Red-Badger

Points to take into consideration when accessing the provided issue.

1. The average time it takes for a signal to travel between Earth and Mars is about 12.5 minutes
    a. We can't try to solve problems in real time.
2. It is currently estimated to be around $130 to $400 per kilogram to send something to mars.
    a. We should thrive to reduce the loss rate with better recovery and detection hardware.
3. The average weight of Mars rovers, excluding the small Sojourner rover, is roughly 600 kg.
4. Confirmation of movement is done with GPS.
5. Quantity and frequency of commands being sent not defined
6. A magnetic compass does not work on Mars. NESW will not work so mapping to compass degrees.
7. No Z axis is taken into account. Not vertical checks.
8. No obstacles are taken into account.
9. Width and height of the robot are constant and will stay the same as the grid size.
10. Energy/battery life needs to be taken into account.
11. Speed of robot is not defined
12. Failures/losses are stored server side on earth
13. There is an assumption that even if the robot is lost it will transmit the final result
    a. We should return a result after each movement
    b. There will be a timer and a trigger in the db for this.

Data Structures and Variables

1. Direction is based on 360 degree values (No Compass)
    a. Min -180
    b. Max 180
    c. All directions can be achieved with the above values.

2. Mappings
    a. R = 90
    b. L = -90
    c. F = 1 GI(grid item)
        i. This could be 1 m, 1 km depending on movement config.

3. Battery
    a. Energy is a big issue on Mars and you can't run the rover constantly.
    b. A check needs to be done before each movement
        i. If movements can’t be achieved the current battery value will be returned.
    c. Battery will be a max of 100
    d. One battery unit consumed per 1m
    e. 24hrs to go from 0 - 100 capacity

Pseudo functions to be used

Below can be the different functions to be used on server and device side.

Server

**If there is ONE Robot running and ONE command being sent you can just use a normal api but seeing as we would want this to scale lets add a queuing system in place so that we can submit a bigger area and wait for results. I would recommend starting with smaller batches and areas.*

```
submitCommands(startPosition: {x-axis:number, y-axis: number},
gridSize:{x-axis:number, y-axis:number}, robotId:string,movementPlan:
Array<string>): commandId
{

   validMovementPlan = validateMovementPlan

   currentPosition = getCurrentPosition(robotId)

   gridValid = validateGrid(gridSize,currentPosition)

   If grid is invalid log results and return error

   If grid is valid submit command for robot to use

}



validateGrid(gridSize:{x-axis:number,
y-axis:number},currentPosition:{x-axis,y-axis}):boolean
{
   Do calculation with current position and grid size if any points overlap
   stored data

   If they do overlap, check for failures.

   Best to use relational db;
}



getCurrentPosition(robotId:string): {x-axis,y-axis}
```

```
validateMovementPlan(movementPlan<string>,gridSize:{x-axis:number,
y-axis:number},  ):{boolean,message}
{
   Validate if movement plan is less than or equal to 100 chars
   Does the movement plan cover the entire grid, return message if the plan
   is inefficient. We could reduce the movement plan

   OR

   We generate a movement plan and all that needs to be passed is the grid.

   This can be done with some prompting from a AI bot.
}


listenForCommandresult(commandId)
{
   We can either poll the DB or set up a websocket and have the db fire an
   event on specific changes to the logs table

   Based on the the speed of the robot and the responses from mars we could
   determine that based on the movement plan how long a command should take
   to complete. If the command is not resolved in said time we will have it
   last known result and can map the NEXT one as a LOST
}


getCommandResult(commandId)
{
    Return the results for a specific command
}


robotMoved(commandId:string, currentPosition:{x-axis,y-axis},
currentDirection(angle), movementPlanIndex:number, movementPlanlength:
number ,currentBatteryLevel:number){

   This will log each and every move of the robot
   If the the move is the final index submit an event to
   listenForCommandResult()

}
```

Robot/device

```
init()
{
   listenForMovementCommands(startPosition: {x-axis:number, y-axis:number,
   direction:(angle)},gridSize:{x-axis:number, y-axis:number},movementPlan:
   Array<string>)
   {

      enoughBattery = validateBatteryUsage(movementPlan)

      If not enough battery return failure - error message can include
      estimated time until battery is sufficient

     loop through movement plan and check

     Let item in movement plan
     {
         If item is L turn -90
         If item is R turn 90
         If item is F move one grid item in direction facing
     }
   }


}




listenForBatteryChanges(): Listener<number>

*This might be needed in future to listen for changes.



listenForMovementCommands(startPosition: {x-axis:number, y-axis: number},
gridSize:{x-axis:number, y-axis:number} )



getCurrentBatteryLevel():number
```

```
validateBatteryUsage(movementPlan: Array<string>):boolean
{
   batteryLevel = getCurrentBatteryLevel()
   Do calculation based on movement plan
}



listenForMovementChanges()
{

   Call api - robotMoved(commandId:string, currentPosition:{x-axis,y-axis},
   currentDirection(angle), movementPlanIndex:number, movementPlanlength:
   number ,currentBatteryLevel:number)
}



moveRobot(x-axis,y-axis)
{
   Depends on hardware what we need to call to make the robot move
}
```