method. This triggers an event to be fired which matches that of the listener registered previously, and thus the new data event reaches the listener at point **2** as well as the callback function is executed at point **3**. At the same time, a notification of the data change is broadcast to software component B via the configured WebSocket channel (which in this case could be either RCEIO, ControlIO or TeleIO) at point **4**. The same

flow of operations would apply to the case where an change notification was received from software component B, in which case the callback function would be fired after having the data be received and "set" internally.

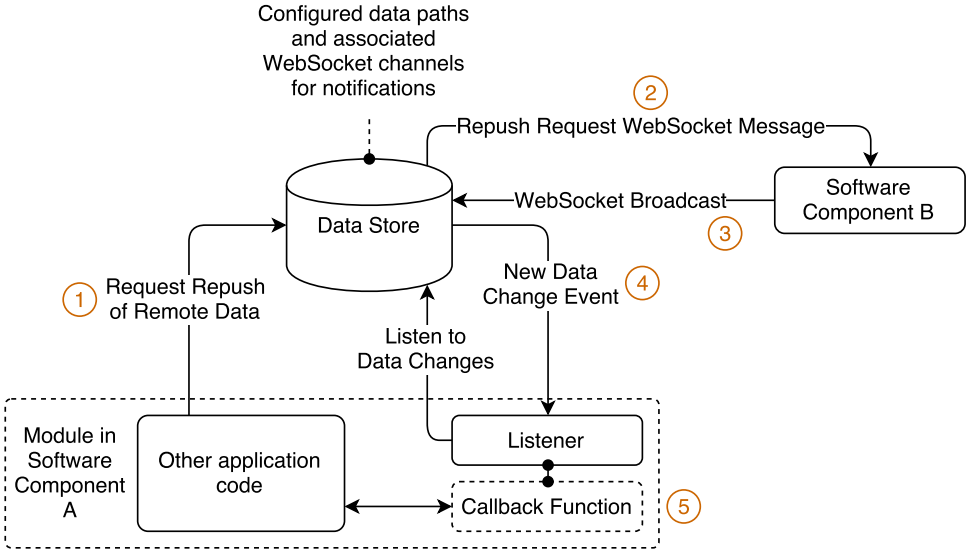


Figure 3.61: Diagram showing a functional sequence of a request for a repush of data from a remote data store.

Figure 3.61 shows the flow of operations in the event that the application code of software component A requests that software component B resends (repush's) all data at a particular path (point **1**). The request is sent to component B via a WebSocket channel at point **2** and the component repushes the data at point **3** making use of its data store's `repush` API method. The same process of the event reaching the listener and the callback being executed applies as in the first example. Requesting data was particularly useful when one or more of the software components had to be restarted and required resynchronisation of its data with the already online software components.

WebSocket Design

As described, the WebSocket implementations comprised of the server/client endpoint (the instance of Socket.io) and a second module, named the translator, which served as a handler of incoming and outgoing messages. Since there were many WebSocket connections and hence subcomponents, a naming convention was introduced where the name of the endpoint subcomponent was formulated by `<subject>IO` and `Client` was appended if the endpoint was to connect to an already existing WebSocket server. Further, `Translator` was appended to the translator half of the WebSocket implementation. Figure 3.62 shows the designed interface between the WebSocket implementation subcomponents and the RCE.

A simple mapping of incoming message events and outgoing message methods was made between the Socket.io instance and the translator. On incoming messages, the translator decides the correct behaviour based on the type of message as well as, in some cases, data sent with the message. The translator also exposes an API which can be used as convenience methods to send standard types of messages. This relieves application

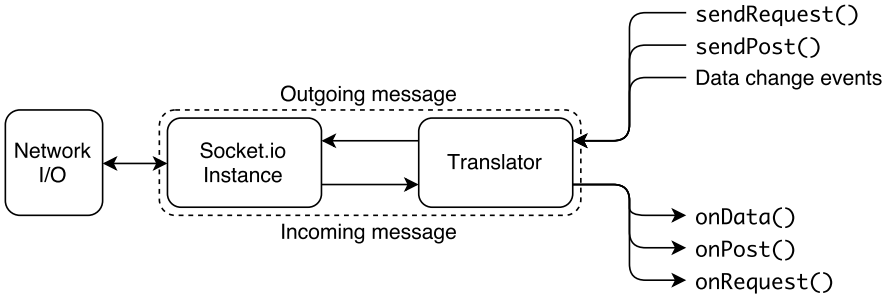


Figure 3.62: Diagram showing the interface with and structure of the Socket.io-translator module pair.

code having to package the message and payload when sending such a message, which could have also become a point of weakness in case of programmatic errors or change in design. Data change events from data stores are also fed into the translator to notify other software components.

3.5 Vehicle Build and Manufacture

The project focus was primarily the design of the rover model and the accompanying software and electronic systems. However, a large portion of the effort was put into the manufacture of the model which will be covered briefly in this section. The manufacture process began with plans and analysis of the design, collation of a bill of materials and finally manufacture and assembly of the rover.

3.5.1 Manufacturing Plan

Analysis of 3D Components

Many of the 3D components formed part of the suspension system, the mechanism which took on the largest stresses compared to the rest of the model. For this reason, correct printing orientations were important for the strength and robustness of the rover. While the printing direction (layer direction) and the coupling of it with both geometrical limitations and stress orientations were taken into account during the design of each piece, a further visual analysis was performed on each part to decide upon the most suitable printing orientation.

Design Preparation

At this point, the entire design was in the form of 3D models and 2D CAD drawings. Printing of the 3D components required exporting of the design files to the correct format. Manufacturer specifications required the object data in `STL` format, an export operation provided by the 3D CAD tool used to design the components. Each file was exported and verified.

The laser cutting process for the acrylic body panels required the 2D detail drawings to be in `PDF` format which was performed and verified as well.

3.5.2 Bill of Materials

Table 3.10 shows the bill of materials for the mechanical and electrical systems of the rover.

Item	Description	Quantity
Intel Edison	With Arduino Extension Board	1
Adafruit PWM Shield	16 Channel, I^2C	1
XL4015 Buck DC-DC Converter	5A, 1.25-36V Output	1
HC-SR04 Ultrasonic Distance Sensor		3
Hextronik HXT900 9G Servo	Plastic gear	10
Servo Extension Cables	150mm	10
USB Web Camera	msi Starcam Flip - UVC compatible	1
Lithium Polymer Battery Pack	3 Cell, 100mAh	1
LM358 Operational Amplifier	DIP	3
10μF Capacitor		3
Dual Row Male Headers	4-pins wide	3
Single Row Male Headers	4-pins wide	3
Single Row Female Headers	4-pins wide	12
M2 Hex Nut		58
M2 Pan Slotted Screw		58
M2 Narrow Washer		≈ 100
M3 Hex Nut		80
M3 Pan Slotted Screw		74
M3 Narrow Washer		≈ 50
Round Extrude Aluminium Tube	15.88mm×1.62mm	1m
Solid Aluminium Rod	6mm	1m
Ball Bearings	5×16×5, Stainless steel, shielded	13
Neodymium Disc Magnets	3mm×2mm	8
Acrylic Sheet	3mm, Soft white, A3	1
3D Printed Components	As exported from CAD	

Table 3.10: Table of the bill of materials for the mechanical and electrical systems of the rover.

3.5.3 Vehicle Assembly

Manufacture of 3D Parts

After the analysis on the designed 3D components was performed and the files exported to the correct format, one of many materials that could be used as filament in the printing process was chosen. The two most common of these materials were PLA (Polylactic Acid) and ABS (Acrylonitrile Butadiene Styrene), each a type of thermoplastic with associated thermal and physical properties. The more widely used of the two was PLA due to a significantly lower glass-point temperature meaning the required temperature of the heating element of the printer nozzle be less than that when printing in ABS. However, it was found that PLA was more brittle, allowing for less flexure before cracking and shattering. ABS, the same plastic used for Lego bricks, allowed for greater flexibility and was observed to plastically deform long before cracking. For this reason, ABS was chosen as it would offer the durability in the pieces that were to undergo higher bending stresses (such as the joints of the suspension system).

Due to the volume of printing required, external, local ABS printing services had to be found to have the parts manufactured. Multiple companies were contacted for quotes, however, the budget of the project was not able to include the cost of such a service. A company was found, ProtoLink 3D[1], which was willing to fully sponsor the project.

Once the parts were printed, all support material and extra plastic was removed and prepared for assembly. All the holes were inspected for correct position for alignment with other pieces and for size. Features that were not correct were altered either using a drill press or a hot piece of steel. Figure 3.63 shows the removal of support material from a component.

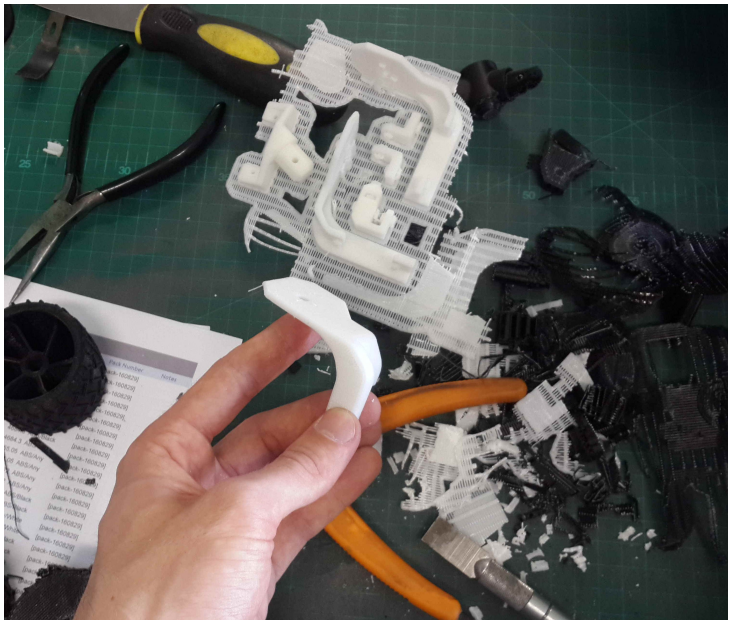


Figure 3.63: Image of a printed component (a wheel pivot) with the plastic support and extra material removed.

Assembly of Suspension and Body

The five acrylic panels that made up the body component of the rover were joined and glued using Tensol Number 17. This resulted in the tabbed edges of the panels to fuse together providing a strong structure for external and internal mounting of components.

Assembly of the suspension system involved the printed joints, aluminium tube pieces, aluminium shafts and bearings. The aluminium tubing was cut to size and the edges finished. Holes were drilled at each end of the tube pieces for fastening to the printed joint components. Aluminium shafts required for the rocker and bogies joints as well as the center wheels were also cut to size and chamfered. The bearings were press-fit into the printed joints and the shafts were inserted into the mounted bearings using a stationary drill press. Some of the shafts were too thick to be easily but securely pressed into the bearings and had to be reduced in size. The aluminium tube pieces were press-fit onto the hexagonal joint plugs and secured using nuts and bolts. Figure 3.64 shows one side of the suspension system without the servo-mounted components.

[1]ProtoLink 3D - `http://www.protolink3d.co.za/`

The servos and servo horns were also fastened to the wheel pivots, struts and the wheels themselves, completing the assembly of the body and suspension system.

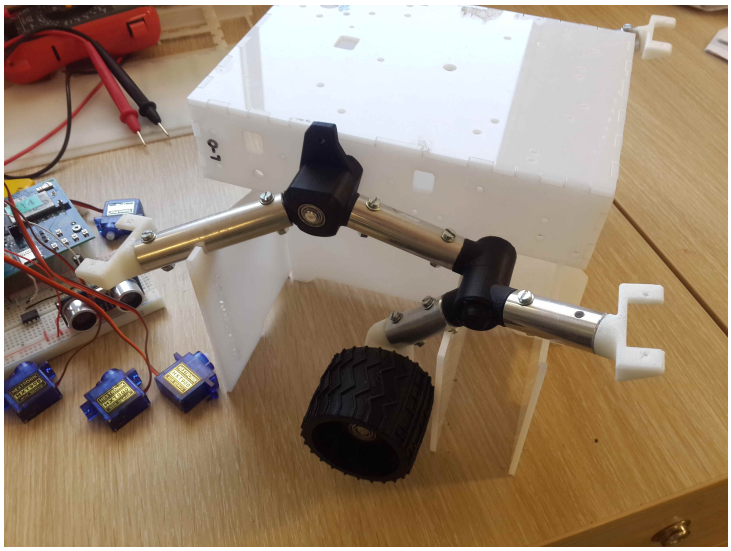


Figure 3.64: Image of one completed side of the suspension system before servo-related components were assembled.

Internal Electronics Mounting

As designed, the internal compartment of the body component of the rover was to be used for mounting of the electronic system. The design of the top panel of the body included holes aligned with mounting points on the Intel Edison board and these were used to fasten the board to the underside of the top panel. Below that, the Adafruit PWM extension module was fitted to the Intel Edison board and the length of the Intel Edison board mounts adjusted so that the rocker joint shaft, which passed through the width of the rover body, fitted between the board and the PWM extension module. The length adjustment also took into account the height of the servo cables plugged into the PWM module and the interference of those with the bottom cover of the body.

An acrylic mounting component, cut from the same acrylic sheet used for the body panels, was fastened to the PWM module to provide a mounting platform for the custom pulse to analog converter boards and the buck DC to DC converter module. the acrylic mounting component also provided insulation of the high power portion of the system from the Intel Edison board and PWM module.

Figure 3.65 shows the mounting and wiring plan used during the assembly of the rover and Figure 3.66 shows the completed electronics system as mounted.

Cables and Wiring As seen in Figure 3.66, the wiring was neatly packed into the area reserved as in Figure 3.65. Wiring included power cables from the batter to the buck DC to DC converter module and from the module to the Intel Edison board and PWM extension, servo cables, sensor cables and the web camera USB cable. The servo cables, sensor cables and USB cable were fed into the body of the rover through ports included in the acrylic panel design. Servo cables that travelled from the extremities of the suspension

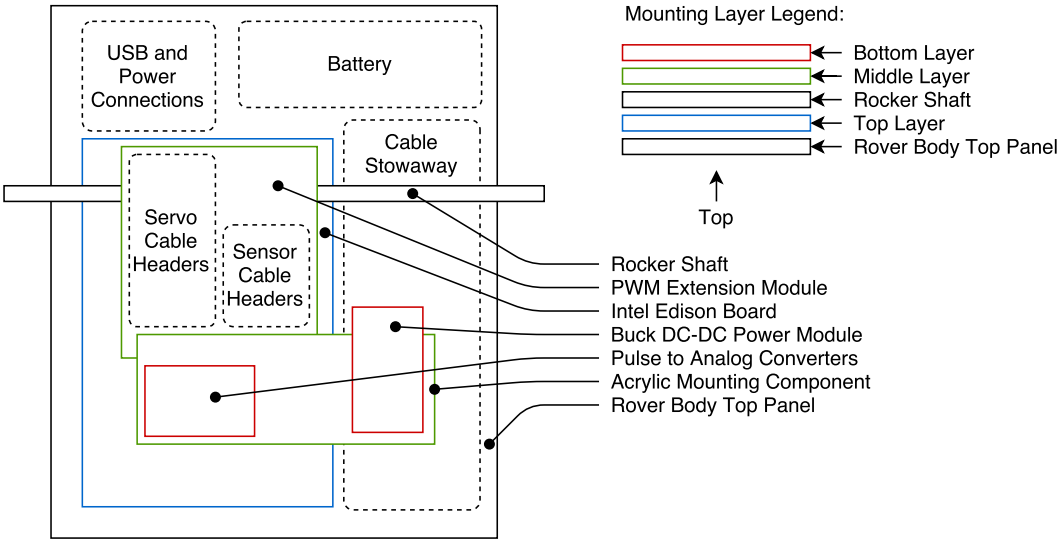


Figure 3.65: Diagram of the rover body internal electronics mounting plan as seen from below.

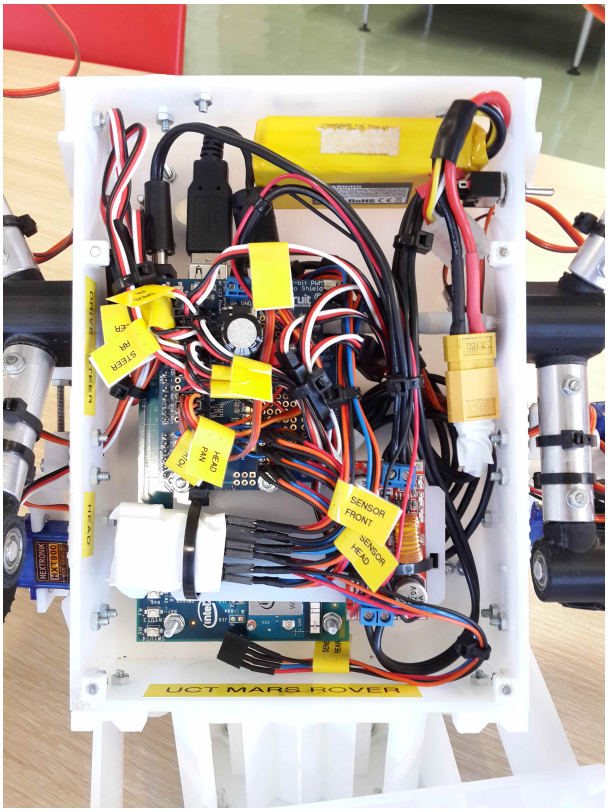


Figure 3.66: Image of the fully mounted electronics system including cabling and wiring.

system beams were tied to the aluminium tubing with enough slack to cater for the full range of motion of the joints and pivots. The servo and sensor cables were tied and labelled inside the body for easy identification for patching.

3.6 Software Development

Previously mentioned was the manner in which the software design and development workflow differed from those of the mechanical and electrical systems as well as the fact that the software system design was driven by the mechanical design and engineering choices within. For these reasons, many structural and functional design and development choices in the software system were made after the design. This section briefly covers those design choices as well as describes the environment in which the software was developed and deployed.

3.6.1 Development Environment

Project Structures

The three software components' development projects were kept as similar in structure as possible, which was achievable due to the use of JavaScript for all of them, two of which were Node.js applications. The file VCS used for the projects was Git with the forward looking intention of deploying the source on GitHub for open source collaboration. The common structure included the git-related files and the `package.json` file, a file used to describe the project for tools that required such information such as `npm` as discussed further on. In the case of the RSVP Client, another metadata file, `bower.json` was included as this file listed web module and library dependencies installed using Bower[1]. JavaScript source files were placed in a directory off the project root named `src`, and utility JavaScript code was placed in the `utils` or `resources` directory. In the case of the RSVP Client, an additional directory, `app`, contained the HTML and Polymer element source to be served to connected Clients.

The RSVP Server and the RCE both required an entry point which was to be invoked by the Node.js runtime. The entry point file, `index.js` was placed in the root of the `src` directory along with other base level files, such as system sequencers or modules responsible for handling communication. The entry point file also imported the files required by each system, which in turn imported required files and so forth.

Included in each of the projects was a configuration file, either of JSON or JavaScript format, in which static configuration data was stored and read into the software component during execution time. Configuration data included details such as the pin configuration for the PWM extension module or the frequency at which to sample the ADC pins for the proximity sensors. The configuration file made organisation of the data more readable and kept it in one place in the project.

Build Processes

All three of the software components in the software system required some form of build process to ensure that they were suitable for and compatible with the target environments.

[1]Bower: a package management tool used primarily for web-related modules and libraries - `https://bower.io/`

The `package.json` scripts were leveraged for the build processes and scripts were written for cleaning, building and watching (where appropriate). The `clean` script cleared out already built files and the `build` script performed the necessary steps as part of the build process to generate the required files, placing them in the `/build` folder. "Watching" invoked the `build` script but also indicated to the build tool to rebuild files as they were saved. This also applied to the remote building for the RCE and made the development flow more convenient. The three build processes are described below.

- RCE
  1. Clean working directory
  2. Transpile `ES6` code to Node.js `v6.8.0` compatible JavaScript
  3. Transmit transpiled code to the RCE board remotely
- RSVP Server
  1. Clean working directory
  2. Transpile `ES6` code to Node.js `v7.0.0` compatible JavaScript
  3. Output transpiled code to the `build` folder
- RSVP Client
  1. Clean working directories
  2. Transpile `ES6` Polymer element code into browser compatible JavaScript
  3. Transpile `ES6` source code into browser compatible JavaScript
  4. Bundle transpiled source code and dependent Node modules into files for browser import
  5. Output files into the directory accessible by the Server

Project Dependencies

Each project made use of multiple open source Node.js modules which were listed in each of the projects' `package.json` file. The dependencies are downloaded from remote repositories during the install process provided by the package manager that is bundled in the distribution of Node.js, called `npm` (Node.js Package Manager). These packages are listed in Appendix A together with a description of each. Towards the end of the project another dependency management tool, `yarn`[1], was used as it provided better caching and faster dependency resolution.

3.6.2 Johnny-Five Hardware Abstraction

A standard skeletal structure for each of the abstractions written for hardware sub-systems was developed to maintain the consistency of the APIs. Since the `edison-io` module, which bridged between the JavaScript `mraa` implementation and the J5 `Board` instance, was a singleton module, each abstraction file need not have imported a reference to

[1]Yarn Package Manager - `https://github.com/yarnpkg/yarn`

the already existing and running `Board`. Snippet 3.3 shows the structure of such an abstraction as well as serves as an example of the general structure of all JavaScript files written for all projects.

```
// Module and library imports
import debug from 'debug'; // Module to log debug statements to stdout
import * as five from 'johnny-five'; // The Johnny-Five library

import { config } from '../config'; // The global configuration file
import * as store from '../store'; // The data store module
import * as sequenceManager from '../sequences/sequence-manager'; // The
 ↪  sequence manager for firing system sequences

// Give the terminal logging a name
const log = debug('rce:<hardware-subsystem>');

/**
 * The hardware associated with this sub-abstraction
 * @type {Object}
 *
 * All hardware instances created from the J5 library are placed in the
 ↪  `hw` object, which is exposed for use by other parts of the application
 */
export let hw = {};

/**
 * Initialise the <hardware-subsystem>
 *
 * The init method is called by the startup sequence to initialise the
 ↪  hardware sub-system
 */
export function init() {
  hw = {
    // All hardware instances created here
  };

  // Save the new hardware state to the store which will notify the rest of
   ↪  the software components of the new state
  store.hardwareState.set('<hardware-subsystem>.initialised', true);
  log('<hardware-subsystem> initialised');
}

// All public methods are written here. A `start` and `stop` method is
 ↪  always included where applicable.

/**
 * Start the <hardware-subsystem>
 */
```

```
40 export function start() {
41   // Code to start the sub-system
42 }
43
44 /**
45  * Stop the <hardware-subsystem>
46  */
47 export function stop() {
48   // Code to stop the sub-system
49 }
50
51 // == Private ==
52 // All private methods are placed here. These methods are not exported and
   ↪  can only be used in this file.
```

Snippet 3.3: Example skeleton used for the hardware abstraction of each hardware sub-system (`<hardware-subsystem>`).

3.6.3 Control Implementation

The Section 3.4 has already covered the command translation and control pipeline. Details regarding implementation of the pipeline are omitted due to complexity of the software and can be seen in the source code provided with this report. This section briefly covers two of the significant control implementation details.

List of Commands

Several commands were implemented to be used for the control of the rover in both interactive and RoSE control modes. The scope for the commands that were possible to be implemented was wide. For time constraint reasons, only a few of the potential commands were implemented to serve as a proof of concept of the command architecture and supporting sub-system. Below is a list of the commands implemented followed by an example of one of the commands as implemented in code in Snippet 3.4.

- **Pause Command:** Commands the rover to "do nothing" for a specified length of time.
  - **Type:** Low
  - **Parameters:**
    - **Duration:** The length of time to pause for.
- **Single Wheel Rotate Command:** Commands the rover to rotate one of the wheel struts about its pivot by a specified angle.
  - **Type:** Low
  - **Parameters:**
    - **Wheel:** The wheel to rotate.

    - **Angle:** The angle by which to rotate the wheel.
    - **Velocity:** The velocity by which to effect the rotation motion.

- **Single Wheel Drive Command:** Commands the rover to drive a single wheel for a specified duration at a specified velocity.
    - **Type:** Low
    - **Parameters:**
        - **Duration:** The length of time to drive the wheel for.
        - **Wheel:** The wheel to drive.
        - **Velocity:** The velocity at which the wheel should drive.
        - **Direction:** The drive direction (forwards or reverse).
- **Drive Command:** Commands the rover to traverse forwards or backwards at a specified velocity with a specified steering arc.
    - **Type:** High
    - **Parameters:**
        - **Duration:** The length of time to traverse.
        - **Velocity:** The velocity at which to traverse.
        - **Direction:** The direction in which to traverse (forwards or reverse).
        - **Arc:** The factor of the maximum rover turning circle about which to traverse.
- **Wheels Rotate Command:** Commands the rover to rotate the wheels in terms of a specified steering arc factor.
    - **Type:** High
    - **Parameters:**
        - **Arc:** The factor of the maximum rover turning circle.
        - **Velocity:** The velocity by which to effect the rotation of the wheels.
- **Rover Rotate Command:** Commands the rover to rotate about its centre point.
    - **Type:** Macro
    - **Parameters:**
        - **Duration:** The length of time to rotate for.
        - **Velocity:** The velocity at which to rotate.
        - **Direction:** The direction in which to rotate (clockwise or counter-clockwise).
- **Head Pan Command:** Commands the rover to position the head at a certain angle about the pan-axis.
    - **Type:** Low
    - **Parameters:**
        - **Angle:** The angle to which the head should rotate.
        - **Velocity:** The velocity at which to rotate the head.
- **Head Pitch Command:** Commands the rover to position the head at a certain angle about the pitch-axis.
    - **Type:** Low

- **Parameters:**
  - **Angle:** The angle to which the head should rotate.
  - **Velocity:** The velocity at which to rotate the head.

- **Head Position Command:** Commands the rover to position the head given pan and pitch angles.
  - **Type:** Low
  - **Parameters:**
    - **Pan Angle:** The angle to which the head should rotate about the pan-axis.
    - **Pitch Angle:** The angle to which the head should rotate about the pitch-axis.
    - **Velocity:** The velocity at which to rotate the head.

```
/**
 * The base class for a command. Each command class will extended this class adding parameters specific to the command
 */
export class SeqCmd {
  constructor(name, type, params = {}) {
    this.name = name; // Name of the commands
    this._name = this.constructor.name; // Name of the constructor (which sometimes differed from the command name) used for creating copies of the command throughout the control flow
    this.type = type; // The command level ['low'|'high'|'macro']
    this.params = params; // The command parameters
  }
}

/**
 * Command the rover to "do nothing" for a specified duration
 */
export class PauseCmd extends SeqCmd {
  constructor(params = {}) {
    super('Pause', 'low'); // Call the superclass's constructor with the name and type of the command

    this.params = {
      // A parameter with the standard parameter properties understood by the control pipeline
      duration: {
        type: 'Number', // Data type
        unit: 'sec', // Parameter unit
        icon: 'rsvp:access-time', // The icon to use in the RSVP Client
        value: params.duration || null, // The actual parameter value
      },
    };
  }
```

```
30 }
```

Snippet 3.4: An example definition of a command class including the super class from which the definition inherited command properties

Traversal Servo Control

The position of the wheels on the rover meant that traversal of the rover along arced paths required the wheels to be at the correct angles and to be operating at differing angular velocities in order to prevent under- or over-rotation. Turning to the left along a gradual arc, for example, required the left side wheels to operate slower than the right side wheels as well as the left side wheels to be angled more than the right. This was simply because the arc radius of the inside wheels was smaller than that of the outside wheels making the tangents different as well as the perimeters different and thus the velocity required to traverse the perimeters differed.

Commands that required this type of motion or traversal calculated the angles and velocities based on the measured physical position of each wheel. Included in the calculations were two simplifying assumptions: that the angle passed to J5 in the hardware abstraction was exactly representative of the output angle of the steering servos, and that the continuous servo velocity versus J5 input velocity parameter was a linear relationship. The latter assumption was tested as well as justified by the principle of operation of the servos. The continuous servo modification meant that the motor speed of the servo device was controlled by means of the input PWM signal. Figure 3.67 shows the wheel layout and the arc lines used to formulate the relationships between the arc factor and the resulting wheels angles and velocities.

The first angle, the angle of the front-inner wheel (relative to the direction of the arc; the closest side to the arc centre-point), was calculated by (3.1) since it was known that the inner wheels would exhibit the greatest angular displacement.

$$[\theta_{\text{wheel}}]_{\text{inside}} = a \times [\theta_{\text{wheel}}]_{\text{max}} \tag{3.1}$$

where θ_{wheel} is the angle of a wheel and a is the arc factor.

It was decided that the amount by which the centre wheels were offset from the centre of the rover, in the x direction, was negligible (i.e. the wheels on each side of the suspension system were equally distributed) and so the rear wheels' angles were simply the same as those of the front wheels but in the opposite direction.

Using $[\theta_{\text{wheel}}]_{\text{inside}}$, the radius of the circle formed by the inner-side wheels was calculated by (3.2) and used to find the position of the centre-point of the circle which was representative of the centre-point of the arc of traversal of the rover. The circle made by the outer wheels shared this centre-point and the distance between the front wheels of each side of the

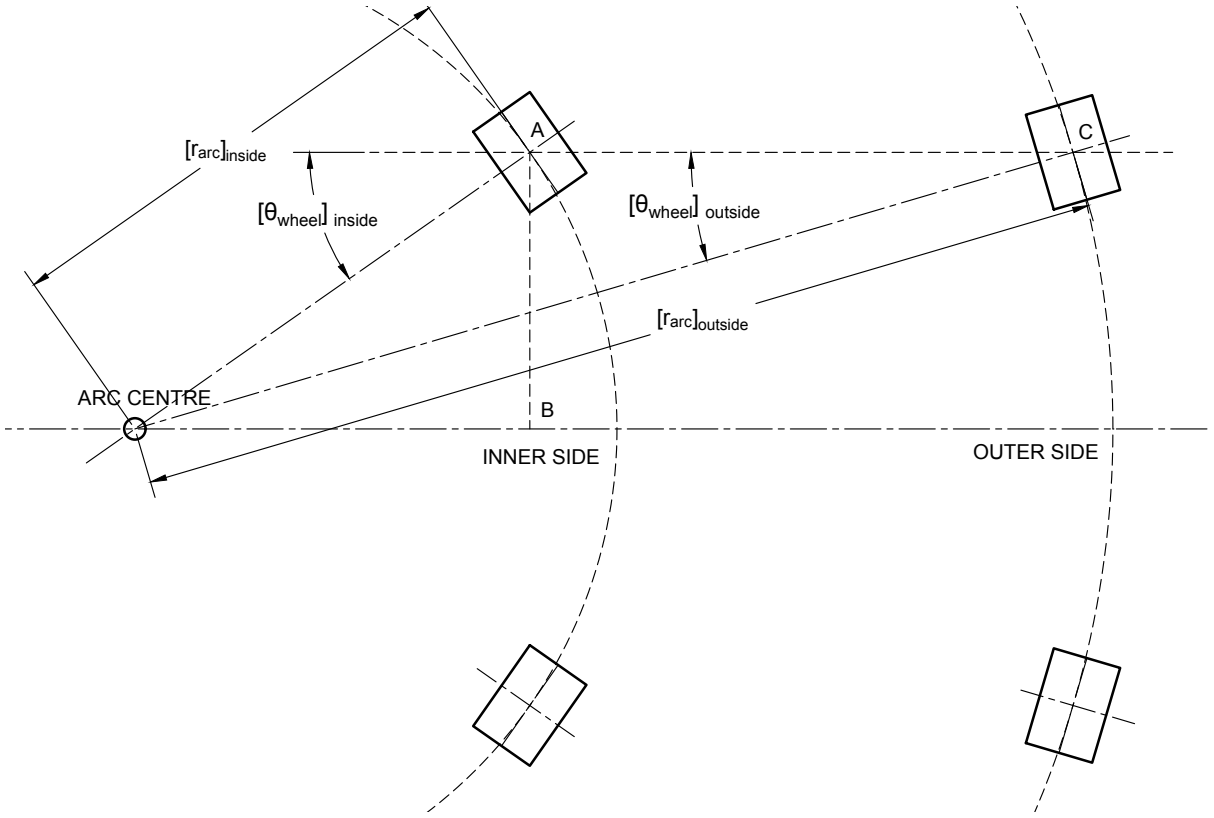


Figure 3.67: Drawing of the front and rear wheels and the associated traversal arcs.

suspension system was used to calculate the radius of the outer circle in (3.3). Therefore, since the outer circle intersected with the outer wheels, the tangent at the points of intersection could be found by (3.4) and this was the required angle of the outer wheels.

$$[r_{\text{arc}}]_{\text{inside}} = \frac{\text{AB}}{\sin\left([\theta_{\text{wheel}}]_{\text{inside}}\right)} \tag{3.2}$$

where $[r_{\text{arc}}]_{\text{inside}}$ is the radius of the circle made by the inner side wheels.

$$[r_{\text{arc}}]_{\text{outside}} = [r_{\text{arc}}]_{\text{inside}} + \text{AC} \tag{3.3}$$

where $[r_{\text{arc}}]_{\text{outside}}$ is the radius of the circle made by the outer side wheels.

$$[\theta_{\text{wheel}}]_{\text{outside}} = \arcsin\left(\frac{\text{AB}}{[r_{\text{arc}}]_{\text{outside}}}\right) \tag{3.4}$$

The velocities were simply calculated by applying the velocity factor to the outer side wheels due to the fact that they were to traverse further, and scaling the factor down

by the ratio of the two circle radii for the inner side wheels. The method used in the dispatch stage to perform the above calculations is included as Snippet 3.5.

```
/**
 * Compute the arc dimensions of the wheels for a given arc factor
 * @param  {Number} arcFactor The factor by which the rover should turn/arc
 * @return {Object}           An object of the calculated dimensions
 */
function _computeArc(arcFactor) {
  const largeSide = (arcFactor < 0) ? 'Right' : 'Left';
  const smallSide = (largeSide === 'Right') ? 'Left' : 'Right';
  const smallAngle = arcFactor * 45;
  const smallRadius = config.hardware.wheelPitch /
   ↪  Math.sin(deg2rad(Math.abs(smallAngle)));
  const largeRadius = config.hardware.wheelSpan + smallRadius;
  const largeAngle = (arcFactor < 0 ? -1 : 1) *
   ↪  rad2deg(Math.asin(config.hardware.wheelPitch / largeRadius));

  return {
    largeSide,
    smallSide,
    smallAngle,
    smallRadius,
    largeRadius,
    largeAngle,
  };
}
```

Snippet 3.5: Method used to perform the arc calculations given a traversal arc factor.

3.6.4 Socket.io Channels

The Socket.io channels used throughout the project is summarised in Table 3.11 based on discussions thereof in the software design sections.

Each Socket.io endpoint component in the endpoint-translator pair included in initialisation method which registered the WebSocket channel and attached the required listeners to the instance. The listeners that were registered referenced handler methods exposed by the translator component keeping the endpoint component simple. The endpoint component also provided lower level methods for performing operations such as sending custom requests. Snippet 3.6 includes an example endpoint implementation structure.

```
import * as rceIOTranslator from './<channel>-translator'; // The
 ↪  translator associated with the socket channel
import Socket from 'koa-socket'; // The Node.js module providing the
 ↪  Socket.io implementation for the server stack used in this particular
 ↪  software component
```

Socket.io Channel	Applicable Software Components	Description	Channel Data Responsibilities
RCEIO	RCE RSVP Server	Serves as the only method of message data type communication between the RCE and the RSVP Server.	- Transmission of commands - Transmission of telemetry - Synchronisation of data stores - Transport of custom requests
KurentoIO	RSVP Server RSVP Client	The signalling between the implementation of Kurento on the RSVP server and those on the connected RSVP Clients. Servers as the transport for SDP and ICE offers as well as other signalling events between the two endpoints.	- Transmission of SDP offers - Transmission of ICE candidates
ControlIO	RSVP Server RSVP Client	The first of the two message data specific channels between the RSVP Server and the RSVP Client. Handles control data and is restricted to only the client that is in control of the rover.	- Transmission of RoSE commands - Transmission of interactive control signals - Transmission of other control related events and messages
TeleIO	RSVP Server RSVP Client	The second channel between the RSVP Server and the RSVP Client facilitating all types of telemetry.	- Transmission of telemetry data - Synchronisation of data stores

Table 3.11: Table summarising the responsibilities of each of the implemented Socket.io channels.

```
3
4  const log = debug('rce:socket');
5
6  let <channel>;
7
8  /**
9   * Initialise the socket instance and attach the associated handlers to it
10  */
11 export default function init() {
12   // Instantiate the socket
13   <channel> = new Socket('<channel>');
```

```
14
15    // Listen on the `connection` event to detect a successful connection
16    socket.on('connection', () => {
17      store.rceState.set('<channel>.connected', true); // Update the store to
        ↪  reflect the new state of the connection
18    });
19
20    // Register the listeners associated with the channel, passing in
      ↪  references to the translator methods
21    socket.on('data', <channel>Translator.onData);
22    socket.on('post', <channel>Translator.onPost);
23    socket.on('request', <channel>Translator.onRequest);
24  }
25
26  /**
27   * Send a `request` type message via the socket, with a given payload
28   * @param^^I{String}^^Itype^^I^^IThe type of request
29   * @param {Object}^^Ipayload^^IPayload to include with the message
30   *
31   * This is an example of a custom low level operation included in the
     ↪  endpoint API
32   */
33  export function sendRequest(type, payload) {
34    <channel>.broadcast('request', { type, payload });
35  }
```

Snippet 3.6: An example Socket.io endpoint file where `<channel>` is the name of the channel.

3.6.5 RCE Video Streaming Utility

A candidate solution to the video streaming on the RCE software component was to use the Linux tool `Video4Linux2` (`v4l2`) which allowed streaming of video in realtime from hardware available to the operating system, including devices of the UVC driver-less class. Another tool, `ffmpeg`, which leveraged `v4l2` for video capture, allowed the creation of an HTTP and RTSP server which could be connected to from an external endpoint on the network. However, tests performed with an `ffmepg` implementation proved the tool unsuitable for the project due to low reliability and high CPU cost. Yet another utility was found, `mjpeg-streamer`, which was also tested and found to be a well performing and reliable alternative to `ffmpeg`. `mjpeg-streamer` provided streaming of the video data in one command line (Snippet 3.7), which was spawned as a child process by the RCE as in Snippet 3.8.

```
1  '/usr/local/bin/mjpg_streamer -i "/usr/local/lib/input_uvc.so -d
   ↪  /dev/video0 -n -r 640×480 -f 30" -o "/usr/local/lib//output_http.so -n
   ↪  -p 8080 -w /usr/local/www"'
```

Snippet 3.7: The `bash` command which was executed to invoke `mjpeg-streamer` and stream the video data via the specified port.

```
import child from 'child_process'; // A Node.js provide API for spawning
↪  additional processes outside that of the Node.js program

// Spawn the camera process
camProcess = child.exec(config.hardware.cameraStartCmdLine, (err) ⇒ {
  // Errors handled here
});
```

Snippet 3.8: Snippet showing the spawning of a child process in which `mjpeg-streamer` was invoked.

Chapter 4

Rover Post-Development

4.1 Final Product

Figures ![] and ![] show the fully assembled rover model taken before the testing phase of the project.

![Rover full image] ![Rover top and side images]

Figures ![], ![] and ![] show the completed RSVP Client user interface in each of the two control modes.

4.2 Testing

Before verification of the project specifications, the rover model and software systems were analysed in a series of test scenarios. All systems were kept online during the tests (including the video stream) and the results documented. The tests were filmed and the results video is included in the report submission.

Due to lack of a Mars-like environment or terrain surface, the test on rough terrain was omitted. This type of testing as well as research and development of a Martian terrain simulation environment is mentioned as a possible future work in Section ![].

TEST-1: **Stationary Test:** The first test performed which involved placing the rover in the designed and assembled stand. The test included performing the self-diagnostics test sequence and using the interactive and RoSE control styles to verify the successful operation of the rover without the presence of terrain. Figure ![] shows the rover model positioned in the stand.

![Stand image]

TEST-2: **Smooth Surface Collision Avoidance Test:** The rover was then placed on a smooth office-floor surface and commanded to traverse in a straight line towards a

large obstacle. The test verified the rovers ability to perform rudimentary obstacle avoidance and prevent control in the presence of an obstacle larger than the rover's traversal capabilities.

TEST-3: **Smooth Surface Single-side Obstacle Traversal Test:** In this test, the rover was commanded to traverse a smooth surface with an A-framed shaped obstacle in an attempt to replicate a similar test performed on *Curiosity*. The test verified the performance and effectiveness of the rocker-bogie suspension in minimising the body tilt and rock during traversal of uneven ground. Figure ![] shows the rover traversing the obstacle.

![Obstacle image]

TEST-4: **RSVP Server Load Test:** The final test, in which a typical number of RSVP Clients were instantiated, connected and allowed to stream the video was performed to verify the RSVP Server's ability to handle a typical Client congestion situation.

4.3 Post-development Verification of Specifications

Having performed the aforementioned tests on the rover model and the complete software system, a full post-development verification was performed in which each of the requirements as outlined in Section 3.1.5 were analysed against the final product. The analysis aimed to determine if each requirement was satisfied and this was used as a platform for discussion on the entire design, development and the project in general (thus justifying the lack of a "Discussions" section in this report).

- **Mechanical**

RM-1: General Specifications:

RM-1.1: **Partially Satisfied**
All components of the vehicle were proportional to the 3D reference model [49] provided by NASA except in the mast and head assembly, whereby the servo and camera dimensions did not allow for smaller components, and in the beams of the suspension system.
Proposed improvements:
- Sourcing of a smaller camera module would allow for a smaller camera head assembly together with better planning of the mounting of the camera inside of the head cavity.
- A smaller ultrasonic proximity sensor would alleviate the requirement for the extension of the head canopy for mounting purposes.

RM-2: Body:

RM-2.1: **Fully Satisfied**
RM-2.2: **Fully Satisfied**
RM-2.3: **Fully Satisfied**
RM-2.4: **Fully Satisfied**
RM-2.5: **Fully Satisfied**
RM-2.6: **Fully Satisfied**
Mounting of the DC to DC Converter module and the pulse to analog converters was moved from the body to an additional acrylic piece fastened to the PWM extension module.
RM-2.7: **Fully Satisfied**

RM-3: Mast:

RM-3.1: **Fully Satisfied**
RM-3.2: **Partially Satisfied**
The range of actuation in the servo components chosen for the panning axis rotation of the head was limited to 180°.
Proposed Improvements:
- Make use of a servo component capable of offering full rotation. This type of servo was not available at the time of design and development of the project.

RM-3.3: **Fully Satisfied**
RM-3.4: **Partially Satisfied**
Using the servo shaft for mounting reduced the structural stability of the

assembly due to play in the plastic gears of the component. While the rigidity of the mast and head assembly was well within that which was required for operation, improved rigidity would have been possible with an alternative mounting configuration and/or use of metal gear servos. This is discussed further in Section 6.2.1.

RM-4: Head:

RM-4.1: **Fully Satisfied**
RM-4.2: **Fully Satisfied**
RM-4.3: **Fully Satisfied**

RM-5: Suspension:

RM-5.1: **Fully Satisfied** as tested in TEST-3.
RM-5.2: **Fully Satisfied** as tested in TEST-3.

RM-6: Wheels and Hubs/Pivots:

RM-6.1: **Fully Satisfied**
RM-6.2: **Fully Satisfied** While the traction capability of the designed wheels was not tested due to lack of a terrain, the wheels replicated the traction pattern on the tires of *Curiosity*.
RM-6.3: **Fully Satisfied** as tested in TEST-2.

- **Electrical**

RE-1: Actuation:

RE-1.1: **Fully Satisfied** as tested in TEST-1 and TEST-3.
RE-1.2: **Fully Satisfied** as tested in TEST-3.
RE-1.3: **Fully Satisfied**

RE-2: Central Control:

RE-2.1: **Fully Satisfied**
RE-2.2: **Fully Satisfied**
RE-2.3: **Fully Satisfied** as partially tested in TEST-2.
RE-2.4: **Fully Satisfied**

Note that the choice of the Intel Edison was suitable for the designed rover, however, the severely limited control of the GPIO pins and other peripherals such as PWM could have been avoided if another device was chosen. This is further discussed in Section 6.2.2.

RE-3: Power:

RE-3.1: **Fully Satisfied**
RE-3.2: **Fully Satisfied**
RE-3.3: **Fully Satisfied**
RE-3.4: **Fully Satisfied**
RE-3.5: **Fully Satisfied**
RE-3.6: **Fully Satisfied**

RE-4: Sensors:

RE-4.1: **Partially Satisfied**
Due to the limitations in the Intel Edison's ability to measure input electrical pulses, the pulse to analog conversion solution introduced a

significant increase in the response time of the distance measurements. This meant that while satisfactory data was acquired, it was not immediate and thus affected the speed of obstacle detection.

Proposed Improvements:

- Source the I^2C backpack designed to allow the Intel Edison to correctly interface with the HC-SR04 Sensors.
- Source digital proximity sensors or range-finders.

RE-4.2: **Fully Satisfied**

RE-4.3: **Not Satisfied** as discussed in the analysis of RE-4.1.

RE-5: Camera:

RE-5.1: **Fully Satisfied**

RE-5.2: **Partially Satisfied**

Post-manufacture modifications to the head canopy printed part had to be made to allow mounting of the camera. Due to the time-scale of the project, ordering of components and design of the components within the mechanical system had to occur simultaneously. The dimensions of the web camera module were unknown during the design of the mast assembly.

RE-5.3: **Fully Satisfied**

Software System Specifications

- **Rover Embedded Software**

RS-1: General Specifications:

RS-1.1: **Fully Satisfied**

RS-1.2: **Fully Satisfied**

RS-2: Control:

RS-2.1: **Fully Satisfied**

RS-2.2: **Fully Satisfied**

RS-2.3: **Fully Satisfied**

RS-3: Telemetry:

RS-3.1: **Fully Satisfied**

RS-4: Video Stream:

RS-4.1: **Fully Satisfied**

RS-4.2: **Fully Satisfied**

- **Server**

RS-5: General Requirements:

RS-5.1: **Fully Satisfied**

RS-5.2: **Fully Satisfied**

RS-5.3: **Fully Satisfied**

RS-5.4: **Fully Satisfied**

RS-6: Video Broadcast:

RS-6.1: **Fully Satisfied**

RS-6.2: **Fully Satisfied** as tested in TEST-4.

RS-6.3: **Fully Satisfied**

RS-7: Data Relay:

RS-7.1: **Fully Satisfied**

RS-7.2: **Partially Satisfied** The means by which long distance communication was simulated was rudimentary in that it consisted only of a time delay below 60 seconds. This was not an accurate depiction of the communication dynamic between Earth and *Curiosity*, however, such a simulation would have taken away from the experience of the user.

RS-7.3: **Fully Satisfied**

RS-7.4: **Fully Satisfied**

- **Client**

RS-8: General Requirements:

RS-8.1: ![Fill out]

RS-9: Control:

RS-9.1: **Fully Satisfied**

RS-10: Telemetry:

RS-10.1: **Fully Satisfied**

RS-11: Video Feed:

RS-11.1: **Fully Satisfied**

Chapter 5

Conclusions

These are the conclusions from the investivation and how the investigation changes things in this field or contributes to current knowledge...

Draw suitable and intelligent conclusions from your results and subsequent discussion.

Chapter 6

Recommendations

6.1 Mechanical Recomendations

6.2 Electrical Recommendations

6.2.1 Suitability of Servos for Shaft Mounting

6.2.2 Choice of RCE Board

6.3 Software Recommendations

Bibliography

[1] E. Kamen and B. Heck, *Fundamentals of Signals and Systems Using the Web and MATLAB*. Prentice Hall, 2000.

[2] J. Cornwell, *Hitler's scientists : science, war, and the devil's pact*. New York: Viking, 2003.

[3] J. Schefter, *The race : the uncensored story of how America beat Russia to the moon*. New York: Doubleday, 1999.

[4] W. Harwood, "Sts-129/iss-ulf3 quick-look data," Oct 2009. [Online]. Available: http://www.cbsnews.com/network/news/space/129/129quicklook2.pdf [Accessed: 2016-07-23]

[5] R. Simberg, "Elon musk on spacex's reusable rocket plans," February 2012. [Online]. Available: http://www.popularmechanics.com/space/rockets/a7446/elon-musk-on-spacexs-reusable-rocket-plans-6653023/ [Accessed: 2016-07-23]

[6] W. Compton and C. Benson, *Living and working in space: a history of Skylab*, ser. NASA SP. Scientific and Technical Information Branch, National Aeronautics and Space Administration, 1983. [Online]. Available: https://books.google.co.za/books?id=Qpax6JcVyG8C [Accessed: 2016-07-23]

[7] NASA, "Fields of research," March 2008. [Online]. Available: https://web.archive.org/web/20080123150641/http://pdlprod3.hosc.msfc.nasa.gov/A-fieldsresearch/index.html [Accessed: 2016-07-23]

[8] N. Aeronautics and S. Administration, *Authorization Act of 2010*, 2010.

[9] S. Robbins, "Journey through the galaxy: Mars program," 2008. [Online]. Available: http://jtgnew.sjrdesign.net/exploration_space_planetary_mars.html [Accessed: 2016-07-23]

[10] J. Nelson, "Mars pathfinder / sojourner rover," 2000. [Online]. Available: http://www.jpl.nasa.gov/missions/details.php?id=5913 [Accessed: 2016-07-23]

[11] JPL, "Nasa's mars exploration program's science theme." [Online]. Available: http://mars.jpl.nasa.gov/programmissions/science/ [Accessed: 2016-07-25]

[12] ——, "Cruise configuration." [Online]. Available: http://mars.nasa.gov/msl/mission/spacecraft/cruiseconfig/ [Accessed: 2016-07-25]

[13] W. Harwood, "Mars science laboratory begins cruise to red planet," November 2011. [Online]. Available: http://spaceflightnow.com/atlas/av028/ [Accessed: 2016-07-25]

[14] T. J. Martin-Mur, G. L. Kruizinga, P. D. Burkhart, M. C. Wong, and F. Abilleira, "Mars science laboratory navigation results," Jet Propulsion Laboratory, California Institute of Technology, 4800 Oak Grove Drive, Pasadena, CA 91109, USA, Tech. Rep., 2012. [Online]. Available: http://issfd.org/ISSFD_2012/ISSFD23_IN1_1.pdf [Accessed: 2016-07-25]

[15] JPL, "Mars science laboratory: Mission objectives." [Online]. Available: http://mars.jpl.nasa.gov/msl/mission/science/objectives/ [Accessed: 2016-07-25]

[16] E. Lakdawalla, "Curiosity in context: Not exactly "Viking on wheels," but close," November 2011. [Online]. Available: http://www.planetary.org/blogs/emily-lakdawalla/2011/3271.html [Accessed: 2016-07-30]

[17] N. Aeronautics and S. Administration, *Mars Science Laboratory Landing*, 2012. [Online]. Available: http://www.jpl.nasa.gov/news/press_kits/MSLLanding.pdf [Accessed: 2016-07-31]

[18] E. Lakdawalla, "Curiosity wheel damage: The problem and solutions," August 2014. [Online]. Available: http://www.planetary.org/blogs/emily-lakdawalla/2014/08190630-curiosity-wheel-damage.html [Accessed: 2016-08-03]

[19] JPL, "Mars Science Laboratory: Curiosity wheels and legs." [Online]. Available: http://mars.jpl.nasa.gov/msl/mission/rover/wheelslegs/ [Accessed: 2016-08-14]

[20] J. Hanna, "'impressive' Curiosity landing only 1.5 miles off, NASA says," August 2012. [Online]. Available: http://edition.cnn.com/2012/08/10/us/mars-curiosity/index.html?eref=mrss_igoogle_cnn [Accessed: 2016-08-14]

[21] W. Wong, "VxWorks goes 64-bit," March 2011. [Online]. Available: http://electronicdesign.com/embedded/vxworks-goes-64-bit [Accessed: 2016-08-21]

[22] S. Anthony, "Inside nasa's Curiosity: It's an Apple Airport Extreme… with wheels," August 2012. [Online]. Available: http://www.extremetech.com/extreme/134041-inside-nasas-curiosity-its-an-apple-airport-extreme-with-wheels [Accessed: 2016-08-21]

[23] NASA/JPL-Caltech, "Pia16106: Curiosity speaks and orbiters listen." [Online]. Available: http://photojournal.jpl.nasa.gov/catalog/PIA16106 [Accessed: 2016-08-21]

[24] JPL, "Dsn: About." [Online]. Available: http://deepspace.jpl.nasa.gov/about/# [Accessed: 2016-08-21]

[25] A. Makovsky, P. Ilott, and J. Taylor, "Mars science laboratory telecommunications system design," Jet Propulsion Laboratory, Pasadena, California, Tech. Rep., 11 2009. [Online]. Available: http://descanso.jpl.nasa.gov/DPSummary/Descanso14_MSL_Telecom.pdf [Accessed: 2016-08-21]

[26] JPL, "MSL science corner: Mast camera (mastcam)." [Online]. Available: http://msl-scicorner.jpl.nasa.gov/Instruments/Mastcam/ [Accessed: 2016-08-21]

[27] ——, "MSL science corner: Chemistry and camera (chemcam)." [Online]. Available: http://msl-scicorner.jpl.nasa.gov/Instruments/ChemCam/ [Accessed: 2016-08-21]

[28] ——, "MSL science corner: Alpha particle x-ray spectrometer (apxs)." [Online]. Available: http://msl-scicorner.jpl.nasa.gov/Instruments/APXS/ [Accessed: 2016-08-21]

[29] ——, "MSL science corner: Mars hand lens imager (mahli)." [Online]. Available: http://msl-scicorner.jpl.nasa.gov/Instruments/MAHLI/ [Accessed: 2016-08-21]

[30] ——, "MSL science corner: Chemistry and mineralogy (CheMin)." [Online]. Available: http://msl-scicorner.jpl.nasa.gov/Instruments/MAHLI/ [Accessed: 2016-08-21]

[31] ——, "MSL science corner: Sample analysis at mars (sam)." [Online]. Available: http://msl-scicorner.jpl.nasa.gov/Instruments/SAM/ [Accessed: 2016-08-21]

[32] ——, "MSL science corner: Radiation assessment detector (rad)." [Online]. Available: http://msl-scicorner.jpl.nasa.gov/Instruments/RAD/ [Accessed: 2016-08-21]

[33] ——, "MSL science corner: Dynamic albedo of neutrons (dan)." [Online]. Available: http://msl-scicorner.jpl.nasa.gov/Instruments/DAN/ [Accessed: 2016-08-21]

[34] I. N. Laboratory, "Fueling the Mars Science Laboratory." [Online]. Available: http://www4vip.inl.gov/research/mars-science-laboratory/d/mars-science-laboratory.pdf [Accessed: 2016-08-21]

[35] S. R. P. Systems, "Multi-mission radioisotope thermoelectric generator," January 2008. [Online]. Available: http://mars.jpl.nasa.gov/msl//files/mep/MMRTG_Jan2008.pdf [Accessed: 2016-08-21]

[36] B. Cooper, "Driving on the surface of mars using the rover control workstation," *Proceedings of SpaceOps 1998*, 1998.

[37] J. Wright, F. Hartman, B. Cooper, S. Maxwell, J. Yen, and J. Morrison, "Driving on mars with rsvp," *IEEE Robotics and Automation Magazine*, vol. 13, no. 2, pp. 37–45, 2006.

[38] J. R. Wright, F. Hartman, S. Maxwell, B. Cooper, and J. Yen, "Updates to the rover driving tools for curiosity," in *System of Systems Engineering (SoSE), 2013 8th International Conference on*, June 2013, pp. 147–152.

[39] JPL, "Robot sequencing and visualization program (rsvp)," Jet Propulsion Laboratory, Pasadena, California, Tech. Rep., 9 2013. [Online]. Available: http://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/20140001459.pdf [Accessed: 2016-08-21]

[40] ——, "Rose command editor (blurring due to itar restrictions)." [Online]. Available: http://www.linuxjournal.com/files/linuxjournal.com/linuxjournal/articles/075/7570/7570f1.jpg [Accessed: 2016-08-21]

[41] ——, "A view of the rover and terrain in hyperdrive." [Online]. Available: https://www-robotics.jpl.nasa.gov/images/RSVP-cropped.jpg [Accessed: 2016-08-21]

[42] MIT, "Mit to make nearly all course materials available free on the world wide web," April 2001. [Online]. Available: http://news.mit.edu/2001/ocw [Accessed: 2016-08-22]

[43] A. Monago, "The culture of open source," 2014. [Online]. Available: https://www.thoughtworks.com/insights/blog/culture-open-source [Accessed: 2016-8-22]

[44] jyoung, "Carbon fiber vs fiberglass." [Online]. Available: http://blog.fibreglast.com/fiberglass/carbon-fiber-vs-fiberglass/ [Accessed: 2016-09-10]

[45] Tilzor, "Library 42- control rc servo with stm32f4." [Online]. Available: http://stm32f4-discovery.net/2014/10/library-42-control-rc-servo-stm32f4/ [Accessed: 2016-09-20]

[46] Mouser, "Beaglebone black vs. beaglebone green vs. beaglebone green wireless." [Online]. Available: http://www.mouser.co.za/new/seeedstudio/beaglebone-black-vs-green/ [Accessed: 2016-09-15]

[47] B. Kumanchik and NASA/JPL-Caltech, "Curiosity (clean)." [Online]. Available: http://nasa3d.arc.nasa.gov/detail/curiosity-clean [Accessed: 2016-09-20]

[48] K. Novak, S. Hendricks, C.-J. Lee, and C. Orrala, "Mars science laboratory: Rover actuator thermal design," March 2008. [Online]. Available: http://trs-new.jpl.nasa.gov/dspace/bitstream/2014/41421/1/08-0699.pdf [Accessed: 2016-09-22]

[49] N. J. P. Laboratory, "Curiosity rover," August 2015. [Online]. Available: http://nasa3d.arc.nasa.gov/detail/mars-rover-curiosity [Accessed: 2016-09-22]

[50] JPL, "Msl mobility assembly lift mishap." [Online]. Available: http://llis.nasa.gov/lesson/6216 [Accessed: 2016-09-23]

[51] M. Lalwani, "Nasa is building the next mars rover in mixed reality." [Online]. Available: https://www.engadget.com/2016/05/23/nasa-hololens-mars-rover-in-mixed-reality/ [Accessed: 2016-09-23]

[52] MakerBot, "Pla and abs strength data." [Online]. Available: https://eu.makerbot.com/fileadmin/Inhalte/Support/Datenblatt/MakerBot_R___PLA_and_ABS_Strength_Data.pdf [Accessed: 2016-09-24]

[53] W3Techs, "Usage of operating systems for websites," October 2016. [Online]. Available: https://w3techs.com/technologies/overview/operating_system/all [Accessed: 2016-10-09]

[54] C. Curran, "Projects, applications, and companies using node," January 2016. [Online]. Available: https://github.com/nodejs/node/wiki/Projects,-Applications,-and-Companies-Using-Node [Accessed: 2016-10-09]

[55] D. Buytaert, "A history of javascript across the stack," February 2016. [Online]. Available: http://buytaert.net/a-history-of-javascript-across-the-stack [Accessed: 2016-10-09]

[56] The Linux Foundation, "2016 user survey report," April 2016. [Online]. Available: https://nodejs.org/static/documents/2016-survey-report.pdf [Accessed: 2016-10-10]

[57] E. Obrezkov, "Why nodejs is so fast?" [Online]. Available: https://blog.ghaiklor.com/why-nodejs-is-so-fast-a0ff67858f48#.gsdpyq91f [Accessed: 2016-10-10]

[58] A. Raoof, "Understanding the node.js event loop." [Online]. Available: http://abdelraoof.com/blog/2015/10/28/understanding-nodejs-event-loop/ [Accessed: 2016-10-10]

[59] Node.js, "Node.js helps nasa keep astronauts safe and data accessible." [Online]. Available: https://nodejs.org/static/documents/casestudies/Node_CaseStudy_Nasa_FNL.pdf [Accessed: 2016-10-10]

[60] Intel, "Schematics for the intel® edison board for arduino." [Online]. Available: http://www.intel.com/content/www/us/en/support/boards-and-kits/000005829.html [Accessed: 2016-10-10]

[61] Kernel.org, "Gpio sysfs interface for userspace." [Online]. Available: https://www.kernel.org/doc/Documentation/gpio/sysfs.txt [Accessed: 2016-10-10]

[62] J. Smith, "Desktop applications vs. web applications." [Online]. Available: http://www.streetdirectory.com/travel_guide/114448/programming/desktop_applications_vs_web_applications.html [Accessed: 2016-10-11]

[63] S. Shaaban, "Web-based vs "desktop" software." [Online]. Available: http://nurelm.com/web-based-vs-desktop-software/ [Accessed: 2016-10-11]

[64] P. Lubbers and F. Greco, "Html5 websocket: A quantum leap in scalability for the web." [Online]. Available: http://www.websocket.org/quantum.html [Accessed: 2016-10-11]

[65] WebRTC, "Webrtc: Architecture." [Online]. Available: https://webrtc.org/architecture/ [Accessed: 2016-10-17]

[66] Kurento.org, "Kurento media server." [Online]. Available: http://doc-kurento.readthedocs.io/en/stable/introducing_kurento.html#kurento-media-server [Accessed: 2016-10-27]

Appendix A

List of Contributory Open Source Projects

Add any information here that you would like to have in your project but is not necessary in the main text. Remember to refer to it in the main text. Separate your appendices based on what they are for example. Equation derivations in Appendix A and code in Appendix B etc.

Appendix B

Addenda

B.1 Ethics Forms