Mobile Robot TwIRTee

REMOTE CONTROL FUNCTION (RCF)

AND

HUMAN MACHINE INTERFACE (HMI)

Version – 1.0

Date: 24/03/2016

MODIFICATION and EVOLUTION HISTORY:

Revision	Date	Author	Description

REFERENCE DOCUMENTS:

Name of the document	Date	Revision	Author	Description
REMOTE CONTROL AND DISPLAY FUNCTIONS Preliminary Specification	**23/10/15**	**1**	**Eric JENN and Arnaud DIEUMEGARD**	
REMOTE CONTROL AND DISPLAY FUNCTIONS Preliminary Specification	**07/01/16**	**2**	**Eric JENN and Arnaud DIEUMEGARD**	

DEFINITIONS and ACRONYMS:

Acronym	Meaning
TwIRTee	Three-Wheeled Integrated Rover Test bench for Equipment Engineering
RCF	Remote Control Function
HMI	Human Machine Interface
USB	Universal Serial Bus
ARINC	Aeronautical Radio, Incorporated
UDP	User Datagram Protocol

Table of Contents

1 INTRODUCTION 4
2 SCOPE 4
3 REQUIREMENTS 4
4 HIGH LEVEL DESIGN SUMMARY 5
4.1 JAVA CLIENT 6
4.1.1 TASK DEFNITION 6
4.1.2 UDP FRAME STRUCTURE 7
4.1.3 JNI COMMUNICATION STRUCTURE 8
4.2 HUMAN MACHINE INTERFACE 10
4.2.1 BUTTONS 12
4.3 BEHAVIORAL CODE 13
4.3.1 SCOPE 13
4.3.2 FINITE STATE MACHINES 14
4.4 SIMULATOR 19
5 TESTING 20
6 FUTURE IMPROVEMENTS 20

1 INTRODUCTION

The Mobile Robot TwIRTee project is to remotely control a rover that represents a airplane in a taxiing phase.

The project provides functionalities to control the rover using a joystick and monitor its location visually in real time.

This document describes the process followed to develop this application and test its functionalities.

2 SCOPE

The scope of this document is to present the detail design of the application developed.

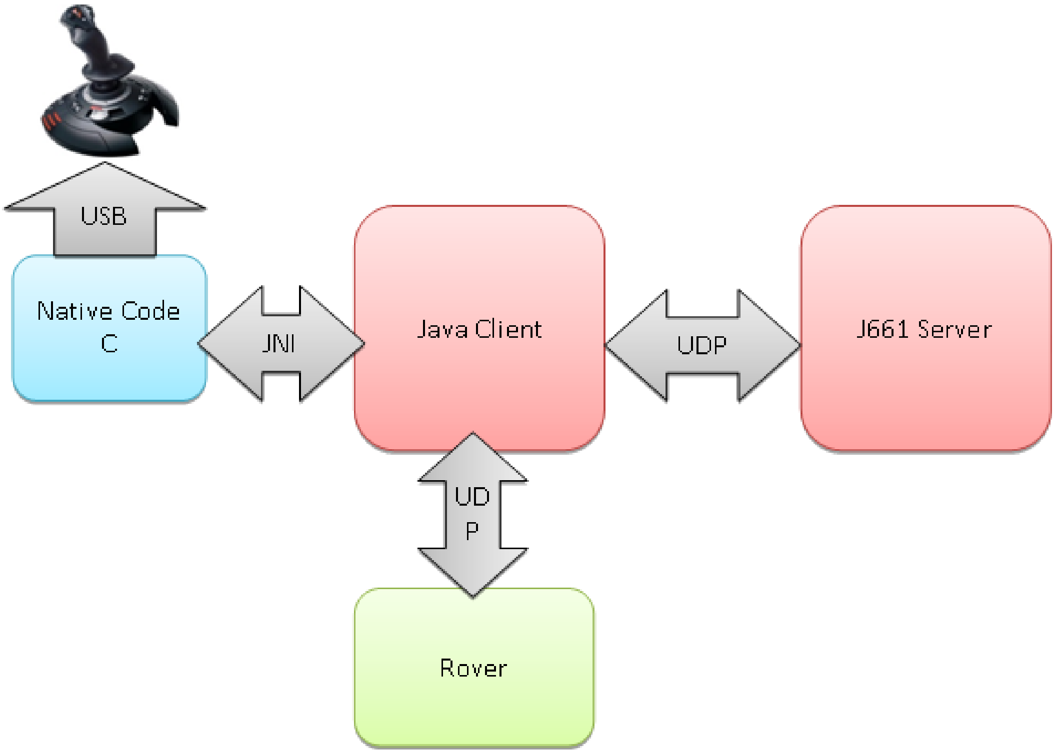
3 REQUIREMENTS

The following are the modules to be developed:

1. Development of the Human Machine Interface
2. Development of Java Client
3. Development of Behavioral Code
4. Development of a Simulator

4 HIGH LEVEL DESIGN SUMMARY

The following figure provides a high-level overview of the information flow and main modules developed.



Main Modules developed comprises of:

1. Java Client
2. Native Client Implementation
3. J661 Server – HMI Interface

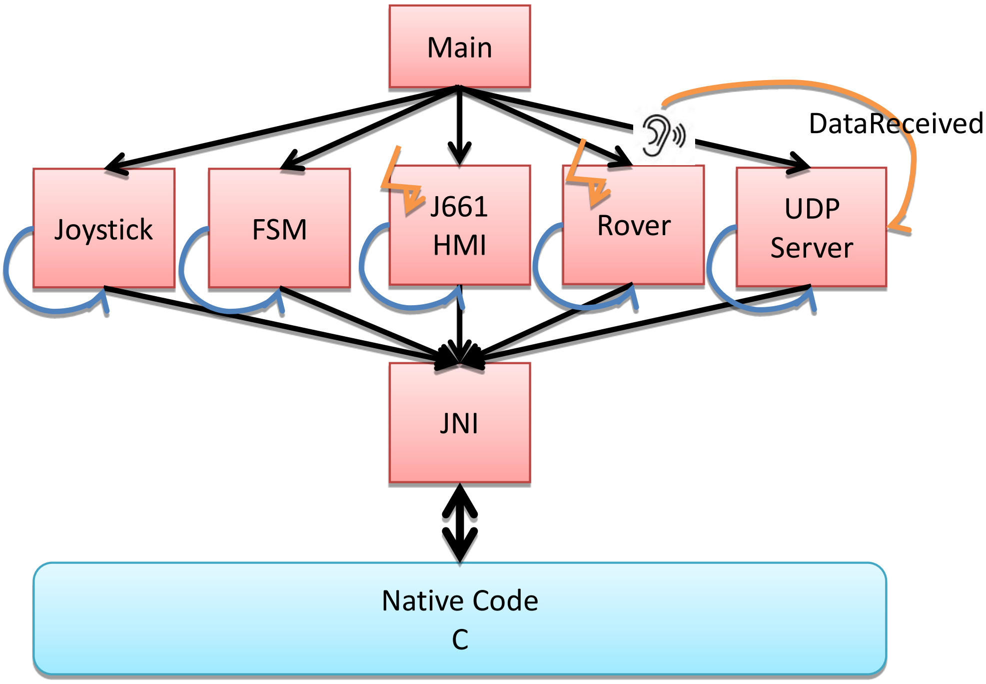
In order to simulate the rover behavior a Simulator has also been developed.

The Communication protocols used are as shown in the diagram:

1. The Java Client and Native Behavioral code communicates using JNI Interface.
2. UDP – protocol is used to establish communication between the Rover, the Java Client and the J661 Server

4.1 JAVA CLIENT

The following figure represents the architecture of the Java Client:



To implement efficient communication, the software implements 5 tasks. The primary consideration for this division is to ensure that related activities are grouped into the same task. Each task performs a distinct activity. This is more efficient as the tasks are simple and independent.

4.1.1 TASK DEFNITION

The 5 parallel tasks are as described below:

- **Joystick Task**: updates periodically the data received from the Joystick
- **FSM Task**: launches periodically the finite state machine that compute the future state of the whole system
- **J661HMI Task**: sends periodically the required data to the J661 server (HMI) to display to the user. This task also listens for events on the buttons on the HMI
- **Rover Task**: send periodically commands to the rover through UDP and requests for rover sensors values
- **UDPServer Task**: listens to the port for any received frame from the rover

Task Name	Description	Timing ms
Rover	Fetches information from UDP	75
Joystick	Fetches information from the joystick	50
J661HMI	Refreshes the information on the HMI	100
FSM	System information is evaluated and communicated both to the Rover and HMI	50
UDPServer	UDP Communication	N/A

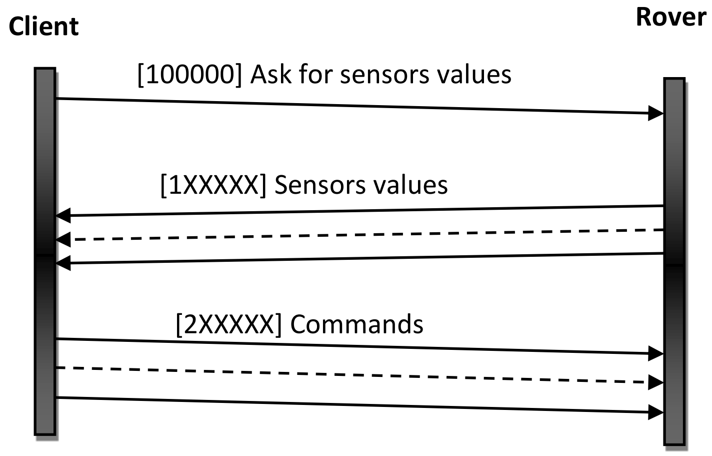
4.1.2 UDP FRAME STRUCTURE

The payload of our UDP message has a length of 6 bytes. The two first bytes are reserved for the frame Id.

Our UDP data can be divided into 3 parts:

- Byte[0] : Table id
- Byte[1] : Element id into Table
- Byte[2-5] : Variable value (float or int)

The communication consists on the following scheme:



List of messages that can be sent through UDP:

Rover Status				
Table	**Element**	**Description**	**Type**	**Direction**
1	0	Ask for Rover status	Void	To Simulator
1	1	Acceleration	Float	To Client
1	2	Speed	Float	To Client
1	3	Speed Left	Float	To Client
1	4	Speed Right	Float	To Client
1	5	X	Float	To Client
1	6	Y	Float	To Client
1	7	Rot acceleration	Float	To Client
1	8	Omega	Float	To Client
1	9	Theta	Float	To Client
1	10	ST_AUTO_ENGAGED	Int	To Client
1	11	ST_REM_ARMED	Int	To Client
1	12	ST_REM_FULL_ACC_ENGAGED	Int	To Client
1	13	ST_AUTO_ARMED	Int	To Client
Rover Command				
Table	**Element**	**Description**	**Type**	**Direction**
2	1	Speed	Float	To Simulator
2	2	Acceleration	Float	To Simulator
2	3	Omega	Float	To Simulator
2	4	Rot acceleration	Float	To Simulator
2	5	CTRL_MODE	Int	To Simulator
2	6	REM_ARM	Int	To Simulator
2	7	AUTO_ARM	Int	To Simulator

4.1.3 JNI COMMUNICATION STRUCTURE

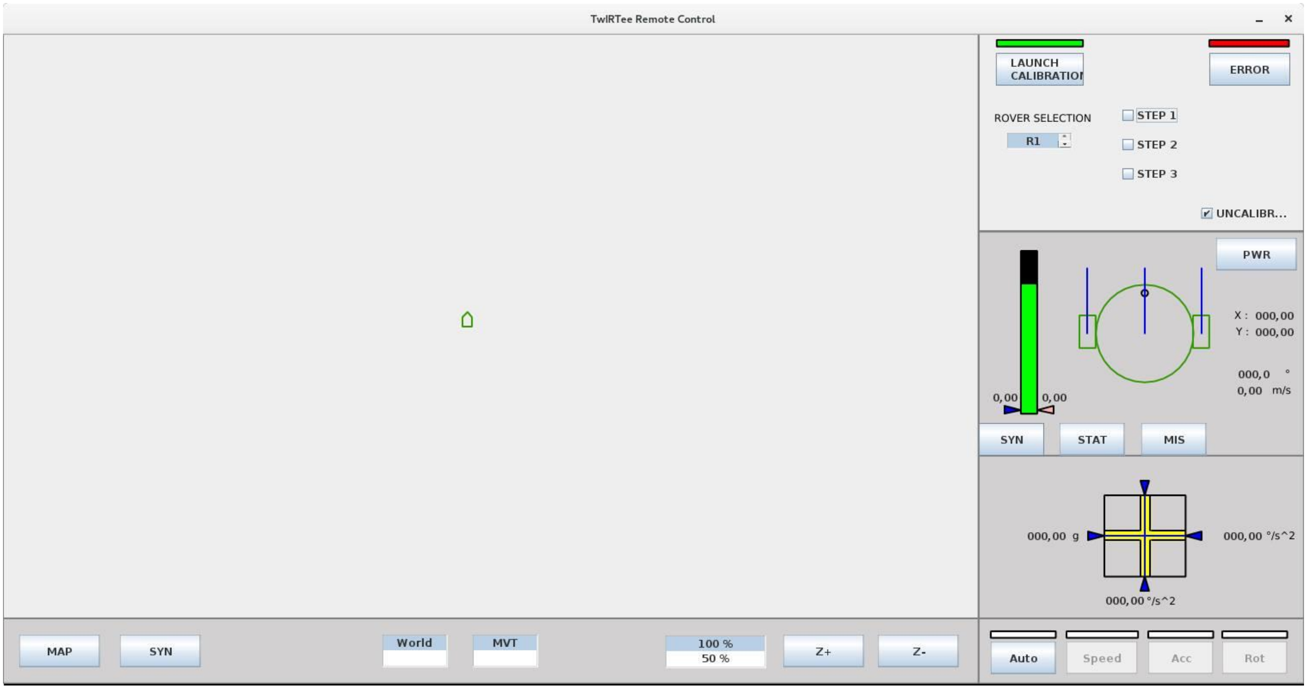
The communication via JNI is mainly done using Structures. One structure is used per application.

List of data exchanged:

Joystick					
Id	Structure	Name	Type	Direction	Comment
100	joystick.	stick_x	float	C to JAVA	Values: -32767 to +32767
101	joystick.	stick_y	float	C to JAVA	Values: -32767 to +32767
102	joystick.	stick_z	float	C to JAVA	Values: -32767 to +32767
103	joystick.	slider	float	C to JAVA	Values: -32767 to +32767
104	joystick.	small_x	float	C to JAVA	Values: -32767 to +32767
105	joystick.	small_y	float	C to JAVA	Values: -32767 to +32767
Rover					
Id	Structure	Name	Type	Direction	Comment
200	twirtee.	acc	float	Both	Linear acceleration (g)
201	twirtee.	v	float	Both	Linear speed (m/s)
202	twirtee.	vl	float	Both	Left wheel speed (m/s)
203	twirtee.	vr	float	Both	Right wheel speed (m/s)
204	twirtee.	x	float	Both	Position (m)
205	twirtee.	y	float	Both	Position (m)
206	twirtee.	rotacc	float	Both	Rotational acceleration (m/s^2)
207	twirtee.	omega	float	Both	Rotational Speed (rad/s)
208	twirtee.	theta	float	Both	Rover heading (rad)
209	twirtee.	ST_AUTO_ENGAGED	int	Both	
210	twirtee.	ST_REM_ARMED	int	Both	
211	twirtee.	ST_REM_FULL_ACC_ENGAGED	int	Both	
212	twirtee.	ST_AUTO_ARMED	int	Both	
213	rover_cmd.	v	float	C to JAVA	Linear speed (m/s)
214	rover_cmd.	acc	float	C to JAVA	Linear acceleration (g)
215	rover_cmd.	omega	float	C to JAVA	Rotational Speed (rad/s)
216	rover_cmd.	rotacc	float	C to JAVA	Rotational acceleration (m/s^2)
217	rover_cmd.	Ctrl_Mode	int	C to JAVA	From 0 to 3: ACC, SPEED, ROT, STOP
218	rover_cmd.	REM_ARM	int	C to JAVA	
219	rover_cmd.	AUTO_ARM	int	C to JAVA	
HMI					
Id	Structure	Name	Type	Direction	Comment
300	hmi.	REM_AUTO_state	int	C to JAVA	From 0 to 3: AUTO_E, REM_A, REM_E, AUTO_A
301	hmi.	REM_AUTO_pressed	int	C to JAVA	
302	hmi.	REM_CTRL_MODE	int	C to JAVA	From 0 to 5: ACC, SPEED_A, SPEED_E, ROT_A, ROT_E, STOP

4.2 HUMAN MACHINE INTERFACE

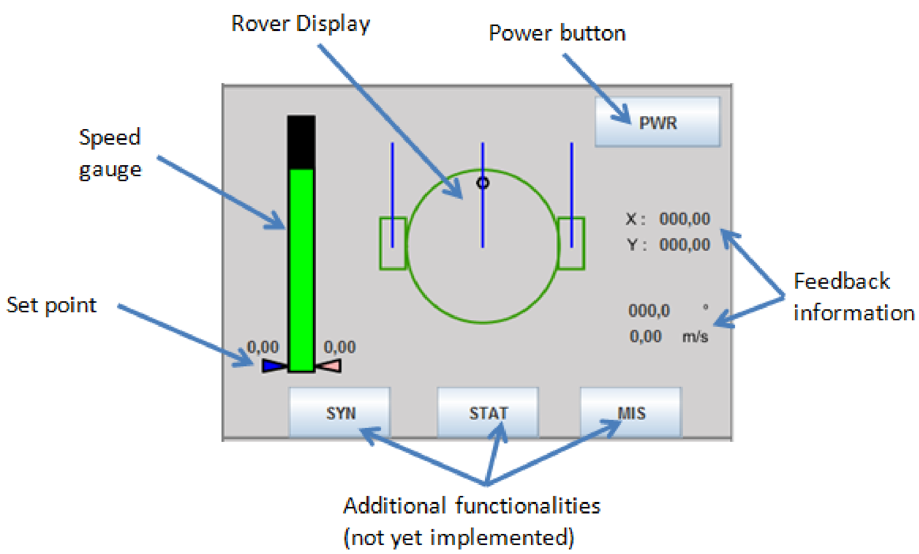
The objective was to provide a user-friendly interface developed on the guidelines of ARINC 661 standard. To realize this, J661 toolset has been used. The Application is Java based.



The main features Included in the HMI are:

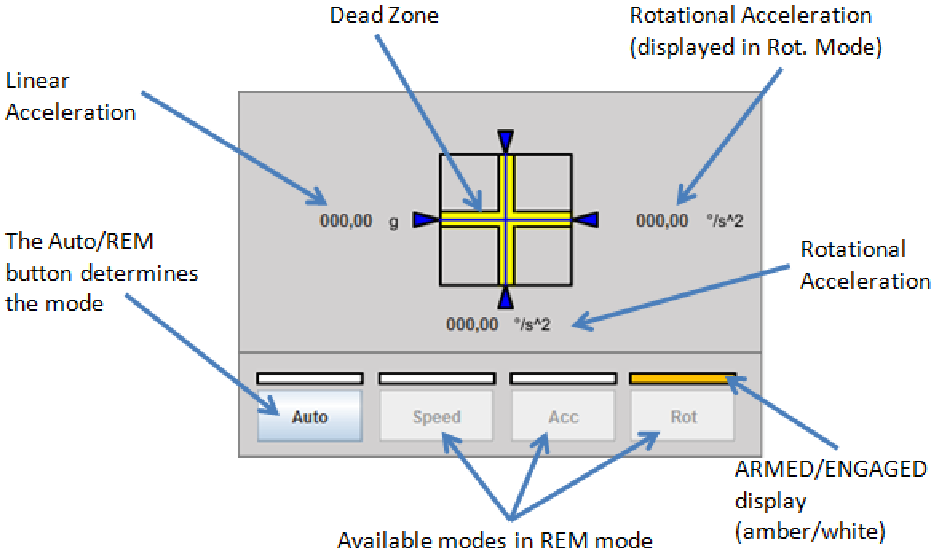
- Map (functionalities not implemented)
- Rover states

  This interface shows the angular position, speed and the coordinates of the rover in a graphical way.

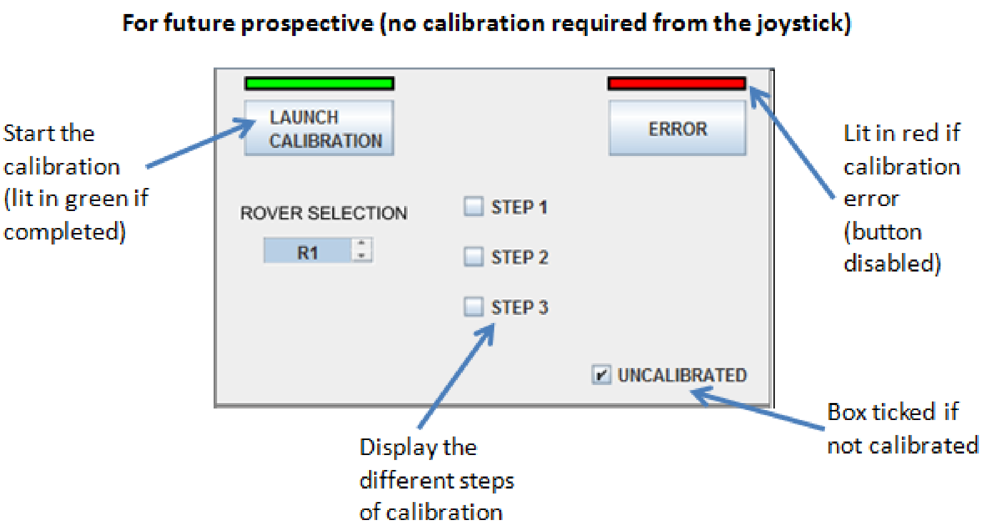


- Joystick states

This part displays the acquisition of the joystick movement and the states (AUTO/REM) of the rover when buttons are pushed. The interface can also rotate when the user switch in rotational mode.



- Calibration interface



4.2.1 BUTTONS

The HMI comprises of 4 buttons. These are event driven.

6. The AUTO/REM button - is an active button for the usage of the pilot
7. The ROT button – deactivated button indicates status of the RCF application
8. The SPEED button – deactivated button indicates status of the RCF application
9. The ACC button – deactivated button indicates status of the RCF application

4.2.1.1 BUTTON BEHAVIOR

Auto / REM mode Button – It is of type `'button_4_state'`. The states consist of:

a. AUTO_ENGAGED – The button shall display:
   i. A label as REM (next state that can be activated)
   ii. A legend indication in white. (indicating the active engaged mode)
b. REM_ARMED – The button shall display:
   i. A label as REM (next state that can be activated)
   ii. A legend indication in amber. (indicating that a mode change may occur)
c. REM_ENGAGED – The button shall display:
   i. A label as AUTO (next state that can be activated)
   ii. A legend indication in white. (indicating the active engaged mode)
d. AUTO_ARMED – The button shall display:
   i. A label as AUTO (next state that can be activated)
   ii. A legend indication in amber. (indicating that a mode change may occur)

ACC mode display – It is of type `'button_2_state'`. The states consist of:

a. ACC_ENABLED– The button shall display:
   i. A label as ACC
   ii. A legend indication in white. (indicating the active mode)
b. ACC_DISABLED – The button shall display:
   i. A label as ACC
   ii. No legend indication

SPEED mode Button – It is of type `'button_3_state'`. The states consist of:

a. SPEED_ENGAGED – The button shall display:
    i. A label as SPEED
    ii. A legend indication in white
b. SPEED_ARMED – The button shall display:
    iii. A label as SPEED
    iv. A legend indication in amber
c. SPEED_DISABLED – The button shall display:
    v. A label as SPEED
    vi. Absence of legend indication

ROT mode Button – It is of type `'button_3_state'`. The states consist of:

a. ROT_ENGAGED – The button shall display:
    i. A label as ROT
    ii. A legend indication in white
b. ROT_ARMED – The button shall display:
    i. A label as ROT
    ii. A legend indication in amber
c. ROT_DISABLED – The button shall display:
    i. A label as ROT
    ii. Absence of legend indication

4.3 BEHAVIORAL CODE

4.3.1 SCOPE

A Joystick (*Thrustmaster T.Flight Stick X*) shall be used to control the rover.

In order to facilitate this control, we have developed a C code that reads the joystick data from the port (the joystick is connected via USB port to the computer).

The behavioral native code basically is responsible for two functionalities:

1. To mirror the states received from the Rover onto the HMI
2. To facilitate information received from the joystick to be communicated to the Rover and to reflect the correct status (i.e. the speed, acceleration, rotational axis and set-points) to be displayed onto the HMI.

The java code updates the HMI. It communicates through JNI (Java Native Interface) with the behavioral code to obtain the joystick or rover state data.

4.3.2 FINITE STATE MACHINES

4.3.2.1 APPLICATION BEHAVIOR

The behavior of the application can be detailed with the help of Finite State Machine Diagrams.

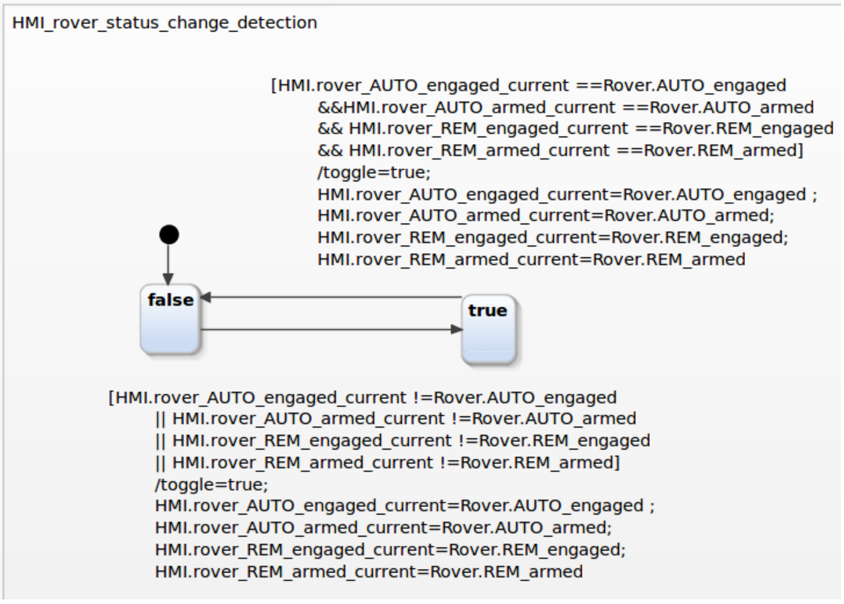


Figure 1 Rover status change detection and HMI update

We read the rover states values and compare it to the HMI copy of the rover state values (from previous cycle). If there was a change, we generate a toggle. In both cases we update the HMI copy of states with the current rover state values.

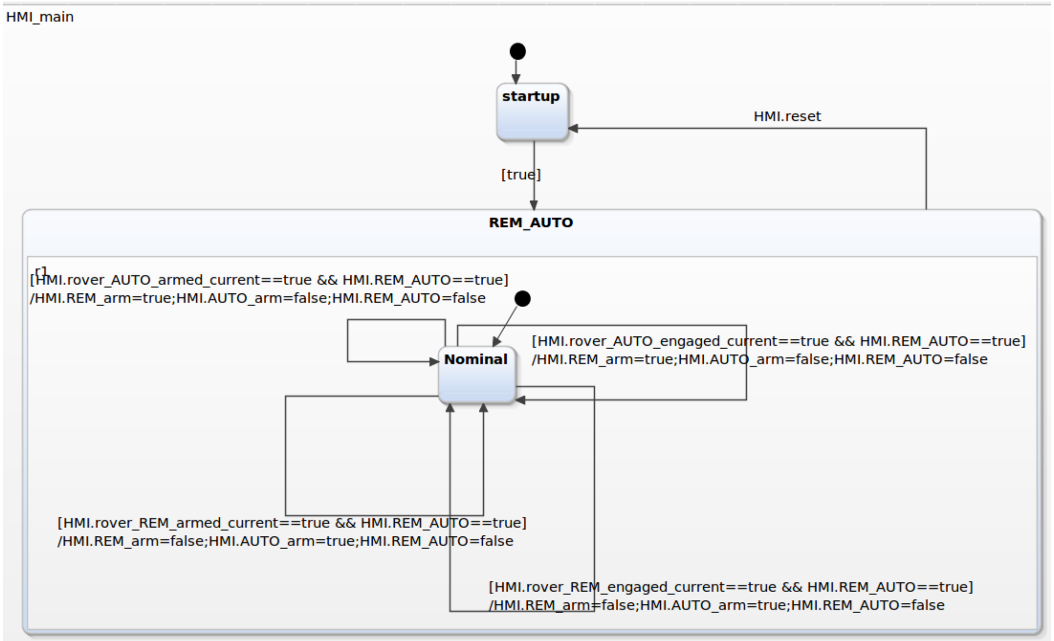


Figure 2 Generate requests for the rover

If the button push flag is true (REM_AUTO is true), we generate an appropriate request according to the current HMI state values.

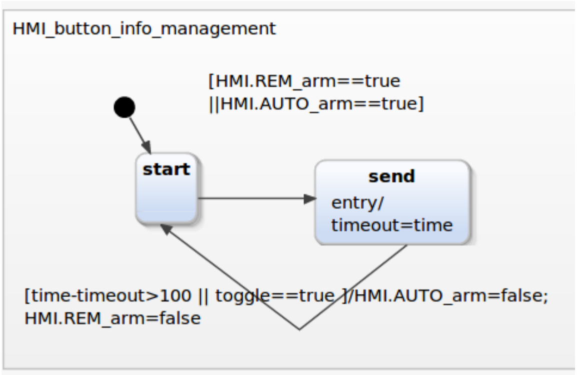


Figure 1 Reset of requests after state change or timeout

Each time a request is generated (REM_arm or AUTO_arm becomes true), we start a timer. If the timer reaches the timeout value, or if we detect a state change of the rover (meaning the rover has responded to our request), we reset all requests.

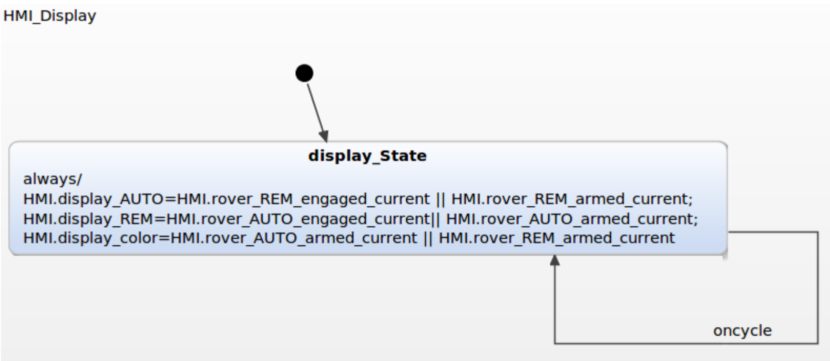


Figure 2 Update the display of the HMI

We display the name of the state which can be reached, and also the color amber if we arm in arm state.

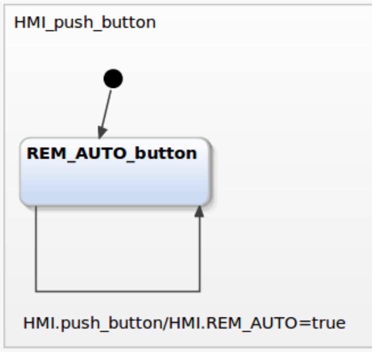


Figure 3 Push button behavior for simulation

Simulation of a push button event, that raises a flag (REM_AUTO is true). It is the behavior we actually have with the J661 interface, which generates an event when there is a click on the REM/AUTO button (the event is generated when the user releases the button of the mouse after clicking). We then process this event and generate a flag.

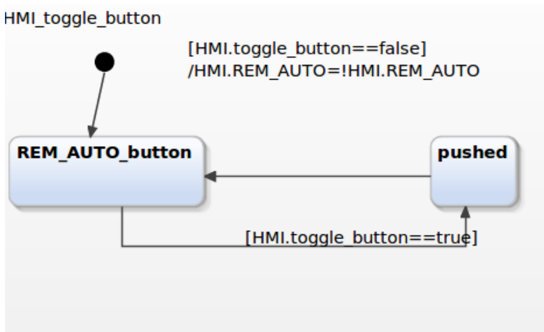


Figure 4 Toggle button behavior for simulation

Simulation of a toggle button. We force the creation of an event, the full event is the user should toggle the button, then toggle back, which will raise the flag REM_AUTO becomes true. Can be implemented if the interface used is a toggle button instead of a push button.

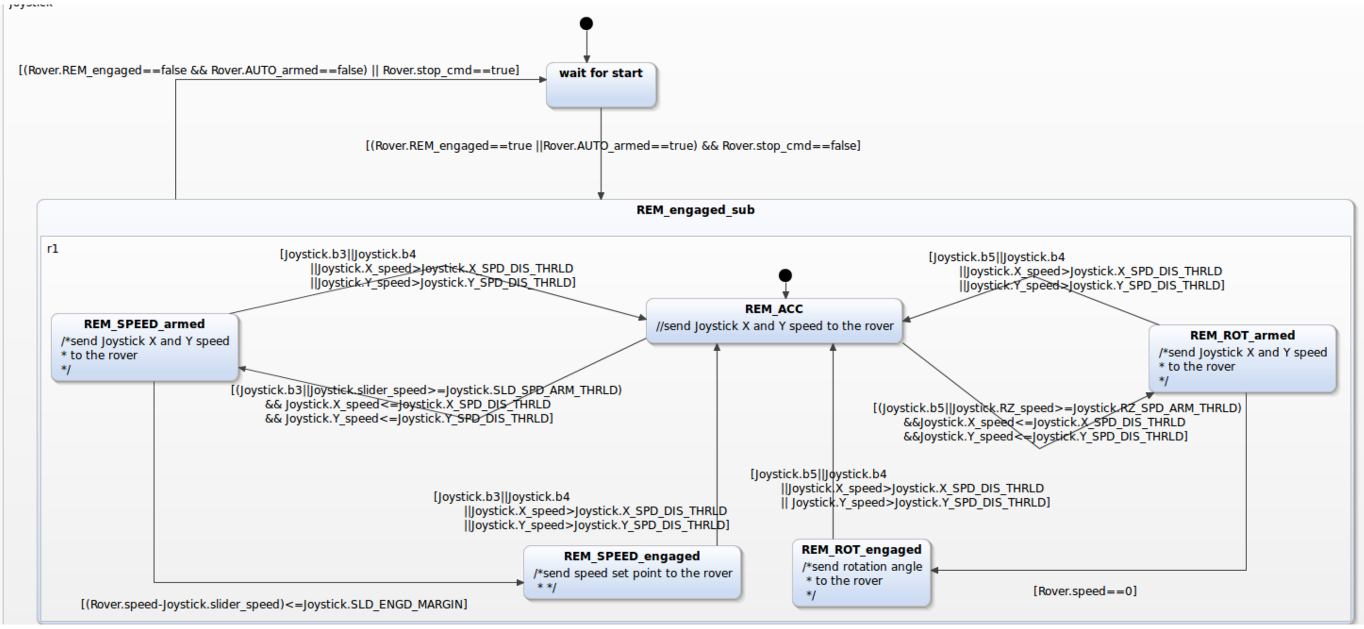


Figure 7 Joystick behavior in REM mode

When the rover is in REM_engaged or AUTO_armed, we are in the REM control mode. In the REM control mode we have 3 type of possible control: ACC (initial), SPEED, and ROT. The transitions between states are conditioned by button presses, axis position, and rover speed.

-In REM_ACC, REM_SPEED_armed and REM_ROT_armed we are in ACC control mode (we send joystick X and Y acceleration to the rover).

-In REM_SPEED_engaged we are in SPEED control mode, we send only a speed set point to the rover.

-In REM_ROT_engaged we are in ROT control mode, we send the RZ axis acceleration to the rover.

4.3.2.2 ROVER SIMULATOR BEHAVIOR

To facilitate the simulation of the state behavior during the design phase we created a simulator of the rover behavior.

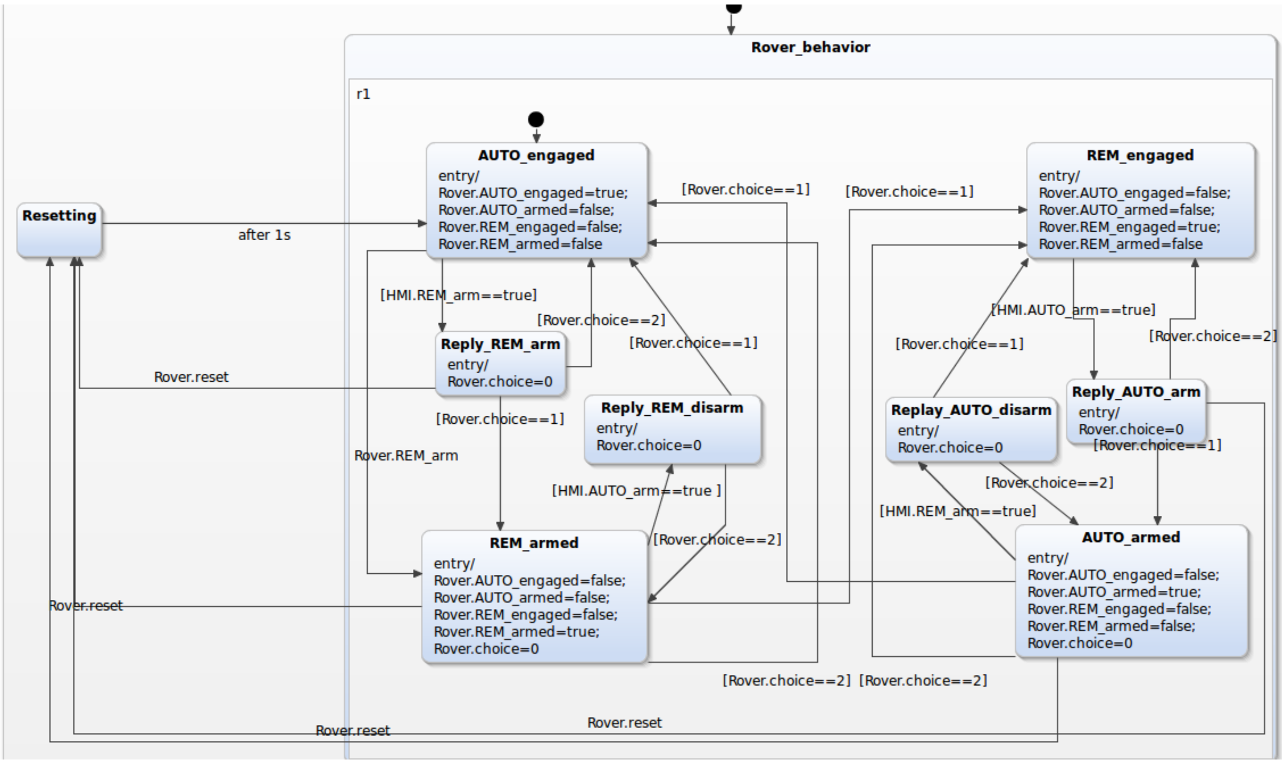


Figure 8 Rover simulator behavior (only state changes, does not include physical simulation)

We have here the 4 possible states of the rover with the conditions to change the state. The user requests a state change and the rover can accept the change or not. There is also a reset that is possible and that will revert back the rover to its initial AUTO_engaged state.

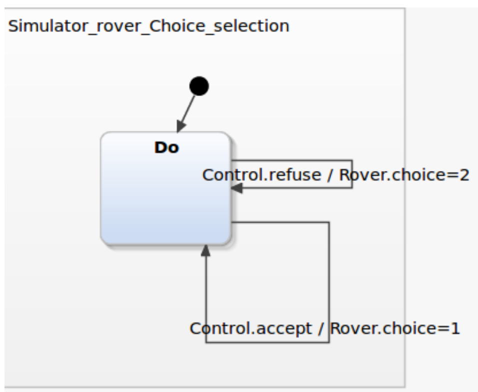


Figure 9 Simulation of rover acceptance or refusal of requests for state changes

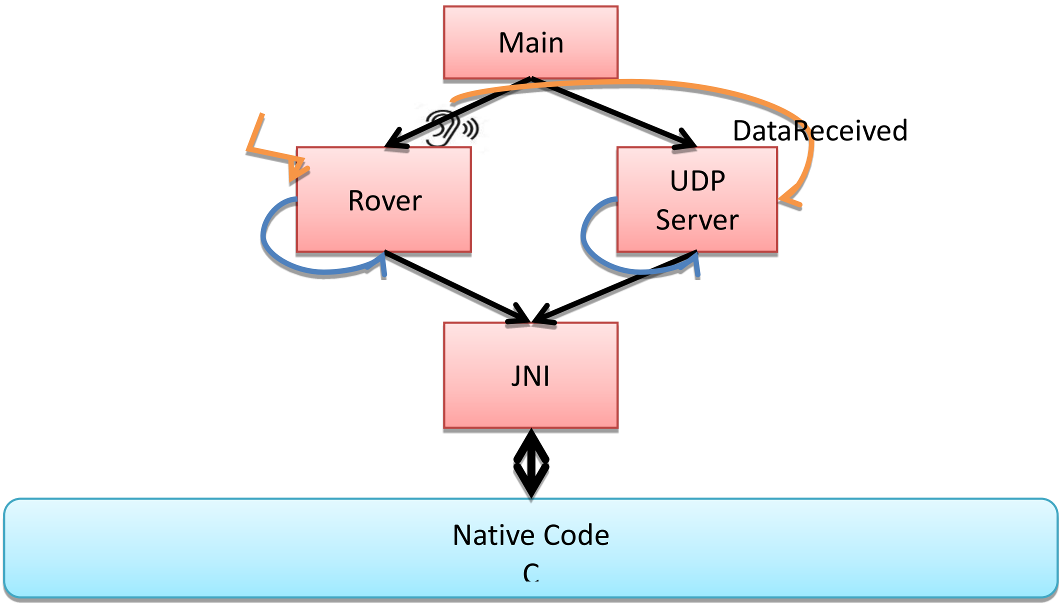
The rover choice selection is just used for simulating the rover's reactions to our commands (it simply changes its state or not, which can be seen on the HMI as we simply read the status of the rover and compare it to the stored status in the HMI).

4.4 SIMULATOR

As the application needs to communicate with the rover, which is not yet available, a simulator has been developed to test the behavior of the different components involved.

The Simulator basically mirrors the communication criteria of the Rover. It receives and sends Information via UDP. Then it transfers the received data to the C part, which contains the simulator of the rover.

The simulator has the following architecture:



This has facilitated behavioral, integration based testing of the application developed.

The simulator implements the kinetic equations in order to simulate a probable reaction to commands. These equations allow calculating absolute position of the rover and speeds in order to test such displays on the HMI.

The simulator computes values according to international units system. And takes many parameters defined in the header file, such as maximum speeds and acceleration, and also the distance between its two wheels.

It allows also to slowly reach a set point such as in a real system.

The simulator is driven can be driven in acceleration or in speed control. It also needs a rotational and linear command. Also it implements a STOP mode that allows to simulate the robot stopping due to frictions.

It also answers states changes required by the state machine. It is possible to tweak the answers to simulate a loss of communication or the state changing refusal.

5 TESTING

We did operational scenarios testing which can be found inside the separate document ‘*ScenarioTesting.ods*’

6 FUTURE IMPROVEMENTS

1. Testing on the real rover may help refine the latency and timing related scenarios
2. Introduction of the choice of the rover and **idle state** mode for the inactive rovers
3. Saving the last status (i.e. previous state)of the Rover and HMI in a configuration file to facilitate continuity
4. Tracking of Rover on a map element in the HMI
5. Display of real-time images that are obtained using the camera