Imperial College London

Department of Electrical and Electronic Engineering

ELEC50008: Engineering Design Project

Group 18: Mars Rover Project Report

Authors

Aixin Zhang
CID: 01738988
az419@ic.ac.uk

Ebby Samson
CID: 01737449
es1219@ic.ac.uk

Igor Dmytrovich Silin
CID: 01756268
ids19@ic.ac.uk

Kaling Ng
CID: 01737644
kln19@ic.ac.uk

Nur Izzah Mohd Zafer
CID: 01738670
nim19@ic.ac.uk

Xin Wang
CID: 01735253
xw2519@ic.ac.uk

June 14, 2021

Contents

1 Project Management **2**
1.1 Conception and Initiation . . . 2
1.2 Definition and Planning . . . 3
1.3 Performance and Control . . . 4
1.3.1 Project timeline . . . 4
1.3.2 Gnatt chart . . . 4
1.3.3 Team communication . . . 4

2 Rover system design **5**
2.1 Structural design . . . 5
2.2 Functional design . . . 6
2.3 Intellectual property . . . 7

3 Rover Submodules **8**
3.1 Command . . . 8
3.2 Control . . . 9
3.3 Vision . . . 10
3.3.1 Implementation problem . . . 10
3.4 Drive . . . 11
3.4.1 Design . . . 11
3.4.2 Movement . . . 11
3.4.3 Position Control . . . 12
3.4.4 Speed Control . . . 13
3.4.5 Optical Sensor . . . 14
3.4.6 Advanced features . . . 14
3.5 Energy . . . 15
3.5.1 Battery Charge Profile Design . . . 15
3.5.2 State of Charge . . . 16
3.5.3 Battery Balancing Algorithms . . . 16
3.5.4 PV MPPT Algorithms . . . 16
3.5.5 Circuit Design . . . 16
3.5.6 State of Health . . . 16
3.5.7 Energy to Control interface . . . 16
3.5.8 Implementation problem . . . 16

4 Integration: Testing **17**

5 References **18**

1 Project Management

The project team utilised the Project Management Institute's 5 Phases of Project Management [1] as a guide to ensure all aspects of project planning and management are captured in the team's project management approach.

Project management was split into 3 areas, each covering a important section of project management.

1.1 Conception and Initiation

Project definition: Design and build a rover system that has autonomous capabilities to detect, avoid and transmit the locations of the obstacles i.e. coloured balls to a server that users can interact with.

Project requirement: The rover system is split into 5 modules, each with its own requirements:

- Command:
  - Enable bilateral communication between user and Control module
  - Enable users to nagivate the rover
  - Plot a map of the locations of the obstacles encountered by the rover
- Control:
  - Enable bilateral communication channels between Command, Drive, Energy and Vision modules
- Drive:
  - Defines the operation of the two rover motors such as:
    - Speed control
    - Direction control
    - Turning method
  - Using the optical flow sensor, measure the distance travelled by the rover
- Energy:
  - Battery charge operation: Profile design, status estimation and melt/explosion prevention
  - Battery voltage balancing and range estimation
  - Implementing PV MMPT calculation algorithm
  - Integrating and testing solar charging system
- Vision:
  - Using the on-board camera detect, avoid and record the location of obstacles encoutered by the rover

[1]PMI: https://www.smartsheet.com/blog/demystifying-5-phases-project-management

1.2 Definition and Planning

The project team had a significant amount of freedom in designing and developing the rover system to meet the project requirements. The team had identified several design themes that guided the design and implementation choices made during the developement of the rover system:

- **Modularity**:

  Having taken into account that the project team spanned four countries with different time-zones and the time constraint of the project, the team felt it was important to incorporate modular design in the development of each rover modules.

  The approach meant each subsystem only had to ensure the pre-agreed connection interfaces such as WebSocket was compatible with the required modules. This was very advantageous due to the following:

  – Each module could independently develop sections of the rover system. This made the team much more dynamic and efficient.

  – The testing strategy [2] was more methodical and could occur early in stages, gradually leading up to a full rover system test.

  – No unnecessary meetings. By reducing the number of meetings the team had, it meant less time was wasted on arranging a time suitable for three time-zones and team meetings were more productive.

- **Scalability**:

  During the first meeting, the team was not certain as to the exact features that are desirable in a rover system. Due to this reason, scalability was a critical consideration factor and gave the team a very flexible approach to the rover system.

  An example would be the MongoDB database implemented by Command. MongoDB is a type of "NoSQL" database that is not as restrictive as traditional SQL databases which allowed the team to store new types of data without having to redesign the database model.

- **Open-source**:

  Where possible, the team opted to use well-supported open-source development packages such as the FastAPI framework. This complimented modularity and scalability themes by ensuring the interfaces are industry-standard and could be easily modified to expand its capabilites.

  Being well-supported, there is ample documentation to support the development and the codebase is well-designed. This meant that the team could reduce the number of unknown bugs, decrease development time and ensure a high-quality codebase.

- **Minimalism**:

  Due to the open-ended nature of the project, the team did not want to limit the scope of their design but, at the same time, did not want to risk a bloated and inefficient rover system due to feature creep and badly integrated modules.

  The team determined any design choice would need to prioritise efficiency and scalablity. All libraries are to be lightweight and only need to support the required features.

[2]Testing was managed by the Integration module

1.3 Performance and Control

1.3.1 Project timeline

[Timeline chart image]

1.3.2 Gnatt chart

[Gnatt chart image]

1.3.3 Team communication

[Communication heirarchy chart]

Due to the remote collaborative nature required, every member was encouraged to design and implement with a standardised minimalistic approach [3] such as concise code documentation, avoiding too many dependencies and consistent code revisions.

[3]Minimalistic coding guidelines: https://dev.to/paulasantamaria/6-ways-minimalism-can-help-you-write-clean-code-45kp

2 Rover system design

2.1 Structural design

[Structural design diagram]

The team established the structural design of the rover system during the first week of the project timeline. The structural design is formed in three stages:

1. **Identifying the core module**:

   The Control module was designated as the core module of the rover system due the ESP32 board's numerous communication interfaces and on-board data processing capabilities. With the remote nature of the project, this also meant the Control module will take the leadership role and act as the "heartbeat" of the team - ensuring each module development was in sync with the planned project timeline and each communication interface was compatible with Control.

2. **Module connection**:

   With the role of Control established, the team could discuss how to structure the other modules that best took advantage of the ESP32's capabilities, ensured efficient operation in various environments and complimented the *scalability* design theme.

   A major design problem was the location of processing the data from sensors like the optical sensor. The team chose to do as much processing locally on the ESP32 because the connection with Command module servers naturally has a certain degree of latency and Control was best equipped module on the rover to handle the processing. The Control module can also pre-process data to minimise the amount sent to Command and reduce the end-to-end latency of the system. [**latency data**]

3. **Communication interface selection**:

   In the second stage, the Control module worked with each module to research and select the most suitable communication interface. This was critical to establish the interfaces early on to allow the modules to start development as soon as possible and asynchronously as mentioned with *modularity* design theme.

   The biggest concern was the type of connection between the Control module and Command server. The form of connection had to be energy efficient to compliment the *minimalism* design theme and be able to scale easily depending on the data generated from the rover. The final design choice was to use the WebSocket protocol since it was native to the FastAPI framework used by Command and allowed real-time updates to the user. Compared to the HTTP protocol, the WebSocket protocol has a higher performance rating at 83 ms on WebSocket and 107 ms on HTTP for an average single request [3].

2.2 Functional design

The team defined the rover system into three functional layers that each handled a different aspect of the system.

2.3 Intellectual property

Intellectual property rights are managed and enforced by the World Intellectual Property Organisation and are supposed to protect creations of the mind" through concepts such as patents for inventions, and copyrights for logos and software. Provided the inventor files himself or through a patent company and is granted IP protection by an IP office such as the U.K. Intellectual Property Office, it allows the holder of the IP to treat it as a tangible asset and reduce the risks of associated with commercialisation. This, theoretically, fosters innovation.

However, the team firmly believes that space technologies like the mars rover should not be patented. With traditional sectors like space, well-established entities like NASA holds majority of the patents related to space technology – U.S. NASA holds 1600[4] patents while the Indian ISRO only holds 270 [2]. With more companies and countries looking to advance into the space sector, IP laws stifle any potential innovation by increasing the cost and length of development. This directly goes against the fundamental reason of space exploration – advancement of science and exploration of the unknown for the benefit of all humankind. Not for the benefit of a single race or country.

With that mindset, the team endeavoured to use as much open-source software as possible. For example, the Drive module uses the Arduino Serial Peripheral Interface (SPI) and INA219_WE library. The Command module uses the FastAPI framework for servers and React for UI design. Depending on the license of the open-source software used and the wording of the patent application, it could affect the success of filing of a patent if the team wanted to patent any component that used the open-source software. These factors are reasons why it is best to hire a professional patent attorney such as Carpmaels & Ransford LLP.

3 Rover Submodules

3.1 Command

3.2 Control

3.3 Vision

3.3.1 Implementation problem

3.4 Drive

The design of the Drive module was based off the Drive requirements stated in the project requirement section of the report (Page 2).

3.4.1 Design

The Drive receives nagivation instructions from the Control module and was designed to give user full control over the distance a rover goes per instruction and angle that the rover turns. The Drive instruction was defined as:

$$[\text{Action}] + [\text{Turning angle}] + [\text{Distance}]$$

where:

- [Action]: Forward (F), Backward (B), Left (L), Right (R), Stop (S), Emergency stop (X)
- [Turning angle] and [Distance] defines the position and speed control mechanisms

The instruction decode structure is shown below:

```
if(String(instruction[0]) == "F") { // Moving Forward
    F = true;
    target_x = 0;
    // Convert to target distance
    target_y = 100*(String(instruction[3]).toInt()) + 10*(String(instruction[4]).toInt()) +
        (String(instruction[5]).toInt());
    B = false;
    L = false;
    R = false;
    S = false;
    // Accumulate distance or angle
    tmp_x = total_x + target_x;
    tmp_y = total_y + target_y;
}
```

3.4.2 Movement

By using the H-bridge circuit, the DC motors of the rover supports four main movement directions: Forward (F), Backward (B), Left (L), Right (R).

Forward and backward movement was acheived by enabling the H-bridge, setting the PWM signals and the motor states. For example, the code below defines forward movement:

```
void MovingForward() {
    digitalWrite(pwmr, HIGH); // Setting right motor speed at maximum
    digitalWrite(pwml, HIGH); // Setting left motor speed at maximum
    // Set motor state
    DIRRstate = LOW; // LOW - Forward, HIGH - Backward
    DIRLstate = HIGH; // HIGH - Forward, LOW - Backward
    digitalWrite(DIRR, DIRRstate);
    digitalWrite(DIRL, DIRLstate);
}
```

Turning left and right was achieved by setting the respective wheel state to LOW. After turning, any distance change was in the x-direction and would equal the circumference defined by the rover's optical sensor. Turning angles were defined from the point where the circle's radius was the distance between the optical sensor to the midpoint along the rover shaft which was measured as 15 cm. It was detected that due to the two wheels not outputting the same amount of power. This affected the turning angle calculations since the circle radius i.e. the distance between the optical sensor to the midpoint along the rover shaft was 17.8 cm and calculations had to be compensated as shown below:

```
float Angle_Conversion(float target_angle) {
    return (target_angle/90)*28; // Convert input angle to distance along x-axis
}
```

The halt instruction was implemented by disabling the H-bridge and set the PWM signal feeding into the left and right wheel pins to LOW.

3.4.3 Position Control

The rover's position control was implemented with a closed-loop PI controller which used the error margin between the current rover position and the previous rover position. The logic was that if the rover moved forwards and backwards by 20 cm and the optical sensor detected that the rover did not reverse 20 cm, the error margin would be the difference between the current rover position and the previous rover position.

The controller was designed such that if the error margin was either in the range $-0.5\text{cm} < \text{error in } y < 0.5\text{cm}$ or $-0.78\text{cm} < \text{error in } x < 0.78\text{cm}$, the controller would consider the difference insignificant and rover would stop. Otherwise, the rover would either keep executing the current instruction or perform the reverse movement until it was within the error margin.

```
if(F){ // Forward movement position control
    err_y = total_y - tmp_y; // Calculate error
    cp = pi_d(err_y);
    Speed_Control(abs(cp)*0.01+0.4); // Using PI controller result to control speed

    // Position Control
    if(err_y < -0.5) { MovingForward(); }
    if(err_y > 0.5) { MovingBackward(); }

    if(err_y > -0.5 && err_y < 0.5) {
        // First time the error lies in the given range
        if (firstTime) {
            instructionCompleteTime=millis();
            firstTime=0;
        }

        // Check the motion after 10sec for position correction
        if(millis()-instructionCompleteTime > 20000 && !firstTime && !instructionCompleted) {
            Stop();
            instructionCompleted=1;
            Send_Instruction_Completed(total_y,1,'F');
        }

        Stop();
    }
}
```

The intial design for position control was a P controller which could acheive proportional controlling but the settling time is too long as a result of the steady-state error. By adding the I (Integral) controller, the steady-state error effect was mitigated. The best effect of the PI-controller was achieved by setting $K_p = 1$ and $K_i = 1$. Taking advantage of the closed-loop control strategy, position control and speed control were linked together to achieve more advanced features that are elaborated in the next section.

As an additional feature of position control, Drive module returns a SUCCESS signal to the Control module once an instruction was sucessfully completed. After 20 seconds, if the instruction was still not completed, the Drive module will return a FAIL signal and stop the rover immediately to await further commands. These signals can be directed to the Command module to display the rover's movement status on the Terminal interface built into the Command website and, with the WebSocket connection, gives real-time feedback on the rover's Drive status.

3.4.4 Speed Control

The speed control was achieved by using the the PWM signals as inputs into the SMPS board duty cycle to control the output voltage applied to the motors. The implementation of speed control is shown below:

```
void Speed_Control(float duty){
    // Use duty cycle as argument to control voltage thus control speed
    analogWrite(6, (int)(255-duty*255));
}
```

By inputting the position error in either the x or y direction into the PI control function, its output was inputted into the speed control function. After testing and caliberation, the duty cycle input function was determined as:

$$(\text{error from PI controller} \times 0.01) + 0.4$$

where 0.4 was the minimum voltage required to activate the rover. If the rover stays at a place for a long time,due to the presence of the integral controller, the error would accumulated as time increases and the speed control function will increase the rover speed to correct the error. Through testing, the rover initially accelerated due to a significant initial error. As the rover approached the target, the rover decelerated, indicating that the measurements taken by the optical sensor was more accurate. This method effectively reduced the number of oscillations and resulted in a more accurate Drive performance.

By using the formula speed $= \frac{\text{distance travelled}}{\text{timestamp difference}}$, the relationship between the duty cycle and speed can be displayed as a look-up table:

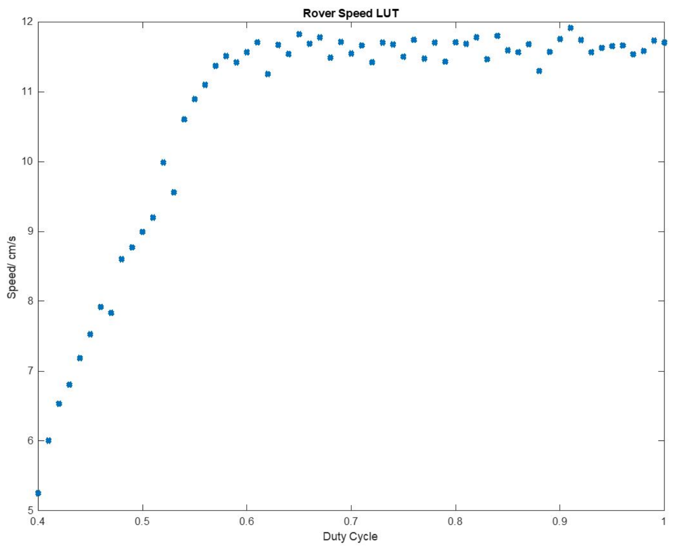

Figure 1: Relationship between the speed and duty cycle

This relationship formed the foundation of two advanced Drive features. By using the tilt data from the FPGA in the Vision component, the rover can detect when it is going up a slope and will increase the speed relative to when the rover is on a flat surface. In addition, Drive could support a power-saving mode where the duty would be fixed between 0.4 to 0.8 to conserve the energy used.

3.4.5 Optical Sensor

The optical sensor measures the distance of the rover travelled and the turning angle. To obtain more accurate measurements, the data type was changed from integer to float. The turning angle and travel distance was sent to the Control module for calculating the coordinates along x and y direction. The orientation was defined as:

- The y-coordinate or x-coordinate changes positively if the rover faces North or East respectively.
- The y-coordinate or x-coordinate changes negatively if the rover faces South or West respectively.

3.4.6 Advanced features

- Emergency Stop:

  Drive module has an "Emergency Stop" action (X) which could be issued by the Vision module when it detected an obstacle that was not a ball. The "Emergency Stop" is different from the standard "Stop" where if the "Emergency Stop" was used, the rover status will be updated to display an error has occured which is not the case for the standard "Stop" action.

- Drive pausing:

  By issung the Pause (P) and Unpause (U) commands, the rover would stop and allow the Vision module to caliberate, detect and process the distances between the rover and an obstacle. After the Unpause command is issued, the rover would continue executing the current set of instructions. The Arduino code for this feature is shown as follows:

  ```
  if(instruction[0] == 'P'){ // Pause state
      Stop();
      while(!Serial1.available() || Serial1.peek()!='U') { // Unpause state
          delay(50); // Delay 50 ms
      }
      Serial1.read(); // Remove 'U' action word
      instructionStartTime=millis();
  }
  ```

3.5 Energy

3.5.1 Battery Charge Profile Design

A battery charging profile ensures that the cells are charged to their full capabilities while within the **constraints of the batteries and solar panels**. The capacity of the cells were determined by the following formula before developing the battery charging profile:

$$\text{Cell capacity} = \frac{\text{Discharging state time}}{3600} \times \text{current}$$

The Energy module developed its own charging code that used a combination of Constant Current (CC) and Constant Voltage (CV) instead of the original method in the provided code which only used Constant Current to charge the cells. Constant Current was used when the cell was nearly empty to prevent overheating during charging[1]. When the battery has been charged to 3500 mV i.e. around 90%, Constant Voltage was used to charge it to avoid overcharging.

The cell capacity measured by the provided standard code is 502 mAh **while the value measured by the custom charging code was around** 565.07 mAh. This difference in measured cell capacity indicates that charging using a combination of Constant Voltage and Constant Current is more effective at fully charging the battery compared to using Constant Current only. The performance of the two charging methods over a period of time is shown below:

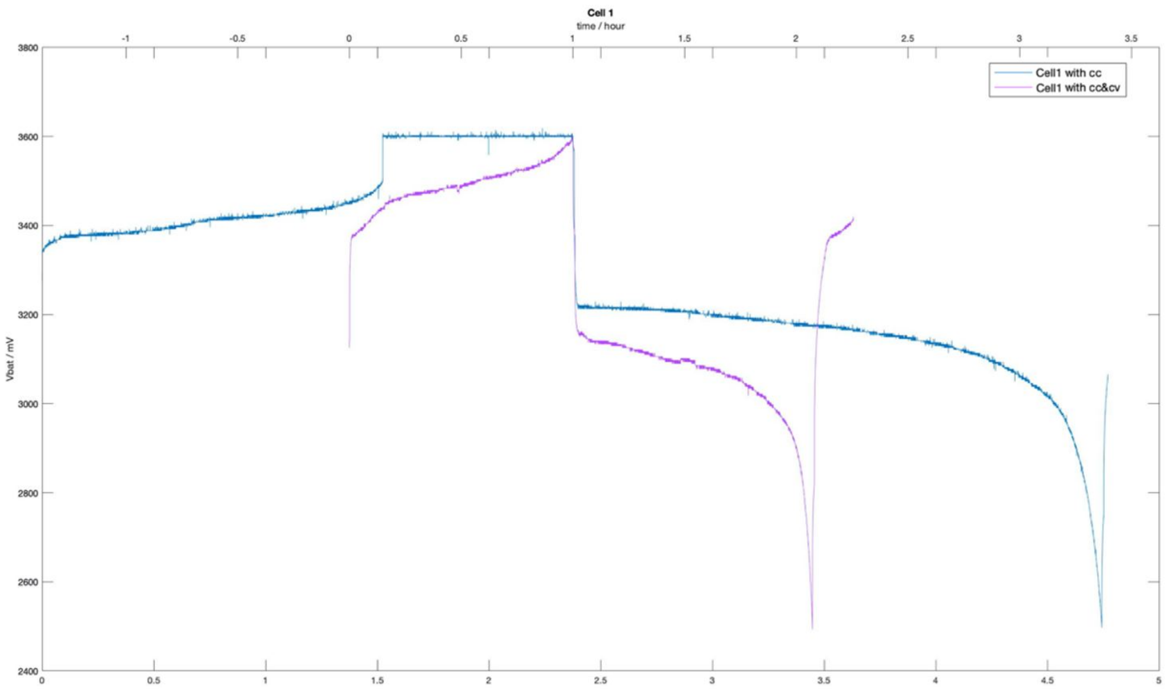

Figure 2: CC vs CC + CV performance comparison

3.5.2 State of Charge

3.5.3 Battery Balancing Algorithms

3.5.4 PV MPPT Algorithms

3.5.5 Circuit Design

3.5.6 State of Health

3.5.7 Energy to Control interface

3.5.8 Implementation problem

4 Integration: Testing

5 References

[1] *Battery Charging Methods & Terminology*. 2021. DOI: https://www.heliosps.com/knowledgebase/battery-charging-methods-terminology/.

[2] *ISRO Technology Transfer Patent*. 2021. DOI: https://www.isro.gov.in/isro-technology-transfer/patent.

[3] David Luecke. *HTTP vs Websockets: A performance comparison*. 2018. DOI: https://blog.feathersjs.com/http-vs-websockets-a-performance-comparison-da2533f13a77.

[4] *NASA Technology Transfer Program - Patent Portfolio*. 2021. DOI: https://technology.nasa.gov/hot100/.