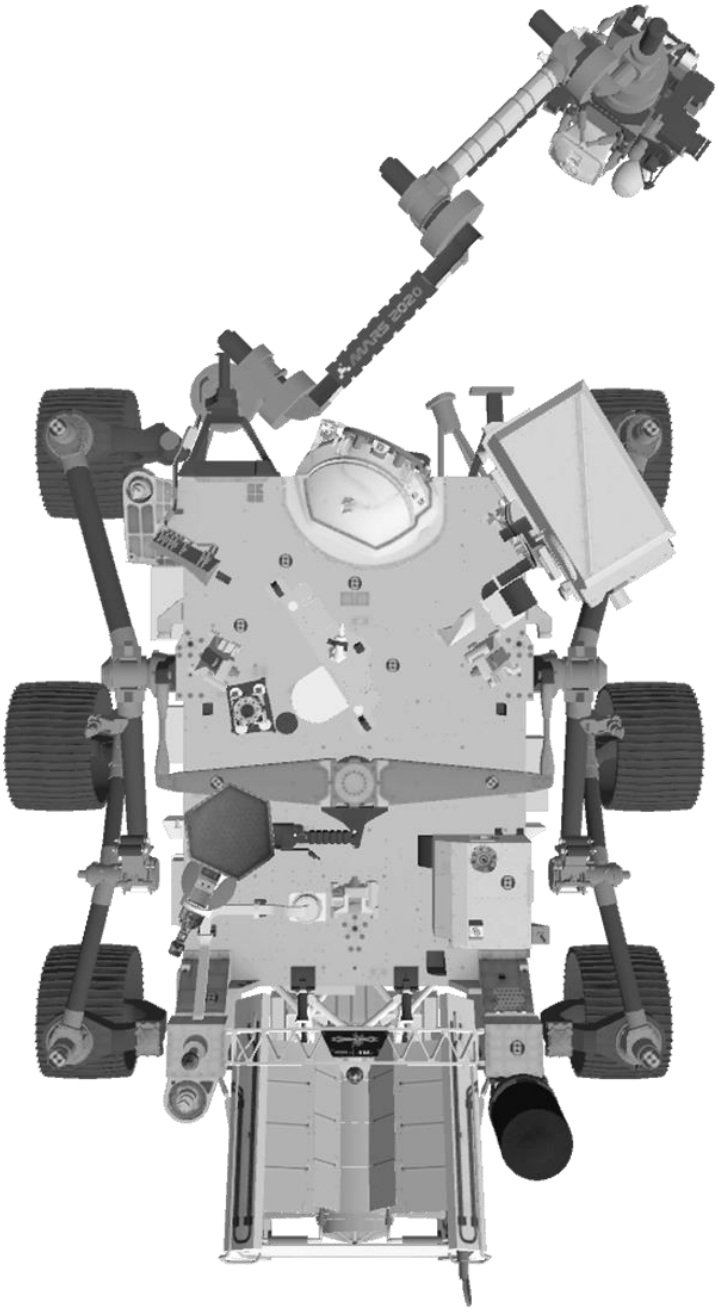
TECHNICAL UNIVERSITY OF LIBEREC



Virtual Interactive Model of a Perseverance Rover

INSTRUCTION MANUAL

Table of Contents

List of Figures

List of Tables

1 Introduction

This manual was created for the proper operation of the virtual rover application. It is also used to describe rover's dimensions, the user interface, its action and sensing elements and implemented functions.

2 Rover Description

The rover is divided into three main parts:

- Chassis
- Robotic arm
- Camera Mast

Each part has built-in motor components to move the individual parts. Each motor also has built-in sensors to check their values and current position. Individual motors can be controlled by setting their values either by command or in the user interface.

2.1 Chassis

The chassis consists of six wheels and a total of ten engines. Each wheel has an engine to adjust the torque and the four side wheels have an additional engine to adjust their rotation.

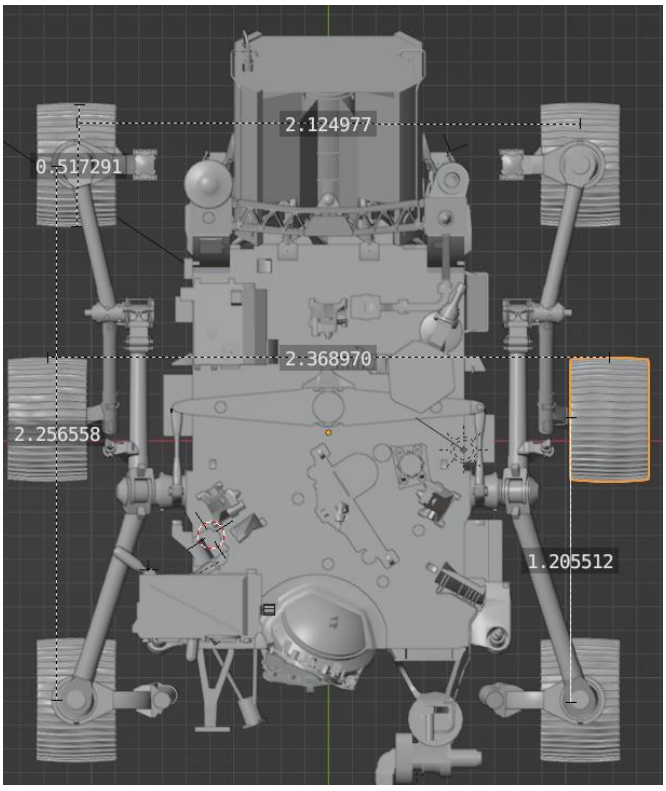


Fig. 1: Rover chassis dimensions

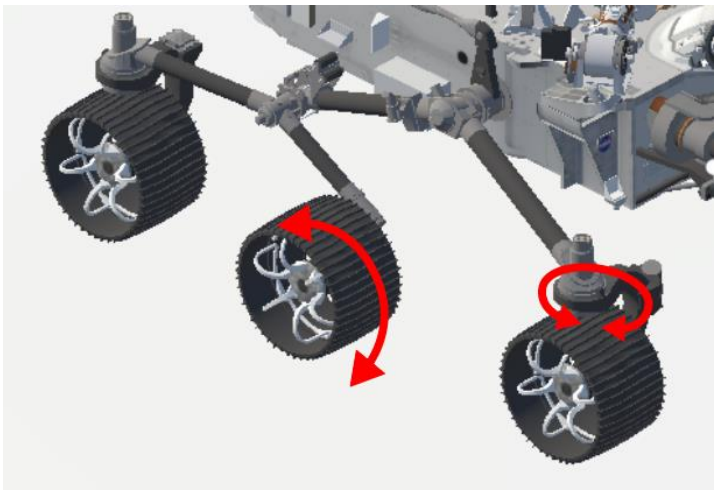


Fig. 2: Wheels movement

Table 1: Chassis engines

	Engine name	min value	max value
Rotation	FrontRightWheel	-50	50
	RearRightWheel	-50	50
	FrontLeftWheel	-50	50
	RearLeftWheel	-50	50
Speed	FrontRightTire	-100	100
	FrontLeftTire	-100	100
	MiddleRightTire	-100	100
	MiddleLeftTire	-100	100
	RearRightTire	-100	100
	RearLeftTire	-100	100

- Rotation of wheels is in degrees
- Speed of wheels is in degrees per second

Every wheel also has sensors. Sensor names are derived from engines:

Sensor name = *EngineName* + *SensorType*

Sensor types can be:

- Position → *SensorType = "Pos"*
- End sensors → *SensorType = "Max"* or *"Min"*

The return value of the end sensors is 0 or 1 based on the comparison of the current engine position and its minimum or maximum values. The return value of the position sensor is the current position of the engine.

Table 2: Chassis sensors

Sensor name
Engine name + (Min/Max)
Engine name + Pos

Example: FrontRightWheelPos = name of the front right wheel position sensor.

2.2 Robotic Arm

The robotic arm consists of a total of five joints. Each joint has a motor into which values can be entered in a similar way to the chassis control.

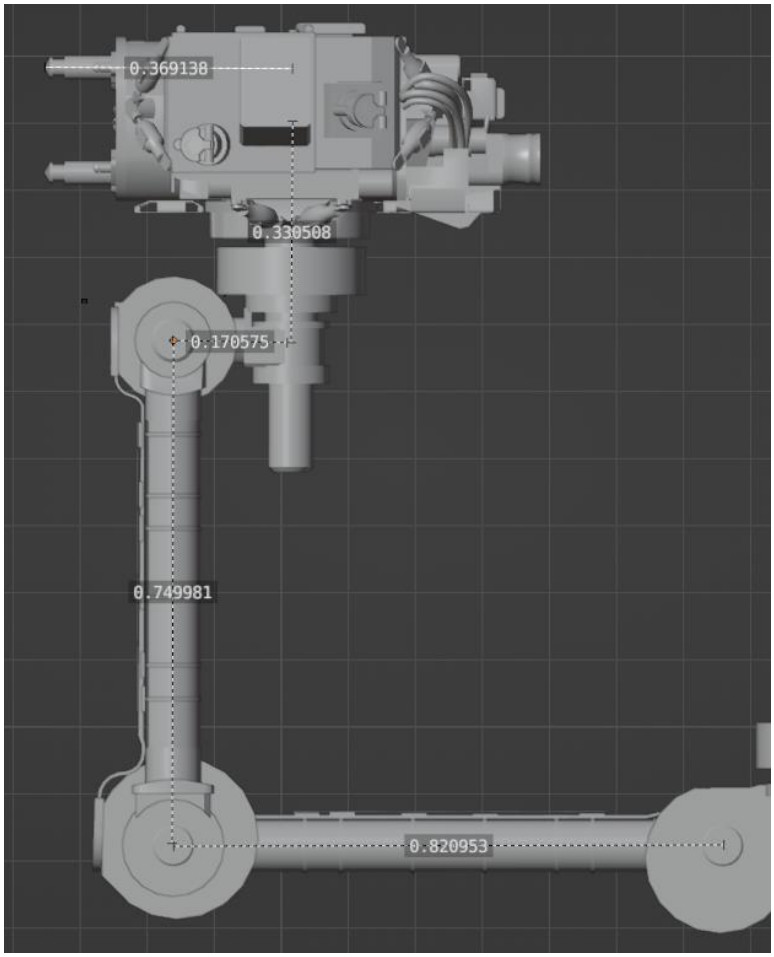


Fig. 3: Robotic arm dimensions

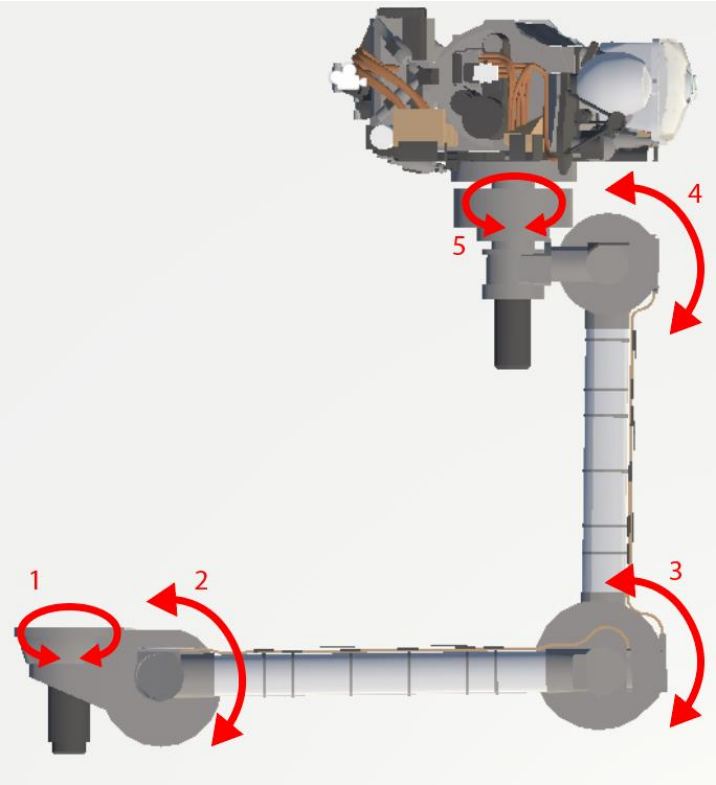


Fig. 4: Robotic arm movement

Table 3: Robotic arm engines

Pos	Engine name	min value	max value
1	ShoulderAzimuth	0	180
2	ShoulderElevation	-45	45
3	ArmElbow	-360	360
4	ArmWrist	0	180
5	ArmTurret	-360	360

- Robotic arm engine values are in degrees.

The robotic arm has similarly designed sensors as the chassis (see table 2).

2.3 Camera Mast

The Mast camera consists of three joints with three motors that can be controlled in the same way as the other motors.

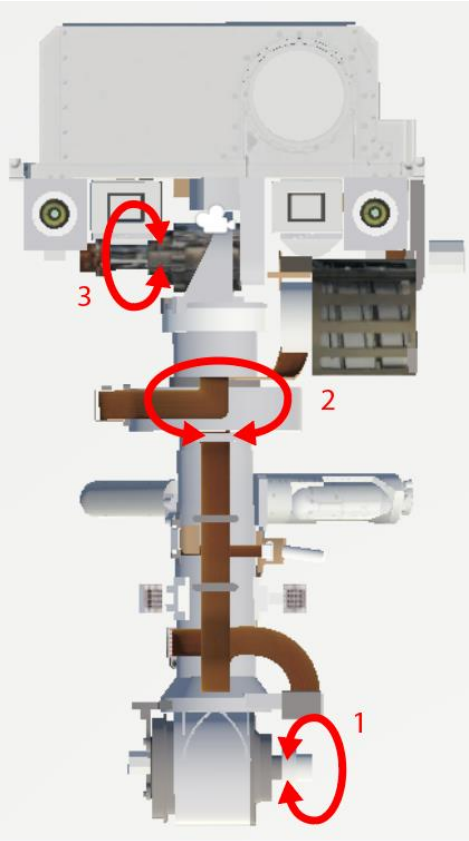


Fig. 5: Camera Mast movement

Table 4: Camera Mast engines

Pos	Engine name	min value	max value
1	MastBase	-90	0
2	MastTop	-360	360
3	MastHead	-90	90

- Camera Mast engine values are in degrees.

The camera Mast has similarly designed sensors as the chassis and robotic arm (see table 2).

2.4 Other Cameras

A total of 6 cameras are implemented in the model. Except for the NavCams camera located on the rover mast, the cameras have the following features:

- Horizontal field of view: **131** °.
- Resolution: **1920 x 1080 pixels**
- Color format: **R8G8B8A8_UNORM**

NavCams camera has a horizontal field of view of **105.5 °** and the same parameters as the others.

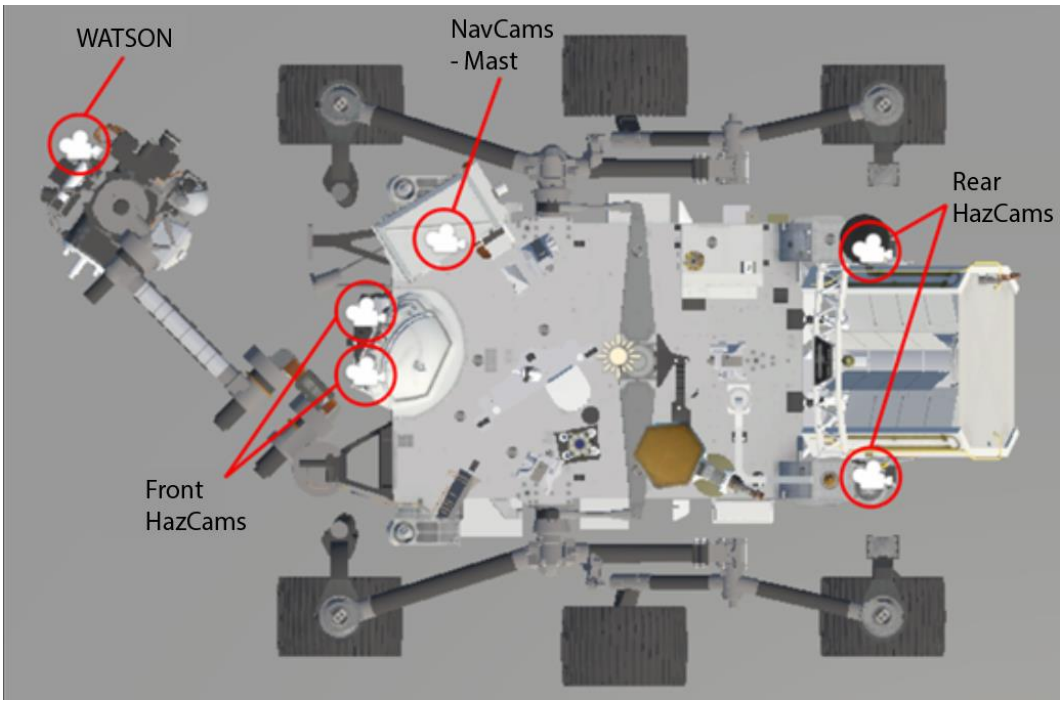


Fig. 6: Cameras on model

Table 5: Names of all rover cameras

Camera name
NavCams
FrontRight
FrontLeft
RearRight
RearLeft
WATSON

3 User Interface (UI)

The UI design is based on the functions that are required by the user to configure and control the model. The functions that the UI contains are:

- Help - Opens manual (this document)
- Wheel controls
- Arm&Mast controls
- Cameras controls
- Sensors
- Console
- Settings
- Time controls

3.1 Wheel Controls

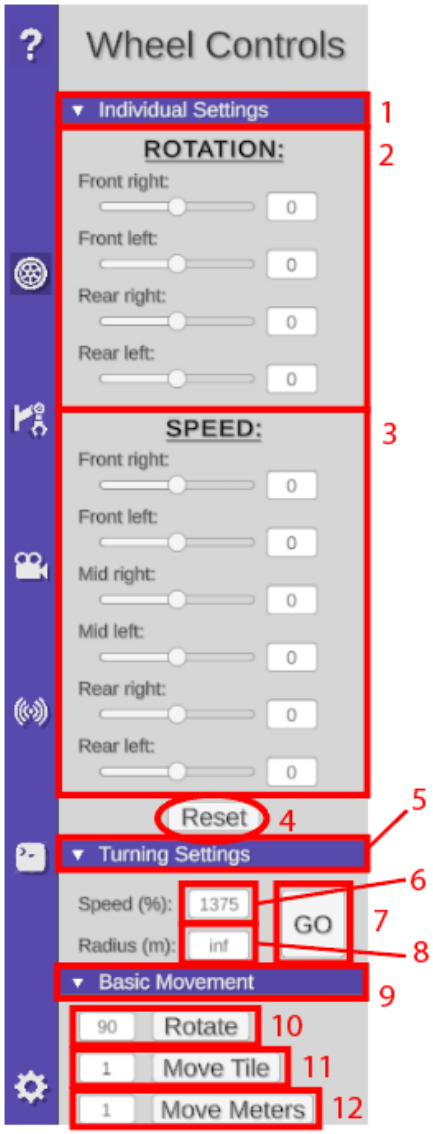


Fig. 7: UI of wheel controls

1) **Button** that opens individual slider settings of wheels.
2) The individual sliders of the wheel engines for controlling the rotation.
3) The individual sliders of the wheel engines for controlling the speed.
4) **Button** that resets sliders to initial values.
5) **Button** that opens panel with function for turning.
6) Speed **input** for turning function (min.: **-1375 %**, max.: **1375** %). 100% = real rover maximum speed of a wheel (7,3 degrees per second).
7) **Button** for starting the turning function .
8) Radius **input** for turning function (min.: **-infinity (inf)**, max.: **inf** meters). Radius < 2,5 causes the rover to rotate in place.
9) **Button** that opens panel with function for basic movement.
10) **Input** for function rotate (range **± 360 °**).
11) **Input** for function movement on the grid – used in playground map (± **number of tiles).**
12) **Input** for function movement by distance (± **distance value** meters).

3.2 Arm&Mast Controls

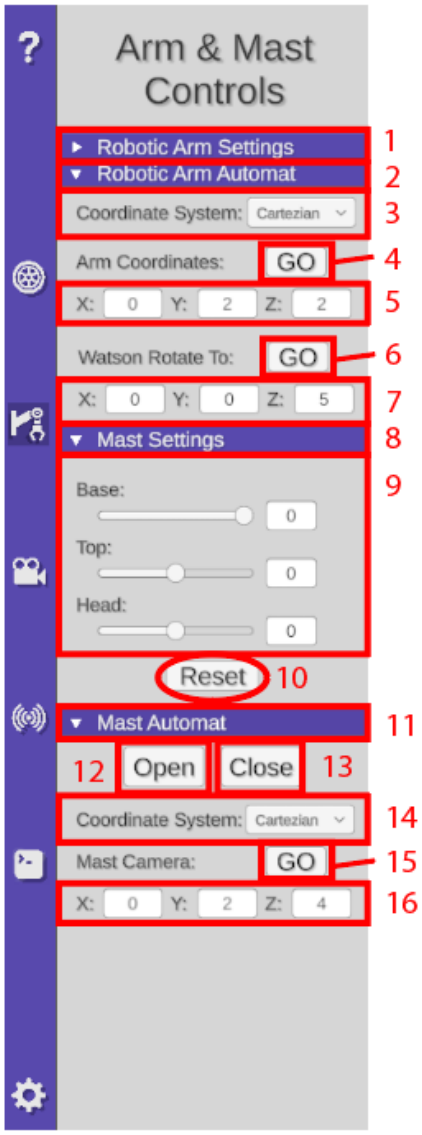
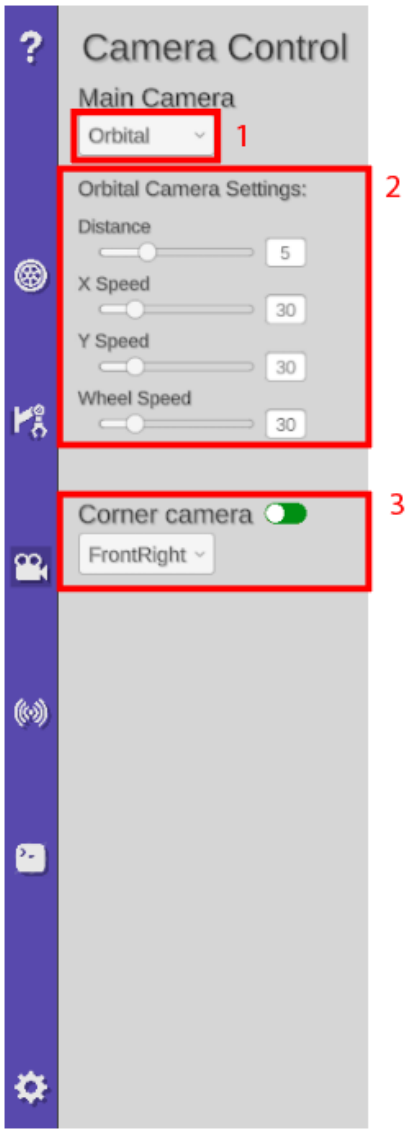


Fig. 8: UI of arm&mast controls

1) **Button** that opens robotic arm sliders.
2) **Button** that opens panel with function for automatic robotic arm.
3) **Drop down** for selection coordinate system (cartesian or spherical).
4) **Button** that starts the automatic function.
5) **Input** for setting the arm target position.
6) **Button** that starts the Watson function (straighten and look at the target position).
7) **Input** for Watson target position.
8) **Button** that opens camera mast sliders.
9) Slider controls of a mast camera.
10) **Button** that resets sliders to initial values.
11) **Button** that opens panel with function for automatic mast camera.
12) **Button** for tilting the camera.
13) **Button** for folding the camera.
14) **Drop down** for selection coordinate system (cartesian or spherical).
15) **Button** for starting the automatic function.
16) **Input** for setting the target position for mast automatic function.

3.3 Cameras Controls



1) **Drop down** options for choosing main camera (orbital, top-view, VR).
2) Settings for orbital camera.
3) **Toggle** for corner camera + drop down options for selection.

Fig. 9: UI of camera controls

3.4 Sensors

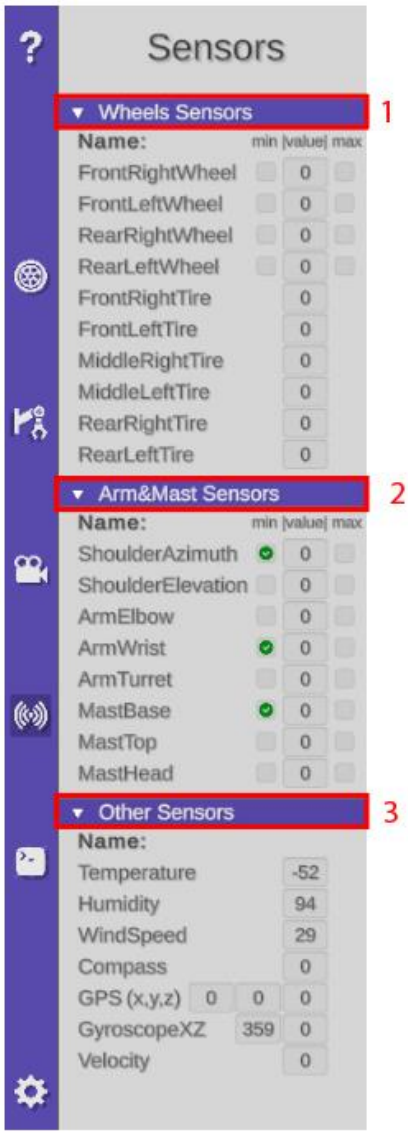


Fig. 10: UI of list of sensors

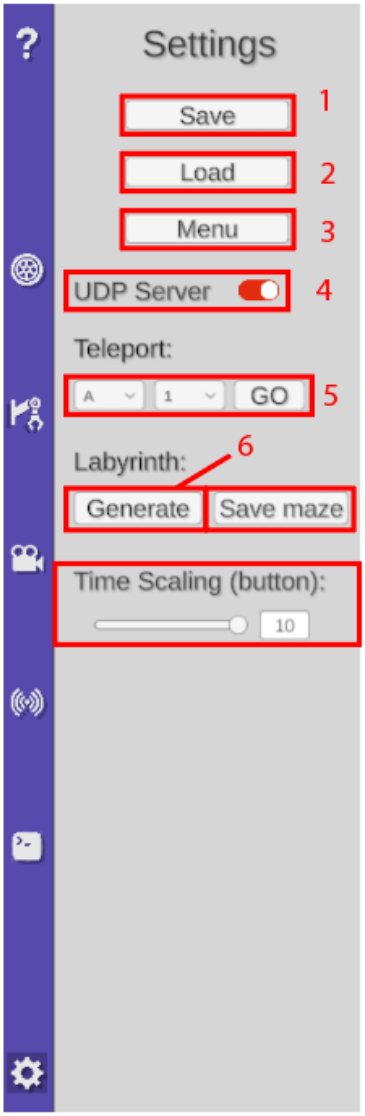
1) **Button** for opening the list of wheels sensors.
2) **Button** for opening the list of Arm&Mast sensors.
3) **Button** for opening the list of other sensors.

3.5 Console

1) **Button** for confirm commands.
2) **Button** for loading commands from .txt files.
3) **Button** for clearing the console.
4) **Button** for saving confirmed commands.
5) **Input** for commands.

Fig. 11: UI of console

3.6 Settings



1) **Button** for saving the project as .json file.
2) **Button** for loading project from .json file.
3) **Button** for going to the main menu.
4) **Toggle** for activating the UDP server.
5) Teleport function usable only in the playground map.
6) **Button** for generating new labyrinth in the playground map.
7) **Button** for saving the labyrinth to .txt file.
8) **Input** for time scaling used in fast forward function.

Fig. 12: UI of settings

3.7 Time Controls

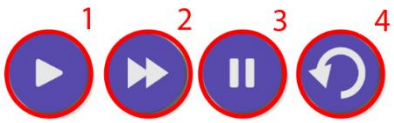


Fig. 13: UI of time controls

1) **Button** for normal time speed
2) **Button** for faster time speed based on settings input
3) **Button** for pausing the time
4) **Button** for resetting the project

4 Playground Map

The playground map is divided into 4 quadrants. In the first quadrant, there is a maze that can be generated using the “generate maze” function. The second quadrant contains obstacles for movement tasks. The third quadrant is where objects for image recognition tasks and camera usage are located. In the final, fourth quadrant, a moving ball and gates are placed, where free tasks can be performed. Additionally, there is a central circle that serves as a calibration point for controlling the chassis.

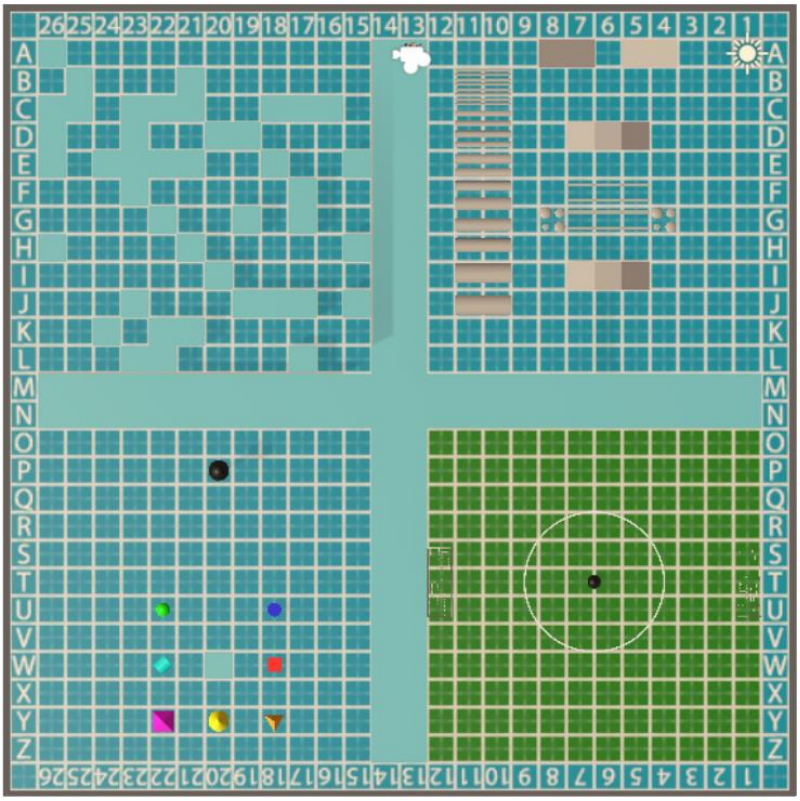


Fig. 14: Playground map

Points of Interests:

A 10/11	-	A place to start testing the chassis by driving over the rollers.
G 9 / F 8	-	A place where a steady drive can begin
W 20	-	Centre of objects for camera and image recognition tasks.
T 4/9	-	A place where chassis control calibration and testing can begin.

5 Commands and UDP Communication

The application can be controlled using commands that are custom created. The table 6 lists all possible commands. The commands are not case sensitive, but must be spelled correctly.

Commands can be entered into the console, that is included in the user interface.

Commands can also be sent using a UDP server. To send commands to rover application use IP address of computer that runs it with default **port 8051.**

5.1 Python Example Connection

Send Data:

```
14  serverAddressPort   = ("127.0.0.1", 8051)
15
16  UDPClientSocket = socket.socket(family=socket.AF_INET, type=socket.SOCK_DGRAM)
17
18  UDPClientSocket.sendto(str.encode("MoveTile 5"), serverAddressPort)
```

Fig. 15: Example of sending a command to the rover application with a Python

Use method **sendto ( )** to send datagrams to a UDP socket.

Send and Receive Data:

```
14  serverAddressPort   = ("127.0.0.1", 8051)
15
16  bufferSize          = 1024
17  #-------------------------------------------------------------------
18  UDPClientSocket = socket.socket(family=socket.AF_INET, type=socket.SOCK_DGRAM)
19
20  def Send(data:str)->str:
21      # Send to server using created UDP socket
22      UDPClientSocket.sendto(str.encode(data), serverAddressPort)
23
24      msgFromServer = UDPClientSocket.recvfrom(bufferSize)
25
26      print(len(msgFromServer[0]))
27
28      return msgFromServer[0].decode("utf-8")
29
30  res =  Send("sensor Humidity")
```

Fig. 16: Example of sending and recieving data with a use of Python

5.2 MATLAB Example Connection

Send data:

```
ip = '192.168.0.208';
port = 8051;
u = udpport("datagram");

write(u, ip, port, "MastOpen");
```

Fig. 17: Example of sending a command to the rover application with a MATLAB

Use function **uddport ( )** to perform byte-type and datagram-type UDP communication using a UDP socket on the local host.

Use function **write** to write data to the instrument.

Send and Receive Data:

```
function res = SendCommand(u, ip, port, command)
tryAgain = true;

while tryAgain
    write(u,command, ip, port);

    data = [];

    tic

    while toc<1
        if u.NumDatagramsAvailable>0
            dataSegment = read(u, 1, "uint8");

            dataS = dataSegment.Data;

            res = native2unicode(dataS);
            return;
        end
    end

end

res = "Error";

end
```

Fig. 18: Example of sending and recieving data with a use of MATLAB

Command	Object	Input	Output	Example	Description
Engine	*"Engine name"*	float	confirmation message	engine ShoulderAzimuth 50	*sets the engine value*
Sensor	*"Sensor name"*	-	sensor name + actual position	sensor ShoulderAzimuthPos	*gets value of the sensor*
TurningRadius	-	2x float	confirmation message	TurningRadius 5 1375	*begins to rotate along radius 5 with speed 1375%*
Collisions	-	-	collision status	Collisions	*gets all collisions*
Rotate	-	float	confirmation message	Rotate 90	*rotates rover by (Input) degrees*
MoveMeters	-	float	confirmation message	MoveMeters 5	*moves rover (Input) meters*
MoveTile	-	float	confirmation message	MoveTile 1	*moves rover (Input) number of tiles*
StartCamera	*"Camera name"*	-	confirmation message	StartCamera NavCams	*activates the corner camera*
Teleport	-	char, int	confirmation message	Teleport A 15	*teleports rover to (Input)*
ArmXYZ	-	3x float	confirmation message	ArmXYZ 2 2 2	*sets robotic arm to (Input) destination in cartesian*
ArmSpherical	-	3x float	confirmation message	ArmSpherical 45 0 1	*sets robotic arm to (Input) destination in spherical*
WatsonXYZ	-	3x float	confirmation message	WatsonXYZ 0 0 5	*straighten the watson and sets it to the targeted plane in cartesian*
WatsonSpherical	-	float	confirmation message	WatsonSpherical 50	*straighten the watson and sets it to the targeted plane at angle.*
MastXYZ	-	3x float	confirmation message	MastCam 2 2 2	*sets camera Mast to (Input) destination in cartesian*
MastSpherical	-	2x float	confirmation message	MastSpherical 45 0	*sets camera Mast to (Input) destination in spherical*
MastOpen	-	-	confirmation message	MastOpen	*sets engine MastBase to -90*
MastClose	-	-	confirmation message	MastClose	*sets engine MastBase to 0*
GetPicture	*"Camera name"*	-	confirmation message	GetPicture NavCams	*saves/sends the picture from*
IsReady	-	-	Ready/Not Ready	IsReady	*checks functions Rotate, MoveTiles and MoveMeters if finished*

Table 6: List of commands

6 Implemented Functions

The rover incorporates multiple functions to facilitate its control. The chassis control module includes a function for executing turns, as well as two functions for driving specific distances measured in tiles or meters. Additionally, there is a function dedicated to rotating the rover in place. Furthermore, there are additional functions designed for controlling the robotic arm using coordinate inputs. These functions enable precise manipulation of the arm's movements. Additionally, there is a camera mast function that adjusts the motor values of the mast based on the input provided. This allows for dynamic positioning and control of the camera mounted on the mast.

6.1 Chassis Functions

Turning Function:

This feature can be found in the wheel controls interface under the "Turning Settings" tab. It can also be accessed by using the "**TurningRadius**" command. In order to utilize either of these options, it is necessary to provide two parameters.

1) **Radius**
   - Input range is **± infinity** meters (infinity = no wheels tilt).
   - If input is in range (**2,5; -2,5**) rover will rotate in place.
2) **Speed**
   - Input range for speed is **±1375% = 1,65 km/h = 100 degrees per sec on wheel** (100% = 7,3 degrees per second which is the maximum wheel speed of a real rover).

     The application was adjusted to increase the speed of the rover as the actual rover was operating at very slow speeds.

Example: Radius = 5 m, Speed = 1375 % → the rover will rotate in a radius of 5 meters at maximum speed

Basic Movement Functions:

This function is located in the wheel controls interface under "Basic Movement" tab. There are three functions. For rotating the rover, moving rover by tiles and moving rover by meters.

Rotate Function: can be called using "**Rotate**" command. There is a single input for an angle in degrees that will start rotating the rover.

Move Tile Function: can be called using **"MoveTile"** command. There is a single input for distance in tiles (only in Playground map) that the rover will move. Note that when using the function, rover can move only in 2 directions by tiles. It will move in the direction in which it is rotated and will prevent all rotation and movement to the other axis.

Move Meters Function: can be called using "**MoveMeters**" command. There is a single input for distance in meters that the rover will move. This function does not affect the avoidance of movement to another axis and can move freely.

6.2 Robotic Arm Functions

Automatic Control:

This feature can be found in the Arm&Mast interface under the "Robotic Arm Automat" tab. It can also be accessed by using the "**ArmXYZ**" or "**ArmSpherical**" command depending on the desired coordinates. In order to utilize either of these options, it is necessary to provide three input coordinates for position of the target.

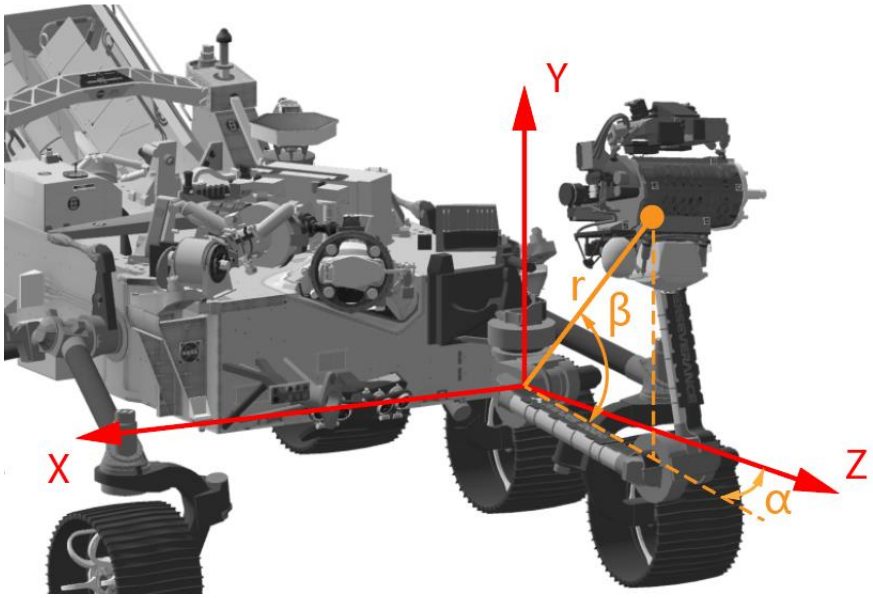


Fig. 19: Cartesian and spherical coordinates for the robotic arm function

Watson Rotate To:

This function is located right below the automatic control function. It can also be accessed by using "WatsonCam" with 3 number input. This function will straighten the turret to vertical position and rotate the camera Watson to the target position.

6.3 Camera Mast Functions

Automatic Control:

This feature can be found in the Arm&Mast interface under the ”Mast Automat” tab. It can also be accessed by using the “**MastCamXYZ**” or “**MastCamSpherical**” command depending on the desired coordinates. In order to utilize either of these options, it is necessary to provide three input coordinates for position of the target.

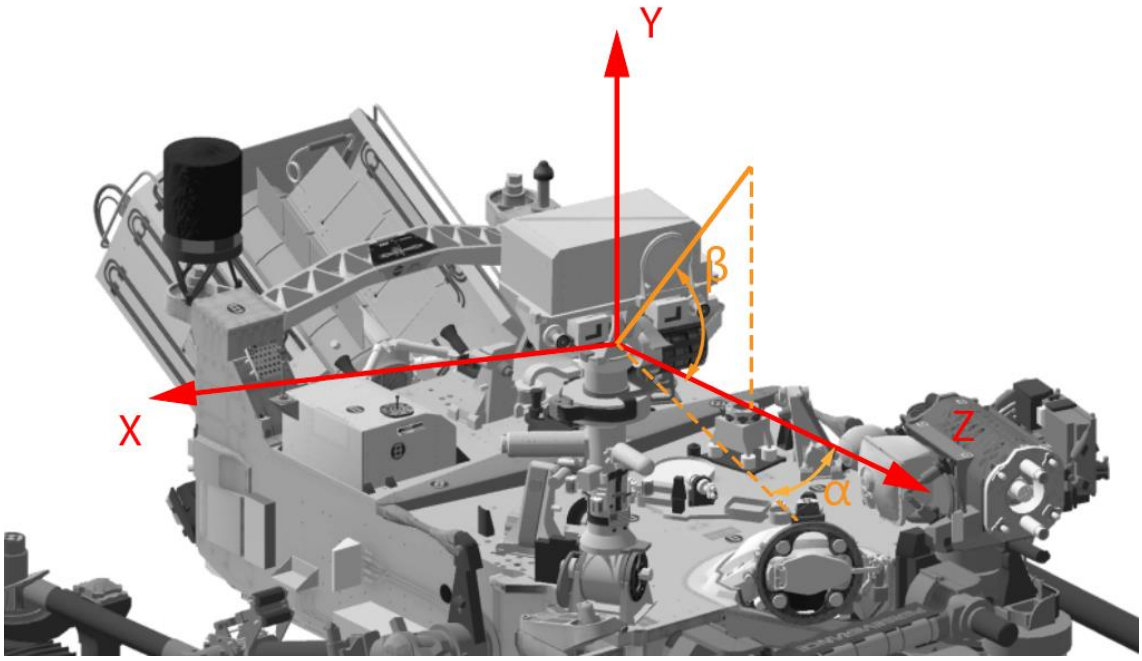


Fig. 20: Cartesian and spherical coordinates for mast automatic control

7 Website

For more information about connecting to an application using Python, MATLAB, C#, Java or Unity, a web page has been created with this link: http://pokrok.ksa.tul.cz/Rover-Perseverance/

You can also use this link to download the Rover Perseverance application.