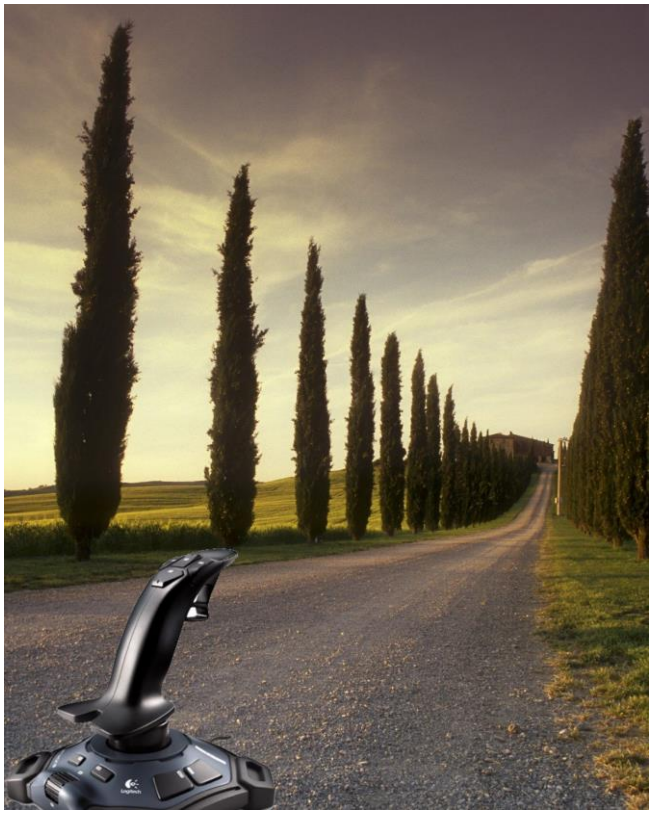
15/8/2014



MISSION CONTROL SOFTWARE DESIGN

Mars Society India (MSI) has been developing technologies in the field of space exploration and robotics. We have developed our very own Mars rover prototype with amazing terrain traversing and sample retrieval capabilities. This document presents the design of a mission control software made by IITB rover team to control this semi-autonomous rover.

Ashish Charan Tandi
Mission Control Head

Contents

Rover: A vehicle, used especially in exploring the terrain of a planet and its satellites

Joystick: An electronic device with one or more sticks that can be moved in several directions to control the movement of any electronic object

UART is usually used for serial communications over a computer or peripheral device serial port.

Telemetry: A process by which data collected from instruments located at remote or inaccessible places are transmitted for monitoring, display, and recording.

Requirements

Design a program to facilitate wireless control of a semi-autonomous Mars rover prototype for a simulated Mars mission

User requirements

The program must

- Be stand-alone (must not need any external dependency to work)
- Easy to use (minimum configuration to start using it)
- Work on a normal personal computer

Technical requirements

The program

- Must take input from a Joystick/Gamepad device and use it to control the rover's mobility and robotic arm's actions
- Must send command and receive telemetry data from the rover using a UART (Universal Asynchronous Receiver/Transmitter) compatible communication module.
- Must track the rover using the feedback from a GPS module mounted on the rover
- Must visualize roll, pitch, yaw using feedback from an IMU(Inertial Measurements Unit) mounted on the rover
- Should be able to plot rover parameters like voltage in the batteries, temperature using the telemetry data coming from rover
- Must work on a windows or Linux system with i3 2nd generation processor or greater processor

Concept Design

Programming Language

C++

- It's a fantastic language for speed and ease of use
- Open source libraries available to do almost anything
- Everyone in the IITB rover team are familiar with it

Framework

QT

- Ample documentations available online and offline
- Cross platform (multiple targets & user sectors)
- UI & backend development can be autonomous

Modules

UART

- QSerial - QT module for serial communication

2D GRAPHICS

- QGraphicsScene - provides a surface for managing a large number of 2D graphical items.
- QGraphicsView - provides a widget for displaying the contents of a QGraphicsScene.

External Modules

JOYSTICK/GAMEPAD

- Gamepad library by - Alex Diener *http://www.sacredsoftware.net/*

GRAPH PLOTTING

- QCustomPlot - widget for plotting and data visualization *http://www.qcustomplot.com/*

System Overview

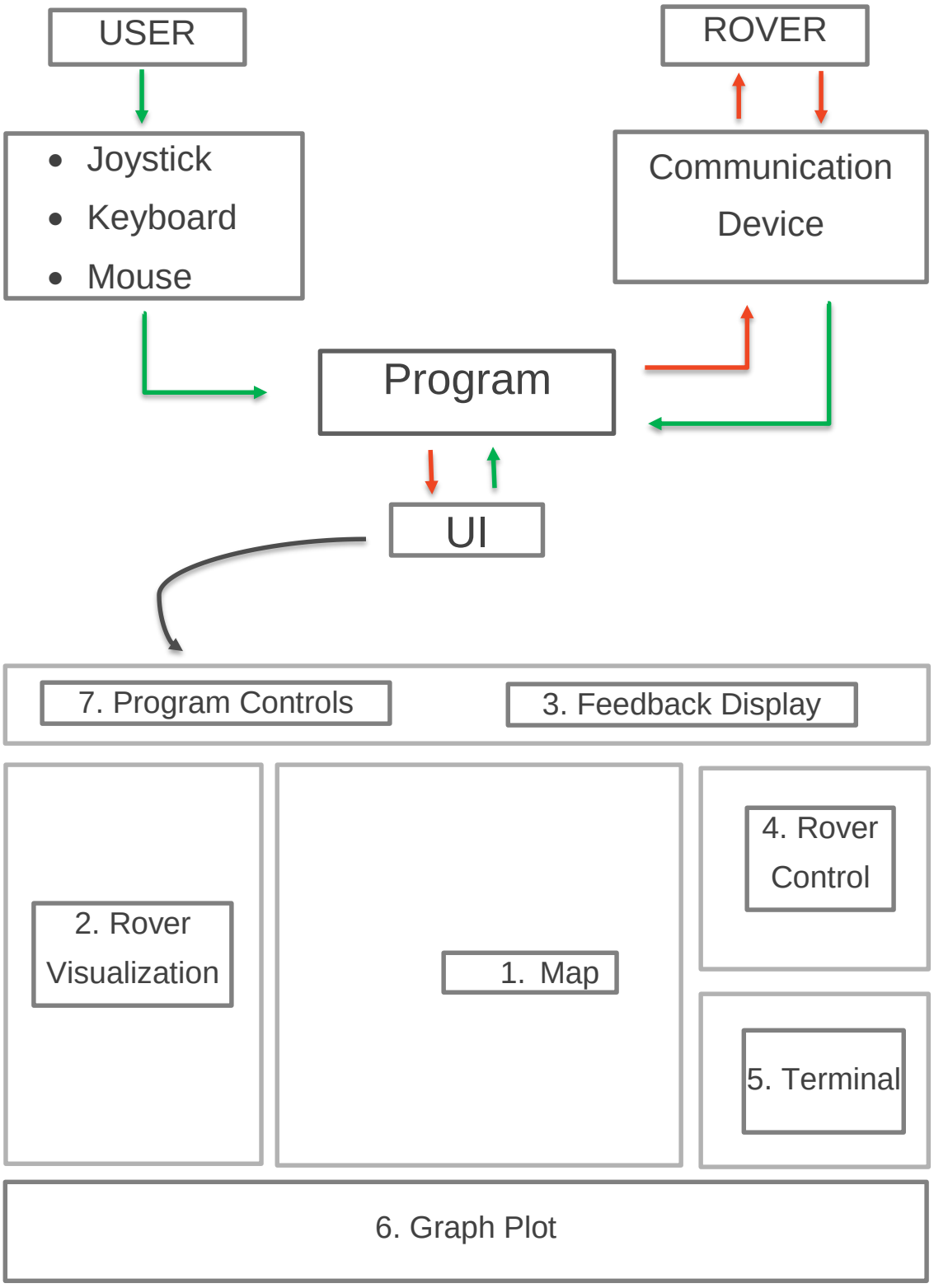
The program takes input from joystick, Mouse, Keyboard; processes them and sends relevant command to the rover

It updates the UI (User Interface) with the data received from the rover through communication module

Program itself gets modified by the input from the UI

The state of the rover gets modified according to the commands sent from the program

…as a result the user controls the ROVER

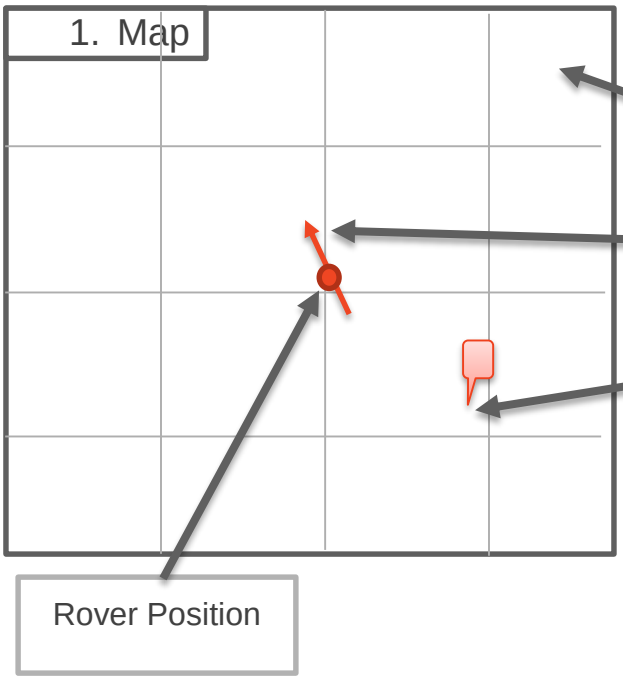


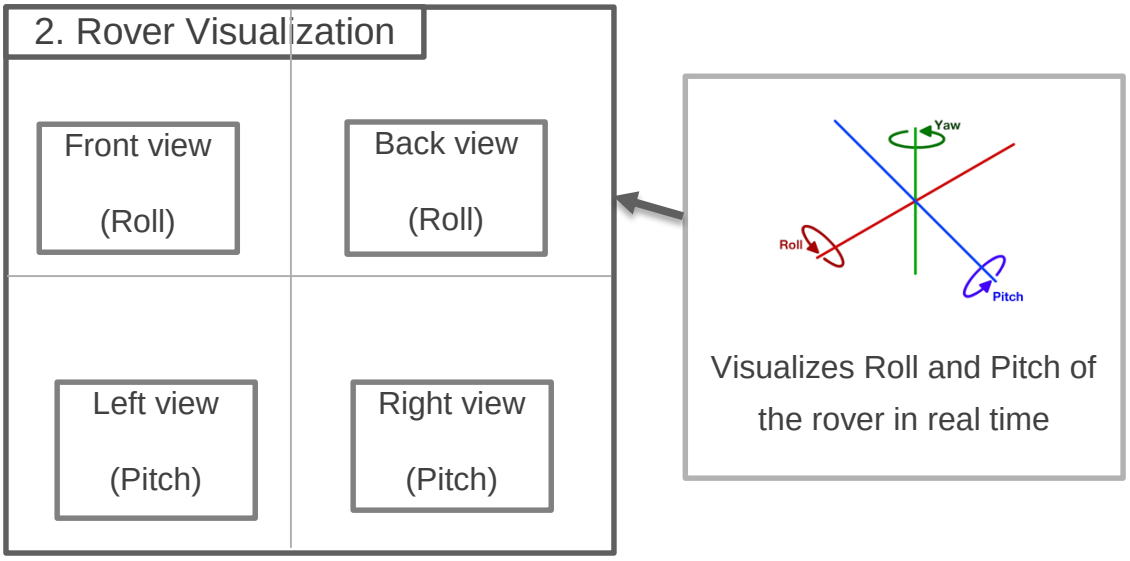
System Overview

Map: collection of tiles stitched together according to the Latitude and Longitude of the view area

Magnetometer data is used to orient the rover towards the target

IMU data is used to update the orientation of the rover in 4 orthographic view





System Overview

Command: controls drive, robotic arm and camera motion

Telemetry: GPS, IMU, Battery status and position encoders

Terminal is configurable to display selected type of messages like – joystick values, serial port input/output, and robotic arm control data

Program control has an option to make a widget visible/invisible

4. Rover Control

Command | Telemetry

Displays all telemetry data from the rover

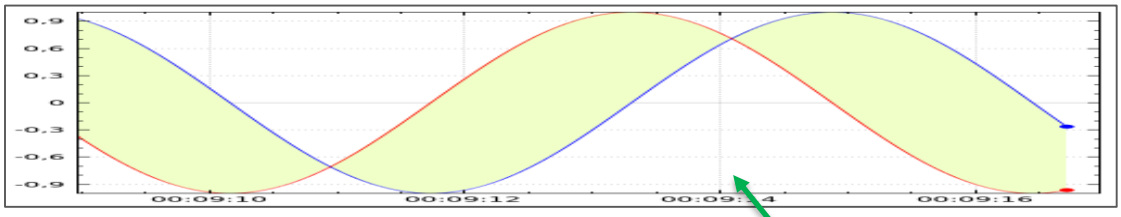
Controls what commands are being sent to the rover

5. Terminal

- Message 1
- Message 2
- Message 3

All the program messages displaced here

6. Graph Plot



Data received from telemetry is sent to this graph to monitor

7. Program Controls

Controls tools for the program

- Connects/disconnect to Serial port
- Choose Baud Rates

Detailed Design

Program Architecture

We have used the model-view architecture. The controller and view object are combined. All the data related to rover and the program itself are put together in a single object called rover.

Main Program

Main program starts by creating a QApplication, creating a standard QT MainWindow object and displaying it using the show() function. When the MainWindow object loads, it connects all the Signals to their respective Slots. The program is event driven hence flow is taken care by the events generated, as QApplication manages the events.

Events

This being an event driven program. These are the crucial events.

User generated events:

- Joystick value changed
- New joystick found / joystick removed
- Centre map to rover GPS position
- Open/Close a serial port
- Key pressed or released
- Mouse right/left clicked or scrolled
- Save rover's path (as Latitude and Longitude)

Programmatically generated events:

- Timer Event
- Read data from serial port
- Decode data received
- Encode data required for rover
- Write data to serial port
- Update rover GPS values
- Update rover IMU values
- Handle serial port error

Class MainWindow

Data Members

- UI:: *ui
- UI:: *control_ui
- URC_Plot *urc_plot
- URC_RoverView *urc_roverview
- QTimer m_timer
- QSlider *mapzoomsslider
- QVector<QString> terminalString
- SettingsDialog *settings
- QSerialPort *serial
- joystickObject m_joy
- Datas *roverdata
- QByteArray comportdata
- ImageCache imageCache
- QMapWidget *m_MapWidget;

Functions

- explicit MainWindow(QWidget *parent = 0)
- ~MainWindow()
- void printTerminalString()
- void initqmapwidget()
- void keyPressEvent(QKeyEvent *e)
- void keyReleaseEvent(QKeyEvent *e)
- void joyEvent(unsigned int deviceID, bool btnEvent, unsigned int axisNo)
- void joystickFound(int n)
- void toggleTerminal()
- void centerMapToSpot()
- void handleClientConnection()
- void timerEvent()
- void openSerialPort()
- void closeSerialPort()
- void writeData(const QByteArray &data)
- void readData()
- QByteArray encodeData(const int type)
- void decodeData()
- void updateGPSValues(double lat, double lon, int region)
- void updateIMUValues(int roll, int pitch, int heading)
- void saveGPSPathstoFile()
- void handleError(QSerialPort::SerialPortError error)
- void resetAll()
- void loadSettingFile()
- void saveToSettingFile()
- void stopRover()
- void stopArm()
- void createDockWindows()

Class QMapWidget

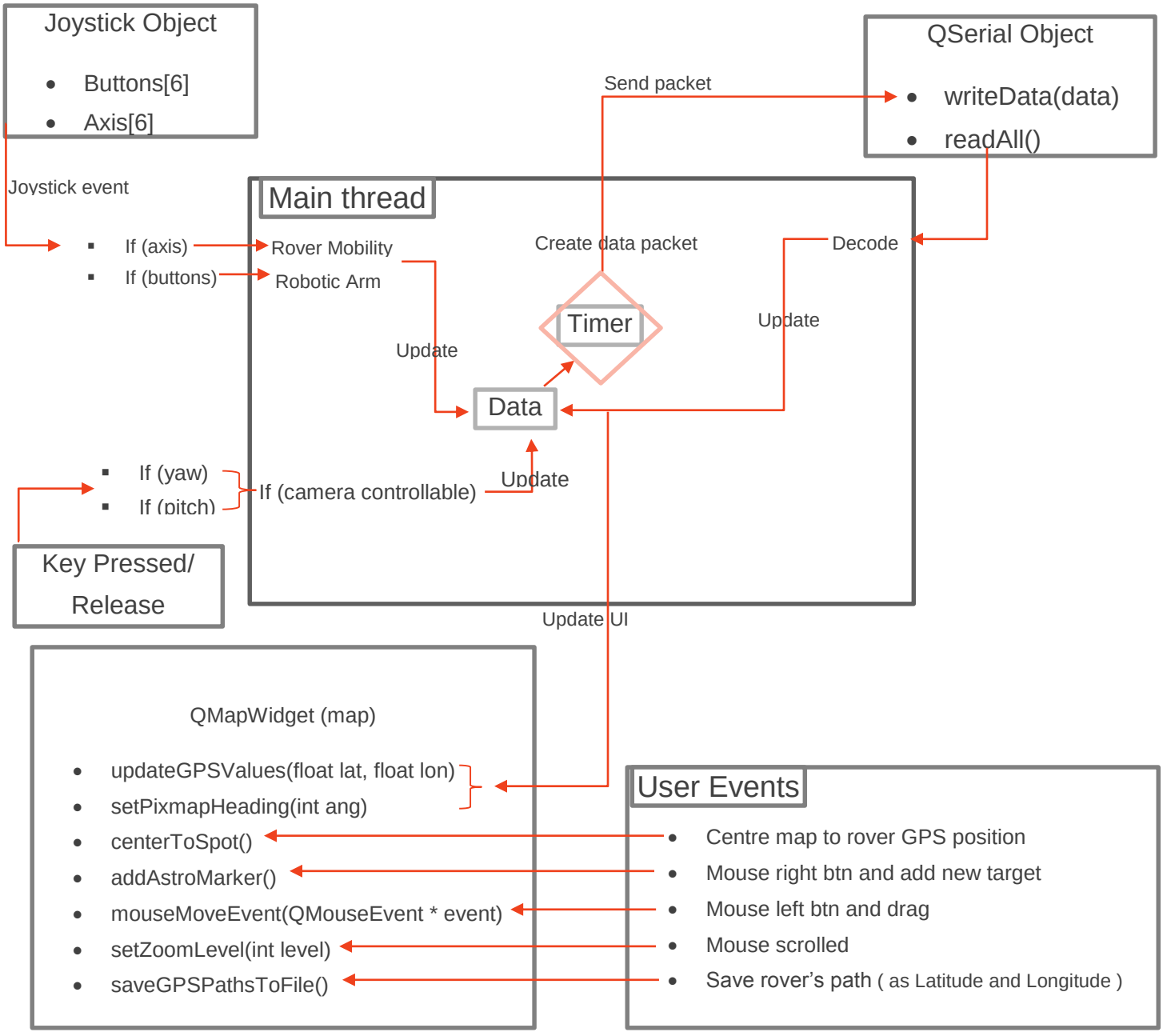
Data Members

- QAction *actionAddMarker
- int astroPixmapSizeInPercentage;
- int zoomLevel
- int zoomlevelText
- int minZoomLevel
- int maxZoomLevel
- qint64 m_ViewportX, m_ViewportY
- int m_MouseX, m_MouseY
- qint64 MyGPS_spotX, MyGPS_spotY
- GraphicsItems gitem

Functions

- void convertLatLonToXY(double lat, double lon, qint64* px, qint64* py)
- QImage *getTileImage(quint32 x, quint32 y, int zoom)
- setGItemPosWrtMapWidget(bool gpspath, int idx, double lat, double lon)
- void updateLocalCordinateSystem()
- void setMyGPSPosition(double lat, double lon)
- void drawPathLegend(QPainter *p)
- void centerToSpot()
- void setPixmapHeading(int ang)
- void setZoomLevel(int level)
- void setCache(ImageCache* cache)
- void addAstroMarker()
- void addNewGPSPath()
- void deleteGPSPath(QAction*act)
- void deleteAstroItem(QAction*act)
- void editGPSPathList()
- void updateGPSValues(float lat, float lon)
- void saveGPSPathsToFile()
- virtual void mousePressEvent(QMouseEvent *)
- virtual void mouseReleaseEvent(QMouseEvent *)
- virtual void wheelEvent(QWheelEvent *)
- virtual void mouseMoveEvent(QMouseEvent *)
- virtual void contextMenuEvent(QContextMenuEvent *event)
- virtual void paintEvent(QPaintEvent *)

Detailed Design



Results

The map has multiple (16) zoom level to navigate, but all map data must be downloaded from a map repository before using

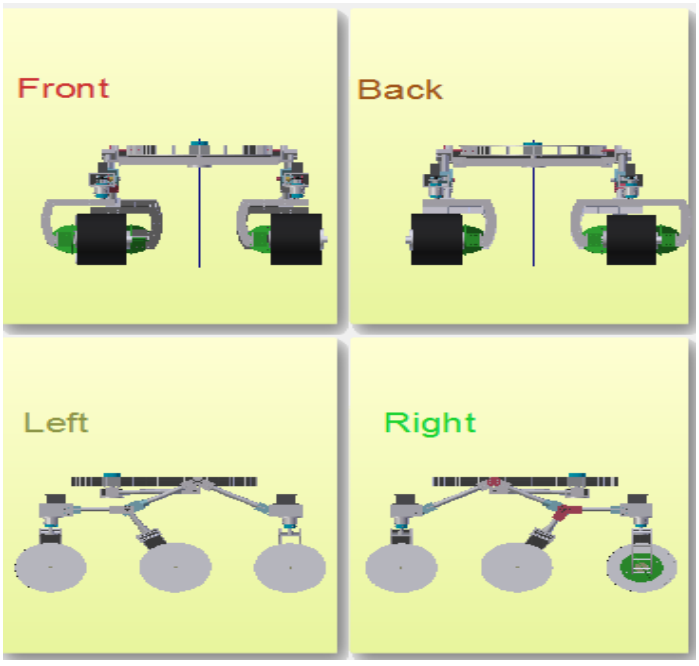
Visualization shows expected yaw and pitch of the rover. As of now it doesn't alert the user about potential danger of toppling

Map

Features

- Adding Target (Latitude, Longitude)
- Magnetometer data integration
- Rover path tracing
- Multiple GPS paths

Rover Visualization



Features

- Pitch, Yaw visualization in orthographic vies
- Rocker and Bogies joint movements independently
- Anti-aliasing on/off
- Dock widget

Results

Change Cache Directory: This command changes the directory of where the program searches for map images

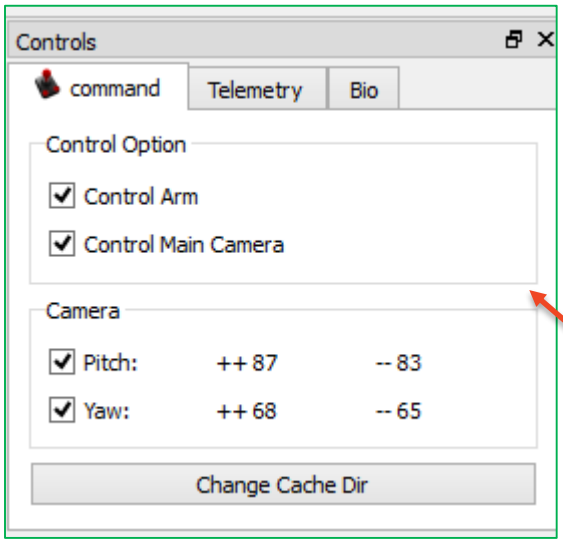
Terminal shows all the low level messages produced by the program, with a facility to stop/start showing them if needed

Feedback Display

Features

- Alert for Joystick, UART compatible devices,

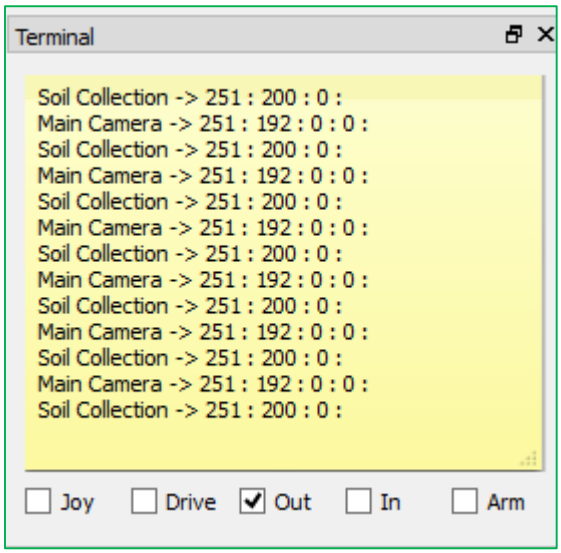
Rover Control



Features

- Arm control
- Camera control
- Change map images directory
- Dock widget

Terminal



Features

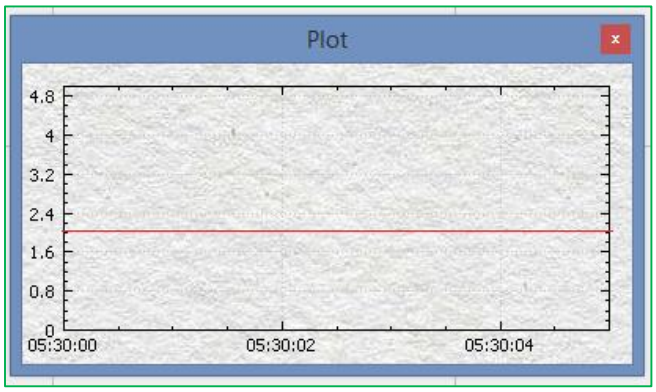
- Shows for raw joystick values
- Command for rover, arm, camera, soil collection assembly
- Subscribe to specific type of message
- Dock widget

Values are plotted against the current time as shown in the operating system, and has nothing to do with the time at which it was sent from the rover

Once a serial port is connected the option for connecting it gets disabled, this ensures safety in communication and make the program reliable

Local echo:
A mode of operation of a communications program in which it displays the characters that enters while communicating with a remote system.

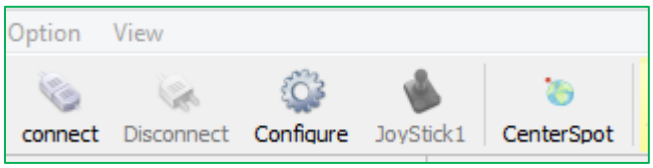
Graph Plotting



Features

- Plot real-time values with operating systems time
- Dock widget

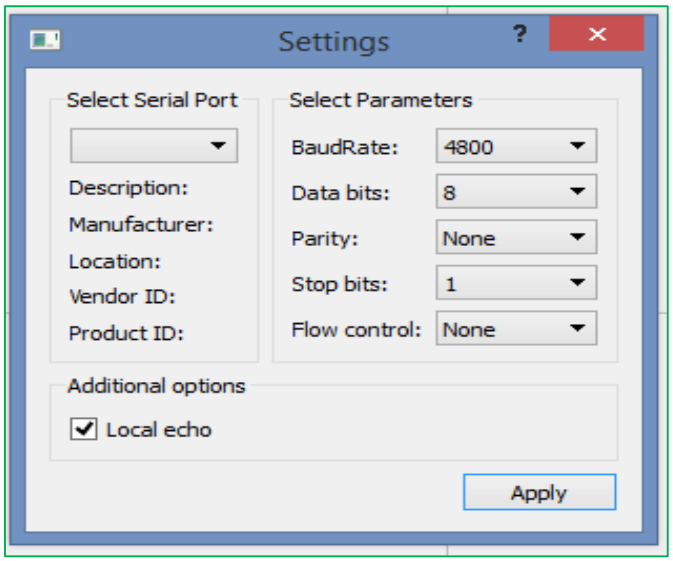
Command Control



Features

- Connect disconnect to a UART device
- Center rover in the map

Serial port configuration



Features

- Connection only to a single device

Future Plans

3D visualization in the main program would bring down performance in terms of controls and communication speed.

3D visualization also can be done in a separate standalone program, but in that case rover telemetry data has to send to the other program using some network protocol.

- **Rover Visualization in 3D**

  As for the current plan the program do not have any 3D visualization element in it. It important to get a 3rd person view of the rover in order to control to a off-road vechile. In future we are interested in integrating Opengl with with the program to see what is the state of the rover in 3D using IMUs and position encoders.

- **Automatic tiled map download from an online repository**

  In order for the program to work with map, we need to have tiled image data downloaded and stored offline at the zoom level we want to use it. In future we would like to have the software download these tiles automatically give a repository.

- **Joystick Calibaration and axis mapping**

  Because we use joystick to control the rover it would be ideal to have a joystick which would have no errors, but unfortunately it's not practical to have a joystick like that. Hence we would like to add a functionality to the program so that we can remove bias in the joystick device

  We should be also able to map different axis for different functionalities without having to rebuilding the program

"Mission control provides actionable access to determining and implementing the most effective "doing" that impact and elevate a mission's performance"

Contact Information

Ashish Charan Tandi
Mission Control Head
act.jnv@gmail.com

Varun S S
Systems Engineer
varunss1993@gmail.com

IITB Rover Team, Software sub-system
Aero Basement Lab,
Department of Aerospace Engineering
http://www.marssociety.org.in/