**Rover ProtoSpace: Telepresence Robot**

**Muhammad Fazlur Rahman**

Robotic Intern – Protospace Toulouse

Email: [fazlur.rahman.ae@gmail.com](mailto:fazlur.rahman.ae@gmail.com)

This rover goal is to be operated as a telepresence robot, part of Glob’ibule Telepresence Robot project by Humanity Lab Airbus. The project goal is to enable children inside an isolation room in hospital to be able to see outside the room by utilizing this robot. A team is formed to work on this project, realizing the rover as a telepresence robot. The team work separated into several topics, which are physical, system, and design. The rover itself can be separated into three main parts, namely rover (lower part), interaction (upper part), user interface (UI).

This internship covers the system part specifically for the rover (lower part). In this document, several sections are provided to have the idea for the current state of the lower part, with brief discussion on the interfacing possibilities with the upper part and UI. The first section is rover design overview, providing the general idea of the rover. Secondly, hardware utilized in this rover is discussed, both for basic configuration and enhanced configuration. Third, an introduction to Robotic Operating System (ROS) is provided as it is used as the main framework for the interfacing. Afterwards, a general idea of companion computer and Arduino used in this project is presented. Lastly, brief explanation of the Graphical User Interface (GUI) is provided.

# Rover Design Overview

A rover is a vehicle designed to move across surface. The rover built in ProtoSpace, shown in Figure 1, utilizes 5 wheels, 2 of them are driving wheel. The other 3 wheels are casters, type of wheel which can turn along z-axis if it is given force on xy-plane. This rover is design by ***Gautier Aidli***, maker in ProtoSpace Toulouse.

Aluminum with 10 mm thickness is chosen as the frame material as it is strong and heavy. As this is a first prototype, a strong frame is required to prevent the rover from being broken. For the heaviness, it is required be heavy as the wheel commonly have a load of a person. Less weight could result to less friction force which can cause slip upon the wheel.

|  |  |
| --- | --- |
|  |  |
|  |  |

Figure . Rover Overview

To control the rover, on a basic mode, remote control and Pixhawk is used. This hardware is discussed later in this document. The main consideration to use Pixhawk is the availability of rover firmware and its firmware safety. Thus, this should be a robust proposition. The other benefit is Pixhawk ability to be interfaced with a lot of modules, to control or to be controlled.

# Hardware

This section explains several basic hardware used in this rover, in which if combined can have basic functions of the rover. The basic functions include controlling the rover by using remote control and connecting it to ground control station (GCS) via mission planner. There are 4 basics hardware, namely Pixhawk, motor and ESC, power system, and remote control. Figure 2 shows the wiring diagram of basic hardware. The enhanced configuration is also discussed in this section, providing information on advanced hardware implementation for the rover.

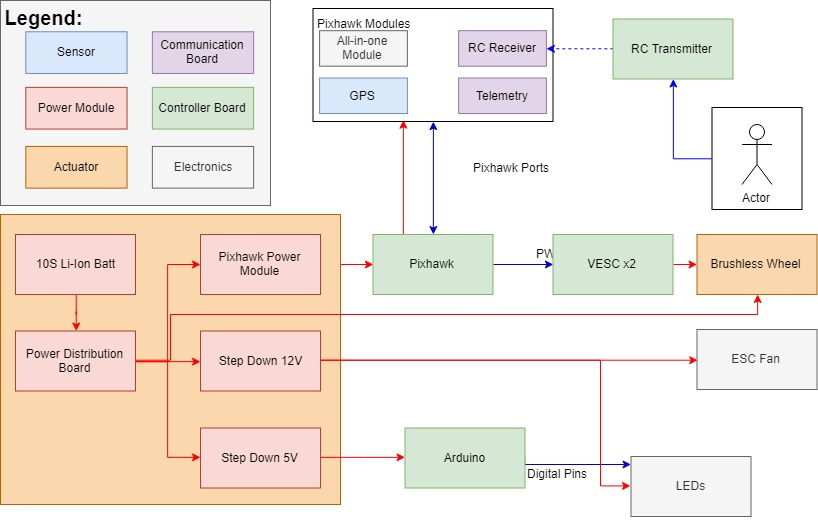


Figure 2. Wiring Diagram, Basic Hardware

## Pixhawk

Pixhawk, as shown in Figure 2, is a dedicated board to control unmanned vehicle such as fixed wing, multirotor, rover, et cetera. It consists of several sensors including 3 axis accelerometers, 3 axis gyroscope, 3 axis magnetometers, and a barometer. An additional sensor can be added to enhance the navigation, such as GPS and rangefinder. With these sensors, one can obtain rover’s orientations, altitude, as well as coordinate (if GPS is included).



Figure 3. Pixhawk Pro 3

On a basic rover, Pixhawk is utilized to handle the motor and ESC movement by taking input from remote control. Pixhawk can also control system’s safety such as activating failsafe when an unwanted event occurs. There are also other parameters that can be configured from Pixhawk firmware, using the GCS.

The firmware used in this project is ArduRover V3.4.2. The saved parameters can be found in <https://github.com/globibule/hardware/blob/master/pixhawk/PixhawkRoverROS.param>. By utilizing this firmware, the rover can maneuver in several movements, shown in Figure 4. The first maneuver is pure translation where the rover moves forward and backward. It happens when rover only receives command in from the throttle and the steering command is in trim position. The second one is pure rotation, where the rover spins around the midpoint between the two wheels, happens when a steering command is given and the throttle command is in trim position. Lastly, turning maneuver, allowing the vehicle to turn by differential speed between two wheels. This maneuver happens when both throttle and steering commands are not in trim position.

|  |  |  |
| --- | --- | --- |
| 1. Translation | 1. Rotation | 1. Turning |

Figure 4. Rover Maneuver

## Wheel and ESC

The motor used in this project is a wheel integrated motor, with rated power of 350W. The operation voltage is between 24V, 36V, or 48V. The wheel has a Hall effect sensor, its diameter is 200mm, and rim size of 8 inches. The speed, based on *banggood* website, is around 12-28 km/h and the wheel should operate under load between 80-150 kg. The wheel is shown in Figure 4.

|  |  |
| --- | --- |
| Figure 5. Wheel with Hall Sensor | Figure 6. VESC |

The ESC used in this project is a version of 50A Brushless 4.12 VESC Controller, shown in Figure 4. One of the benefits of this ESC is that the firmware is programmable through a software called VESC Tool. The ESC can have several modes, including field-oriented control (FOC), which enable the brushless motor to be controlled in a high performance such as full torque at zero speed and high dynamic including fast acceleration and deceleration. Also, it can be used to reverse the wheel spinning direction or to have a brake. Unfortunately, there’s no firmware which allows direction reverse and brake at the same firmware.

## Power System

This rover uses 10S lithium ion battery (*li-ion*), manufactured by LALALA, shown in figure XXX. The recommended voltage for the battery to be used is between 35V to 42V. Lower than that, the battery should be recharged. The other feature of the battery is the output current which can go up to LALA amperes.

|  |  |
| --- | --- |
| Battery Li-ion | Power Distribution Board |
| Figure . 5V Step Down Module | Figure . 12V Step Down Module |

A power distribution board is used to supply power to several components such as Pixhawk power module, 5V step down module, and 12V step down module, as shown in Figure 1. The 5V step-down module, shown in Figure 1, has an input voltage between 6-50V and supply current up to 20A. It can supply a variety of output, ranging from 5V to 9V. The other step-down module, ***LM2596***, has input voltage up to 40V and can supply a variable output voltage up to 37 volts and output load current of 3A. In this case, it is used to supply 12V.

## Remote Control

Remote control transmitter in this project is a Hi-Tec Aurora 9 while for the receiver, Delta 8 is used. This remote control operates at 2.4 GHz which is a common frequency band for remote control. There are two available output of this remote-control receiver or airside, which are PWM and CPPM. For this project, the CPPM output is used since Pixhawk only accept this type of signal as an input. The remote control has up to 8 channels and allocated as shown in Table 1.

Table 1. Remote Control Channels Allocation

|  |  |  |  |
| --- | --- | --- | --- |
| **Channel Number** | **Allocation** | **Channel Number** | **Allocation** |
| Ch 1 | Steering command | Ch 5 | Arming command |
| Ch 2 | Not allocated | Ch 6 | Not allocated |
| Ch 3 | Throttle command | Ch 7 | Control mode |
| Ch 4 | Not allocated | Ch 8 | Not allocated |

## Screen

In the rover, a screen is attached to enable communication with the rover surrounding. This screen is a 7 inches screen, shown in Figure 9, and it has a motor to move the screen up and down, shown in Figure 10. The screen is connected to a Raspberry Pi and is able to show picture from the user camera. There’s also speaker and microphone to enable audio communication with the surrounding. This upper part is done by another team.

|  |  |
| --- | --- |
| Figure 9. Screen on Rover | Figure 10. Motor to Move Screen Up and Down |

For the future development, the motor to control the screen can be connected to an H-Bridge circuit to enable the movement. This H-Bridge circuit should be connected to a 12V power source and can be controlled from Arduino connected to Raspberry Pi. The Raspberry Pi receives the command from the user through Graphical User Interface (GUI).

Another improvement is to combine the Raspberry Pi used to operate ROS as the control hardware and the Raspberry Pi used for this screen. The combination can be done as both applications do not consume a lot of computation sources. Also, the display of the image can be done by working on ROS framework (see <http://wiki.ros.org/video_stream_opencv>).

## Enhanced Configuration

In this enhanced configuration, a modification from the basic configuration is described. The hardware from the rover side doesn’t change a lot except for the control part. The control paradigm here is to enable the rover to be controlled from a daily device such as smart phone, tablet, or computer from distant. Another feature is to enable the rover performing an obstacle avoidance, or at least prevent the robot from colliding with an obstacle.

At this moment, the proposed idea is to have the rover controlled by a companion computer, in the rover side. This companion computer handles all information related to the input and output. It receives information from three main modules, which are Pixhawk, Arduino, and User Interface. These interfacings are handled by Robotic Operating System (ROS), enabling high level abstraction.

Pixhawk and companion computer interfacing enables user to observe states of the rover by existing sensor in Pixhawk as well as enabling companion computer to give command to Pixhawk. The received information covers the rover state, attitude, battery level, etc. While the possible commands to Pixhawk are overriding RC command, arming/disarming, etc.

The Arduino is used to send signal to LEDs as well as reading ultrasonic measurements. Arduino reads commands from companion computer, which is rover state, and send it to LEDs. The ultrasonic measurements are read by Arduino and given to the companion computer. The measurement is useful to determine whether the robot should stop moving or keep going.

Lastly, the User Interface, which is a ROS enabled device, can be connected to the companion computer as long as they are in a common network. The UI is used to give commands to the rover, such as maneuver and adjusting screen, as well as receiving information. More details are discussed in section Graphical User Interface (GUI).

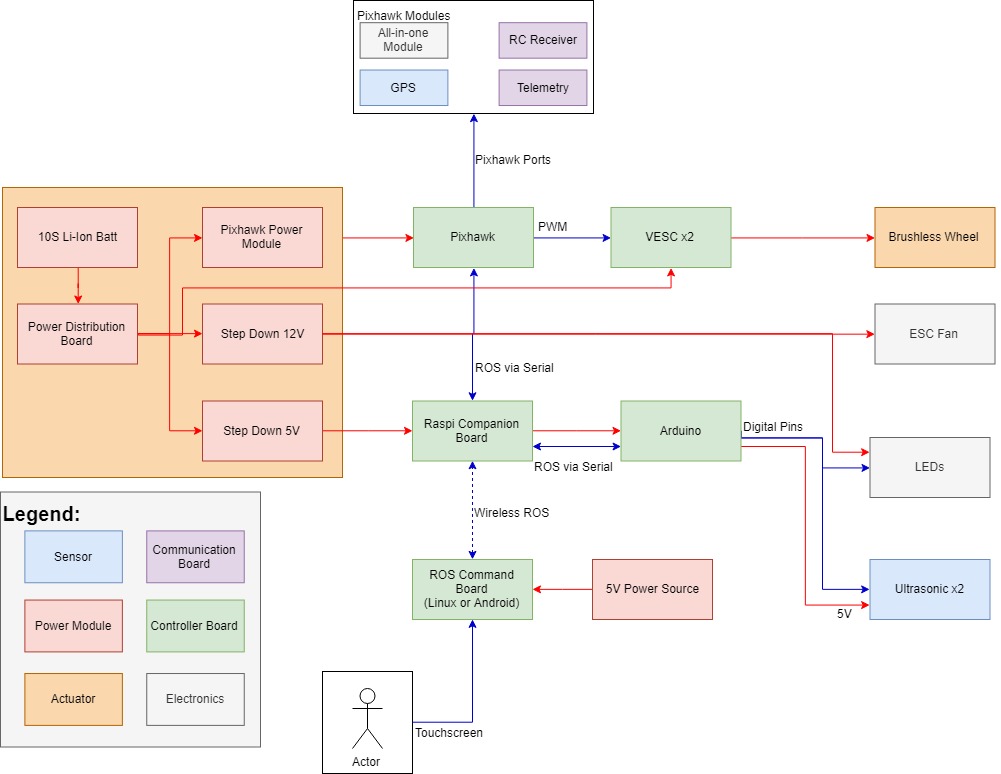


Figure . Wiring Diagram, ROS Integration

# Robotic Operating System (ROS) Framework

ROS is an open-source, meta-operating system which provides services including hardware abstraction, low-level device control, implementation of commonly-used functionality, message-passing between processes, and package management [1]. ROS is not a hard-real-time framework, since it has no guarantee of missing a deadline. It is a language independent framework as one can implement either Python or C++. Understanding ROS itself is challenging at the beginning since the architecture of this software is big and it provides a lot of feature. One should understand the architecture and feature before working on it.

There are three levels of concept in ROS which are file system level, Computation Graph level, and community level. The file system level covers ROS resources that one can encounter on disk, such as packages, meta-packages, package manifest, repositories, message (*msg*) types, and service (*srv*) types. One should specify type of message of each transferred data. The other level, computation graph, is the peer-to-peer network of ROS processes that are processing data together [1]. The basic concepts include nodes, Master, parameter server, messages, services, topics, and bags. All these concepts provide data to the Graph in different ways. The last level, community level, enables separate communities to exchange software and knowledge. This is common in an open source platform.

For the basic operation, one needs to understand specifically the basic computation graph concepts, as shown in Figure 4. Master acts as a name service which stores topics and services registration information for the nodes. Nodes can communicate with the master to report their registration information. A node can publish one or several topics, by writing a message with a specified data type. One can access this published topic by using subscription, which can be done by a node. This subscribing node should specify the data type of the message that it is expecting.

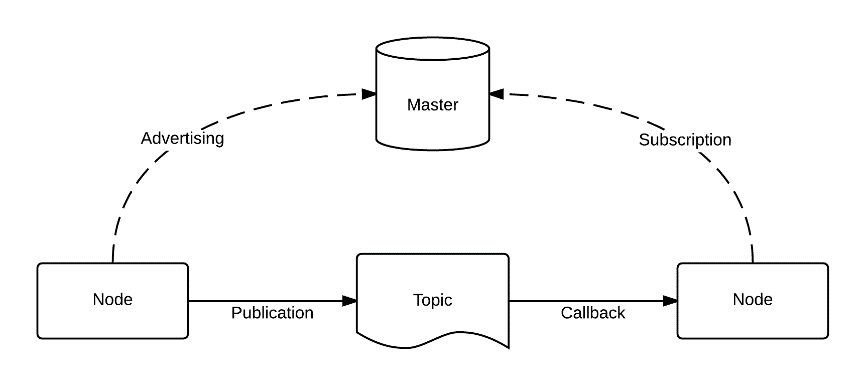


Figure . ROS Basic Computation Graph Concepts [1]

As an example, an inertial measurement unit (IMU) provides a ROS library which starts a node called IMU. This node publishes a message *sensor\_msgs/accel* on the sensor topic. To process the data, such as making a command based on its value, one needs to write a node that subscribes to message *sensor\_msgs/accel* on the specified topic. After the subscription, one can receive the data in the other node.

The benefit of ROS is its abstraction method. One can easily change the hardware and rewrite the same message data type on the same topic. The other nodes do not need any change as it is not affected by the change. The other benefit is one can easily communicate between devices as long as they are connected in one network and have a master running.

# Companion Computer

Companion computer for this research project is used to host ROS in its operating system. Several requirements for the selection of the companion computer are Linux based operating system, supports serial communication on its pins (as an interface to Pixhawk), and its size and weight is relatively small as it will be on board.

For this project, a Raspberry Pi is used, as shown in Figure 11. It has a powerful and energy efficient hardware and smaller in terms of size. The board supports various Linux operating system, for this project, an Ubuntu 16.04 is used as it has a stable ROS version with a lot of libraries.

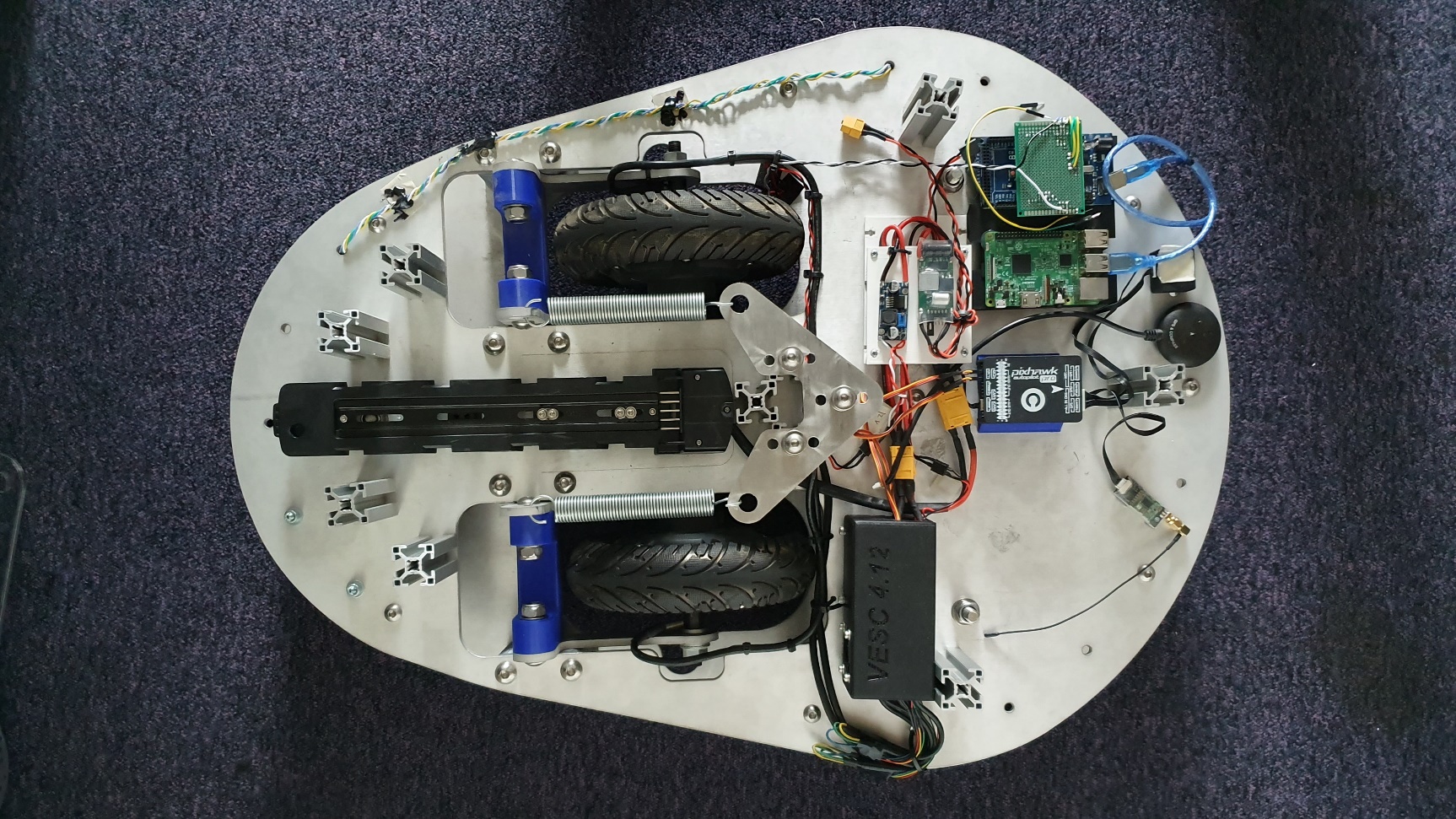


Figure 13. Raspberry Pi on the Rover

## Interfacing: Companion Computer and Pixhawk

Interfacing between Pixhawk and companion computer can be done by utilizing serial communication. Connecting *Telem2* port from Pixhawk to corresponding companion computer pins is a common way in to make the two boards interfaced, as shown in Figure 7.

Port *Telem2* is chosen, instead of *Telem1* and USB, port because *Telem1* is used for data transmission between rover and ground control station. While for the USB, it is used to connect it to ground control station when the rover is on troubleshooting mode and a faster connection is needed.

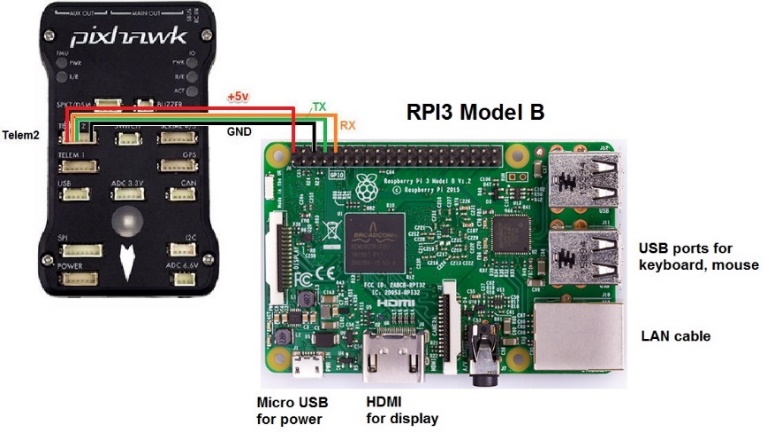


Figure . Pixhawk to Companion Computer Wiring

On the software side, MAVROS is used as the interface between both boards. When all setup is ready, on the terminal, a command ***roslaunch mavros px4.launch***can be executed, as shown in Figure 13. This command creates all the nodes, topics, and messages relate to data coming and from Pixhawk. The list of those data can be found using *rostopic list* from the command line.

Figure 14 shows an example of data sampling. Using *rostopic echo name\_of\_topic*, one can see that the Pixhawk rotates about 90 degrees in x axis. On the monitor, the printed roll value is around 90 degrees, as what we expect. Therefore, one can be sure that the data is correctly obtained. With the ability to connect to Pixhawk, one can obtain any data published by Pixhawk in the ROS topic. On the command side, Pixhawk listen, or subscribe, to several topics and it will react accordingly.

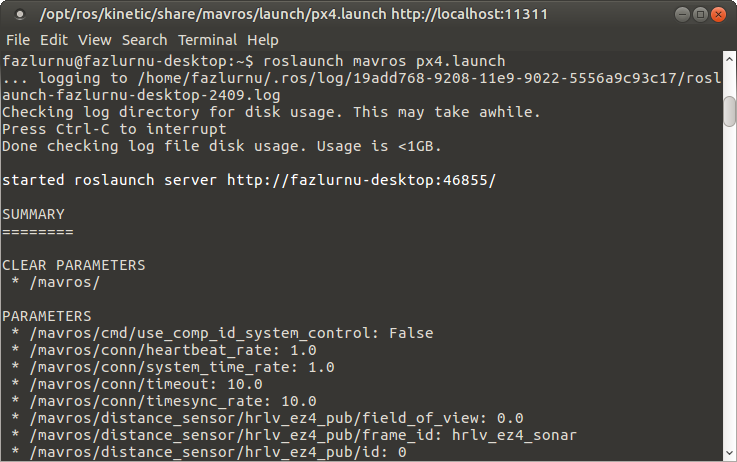


Figure . Launching MAVROS

In the current state, Pixhawk and Raspberry Pi are connected through USB cable onto the USB port. The connection doesn’t use the *Telem2* port due to a problem in interfacing the port and Raspberry Pi serial pins. Nonetheless, the connection is proven to be working. A future update needs to be done.

**Interfacing: Companion Computer and Arduino**

As in Pixhawk, the interfacing of companion computer and Arduino is done by utilizing serial communication. The serial communication can be done by connecting the USB port together, as shown in Figure 13. The other way is to connect Raspberry Pi serial pins to Arduino serial pins.

The communication between Arduino and Raspberry Pi is handled by ROS. Arduino publishes and subscribes to topics in ROS Master. The utilization of Arduino for this rover is to handle the LED color and to measure distances from 2 available ultrasonic sensors. LED color is set based on Pixhawk state, to tell surrounding the safety of Pixhawk. For the sensors, it is used to measure distance and tell the rover to stop moving when a nearby obstacle is detected. Code for Arduino and ROS is presented in the following link, <https://github.com/globibule/arduinoROS>.

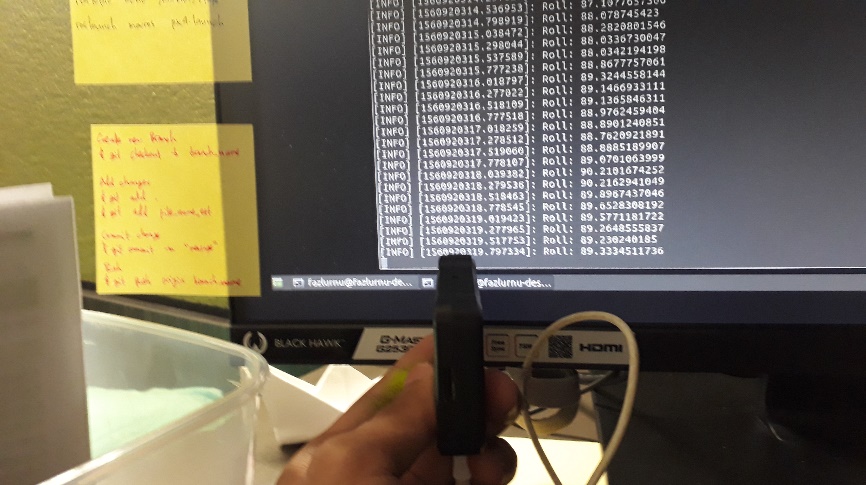


Figure . Data Sampling Example

## Software

The software used in the Raspberry Pi has a goal to control the rover in a semi-autonomous way. Through this code, Raspberry Pi is connected to Arduino and Pixhawk by utilizing ROS framework. The list of subscriber and publisher is shown in Table 2. This ROS code is provided in <https://github.com/globibule/raspiROS>

Table . Publisher and Subscriber

|  |  |
| --- | --- |
| **Type** | **Description** |
| rospy.Subscriber("/ultrasound1", Range, getRange1) | Subscribe to ultrasound1 topic published by Arduino. The value is used for front obstacle avoidance |
| rospy.Subscriber("/ultrasound2", Range, getRange2) | Subscribe to ultrasound2 topic published by Arduino. The value is used for back obstacle avoidance |
| rospy.Subscriber("/positionControl", Twist, getTeleop) | Subscribe to positionControl topic published by GUI. The value is used to control rover and screen movement |
| rospy.Subscriber("/armCommand", Bool, getArmCommand) | Subscribe to armCommand topic, published by GUI. The value is used to activate Pixhawk arming |
| pub = rospy.Publisher('/mavros/rc/override', OverrideRCIn, queue\_size=1) | Publish to /mavros/rc/override topic, subscribed by Pixhawk. The value is used to command Pixhawk movement. |

# Graphical User Interface (GUI)

Graphical user interface (GUI) in this rover is used to communicate with the robot remotely, depicted by Figure 9. The use of ROS in this project has made it easier to communicate between the rover and another ROS enabled device. This ROS enabled device can be in form of a Linux based computer, Android Smartphone (see <http://bit.ly/ROSControl>), or via website (see <http://robotwebtools.org/>).

During this internship, an example of GUI has been made, shown in Figure 15. The code for this GUI can be accessed at <https://github.com/globibule/controllerGUIRaspi>. There are several features provided by this GUI, namely ***control box*, *slider bar*, *arm/disarm* *button*, *armed/disarmed text***. All the information published and subscribed by this GUI is handled by ROS.

The control box, shown by the box in the center, provides the ability to control the rover according to its possible maneuvers (see Figure 4). The red dot represents current condition of the “stick” and its relative coordinate inside the box is transformed to rover command mimicking the remote control. The black dot with line is used to track user’s cursor location, this one is only valid for a non-touch screen device.

The slide bar, shown on the right of the GUI, is used to give command to the screen position (see Screen section). The position of the red dot in the slide bar is related to the screen position in the screen rail. The lowest position of the red dot in the slide bar related to the lowest position of the screen.

For safety purpose, an arm/disarm button is provided. The goal is to activate the rover before user gives any maneuver command. This is used to prevent an unwanted event. The button has red color when a disarm command is given, and it goes green when an arm command is given.

As feedback for the rover state, an armed/disarmed text is provided. This text changes according to the rover state. If the rover is armed, the text will change automatically to armed. If the rover refuses to be armed, due to any safety issue, the text will show disarmed.

|  |  |
| --- | --- |
|  |  |

Figure 17. GUI Prototype

To be connected to the rover, this GUI has to run in the same ROS MASTER by connecting them in the same network (see <http://bit.ly/ROSMultiDevice>). This can be done as long as the GUI has access to the rover companion computer IP address (see <https://github.com/globibule/controllerGUIRaspi>).

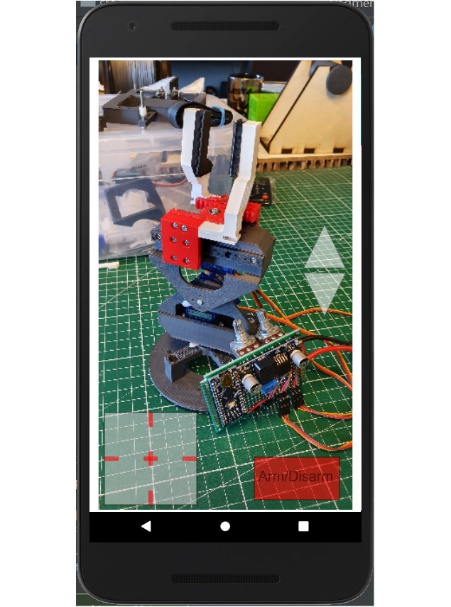


Figure 18. GUI on Android Device

## Further Work

As one may have notice, it is possible to use an Android device to provide the GUI or by providing a web service as long as they have a ROS interface. Due to lack of Android software programming ability, it was not possible to create the software during this internship. Having it on an Android device will provide much convenient for the user as they do not have to have a new device. Another solution is the the web service which will be a versatile, device agnostic solution.

The GUI, in terms of appearance, also need several modifications. First, it should enable the user to see the camera available in the rover. Another note is to have the a “up and down” button instead of a slide bar to control the screen movement. The button can be helpful to avoid the need of having position sensor for the screen, since the user can always decide when to stop and it is more ergonomic to have an “up and down” command. Lastly, an obstacle warning might be useful to inform the user about its surrounding. A possible GUI on an Android device is shown in Figure 18.

**Afterwords**

**Reference**

[1] <http://wiki.ros.org/ROS/Introduction>