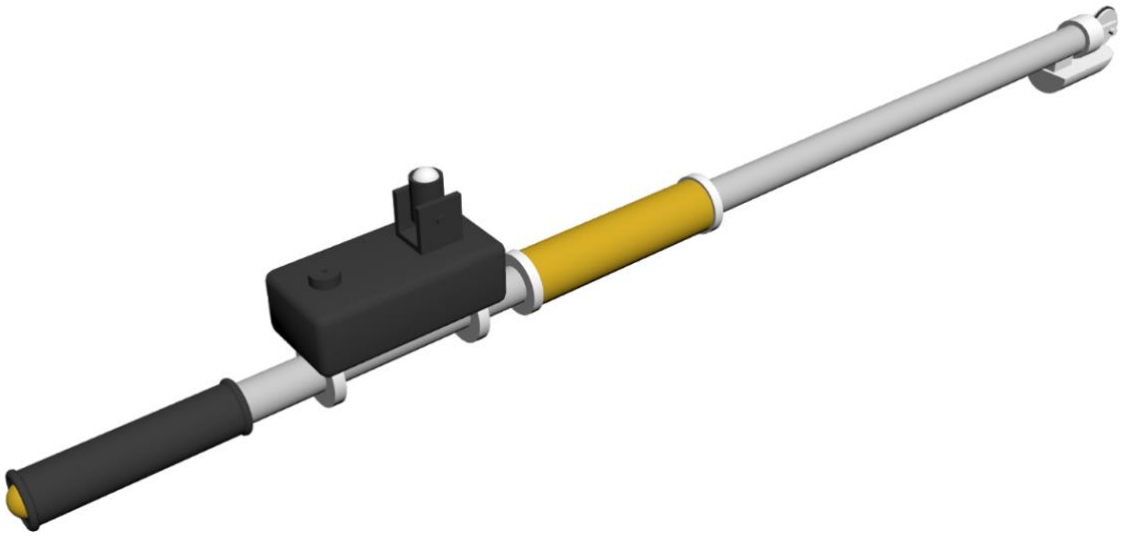
Hochschule für Angewandte Wissenschaften Hamburg
*Hamburg University of Applied Sciences*



# ELECTRONIC GUIDE DOG

an intelligent white cane

Turan Elchuev
turan.elchuev@haw-hamburg.de
10 June 2019

# Table of Contents

**Introduction** ..... **2**

**Device specification** ..... **3**

**Software and Hardware involved** ..... **3**

*Remote Control* ..... *3*

*Cane* ..... *4*

*Pin Connections* ..... *4*

**Development** ..... **5**

*Raspberry Pi setup* ..... *5*

OS installation ..... 5

Configuration ..... 5

*Android App* ..... *6*

*RPi script* ..... *9*

**Testing** ..... **10**

**Delivered configuration** ..... **15**

**Open issues** ..... **15**

**Conclusion** ..... **15**

**References** ..... **16**

# Introduction

The aim of this project is to design a concept of an intelligent assistive device for the visually impaired. As the name suggests, Electronic Guide Dog is meant to replace guide dogs in their function by means of technologies. Due to proven practicality, the classic white cane is selected as a base form factor. With additional mechanics, electronics and intelligence, the device will acquire such capabilities as autonomous navigation, perception of the environment, interaction with the user, etc.

At the initial stage of implementation, decision was made in favor of designing a mockup of the device – a remote controlled cane with all intelligence imitated by a controlling person – for the proof of concept.

The general approach to the design of a mockup is described by the following diagram:

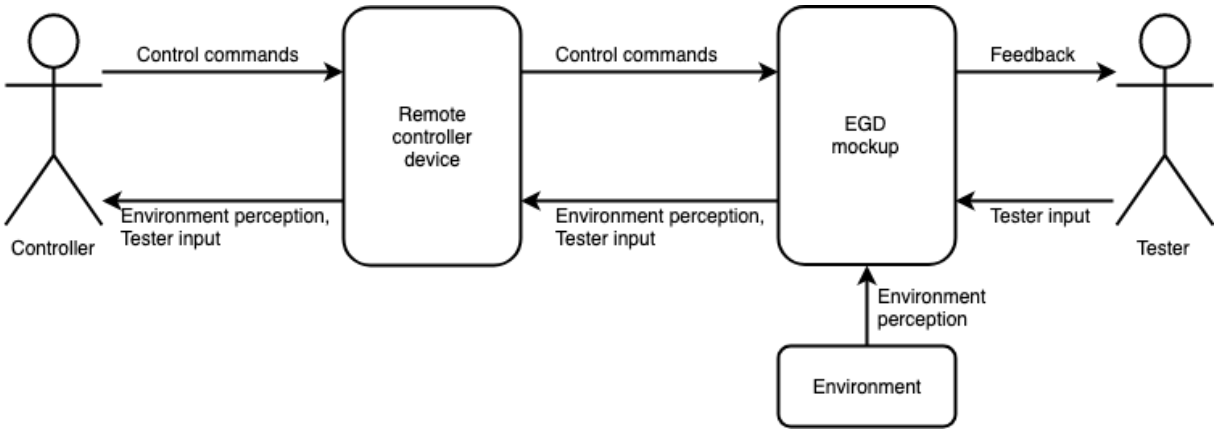



Fig. 1: mockup general diagram

This paper describes implementation of the mockup from Electronics, and particularly, Software perspectives, provides instructions on how to setup and use the developed software as well as introduces a brief summary of the still open issues.

This document does not provide detailed instructions on usage of software such as IDEs or basic Linux commands. For instructions, please refer to corresponding documentation.

## Device specification

As was mentioned in the introduction, the mockup of the Electronic Guide Dog should have capabilities of imitation of autonomous navigation and interaction with the user by means of remote control. Detailed description of the device is provided in the Report by Mr. Aliaksei Khomchanka. Here is a brief summary on the features of the device, omitting the design decisions:

- the cane has a Motor, LEDs and Buzzer, which should be controlled from the RC;
- the cane has a Button, which serves as an input from the testing person. The RC should be notified whenever the button is clicked;
- the cane has a camera which should transmit video to the RC;
- debug data should be published by the cane and visible in the RC.

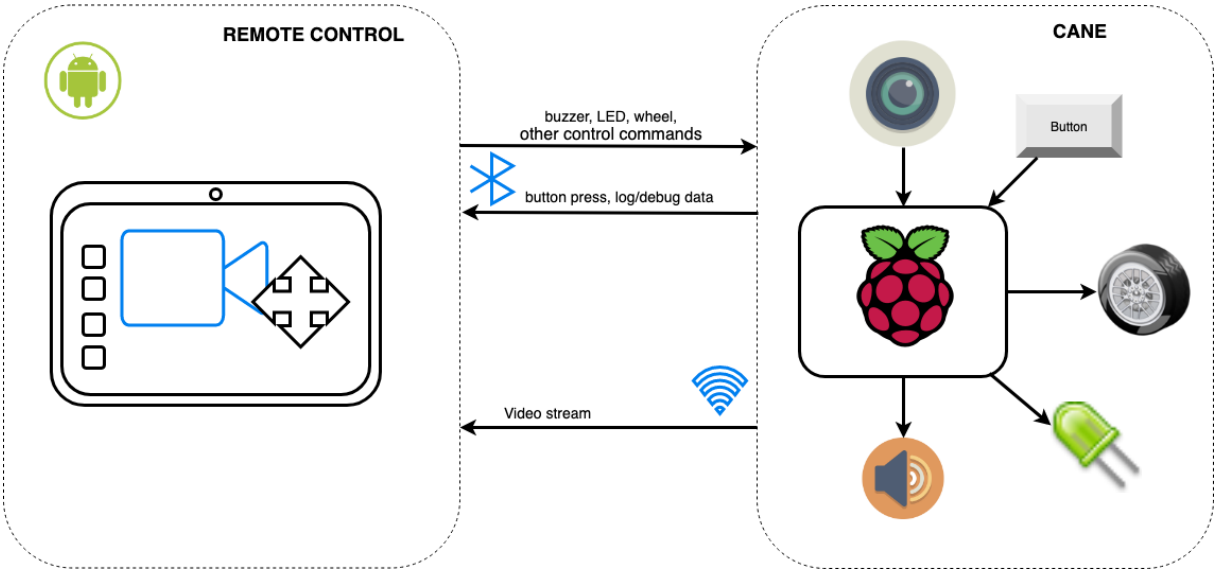



Fig. 2: Device specification

## Software and Hardware involved

### Remote Control

For the design of the RC, preference was given to a pure software implementation since it provides necessary flexibility when further modifications are needed. The Android platform was selected as optimal due to its portability and affordability as well as availability of necessary hardware (e.g. Bluetooth, Wi-Fi modules) on Android devices.

Software:

- Android Studio [1]

Hardware:

- An Android device, e.g. a tablet or a smartphone

## Cane

Raspberry Pi 3 Model B [2] was selected as the core of the system on the cane, because it is powerful and provides capabilities of a fully-fledged PC along with necessary wireless communication modules as well as GPIO pins to interface the periphery. Programming was performed using Python [3] programming language.

Software:

- Raspbian OS [4]
- Idle - IDE for Python programming [5]

Hardware:

- Raspberry Pi 3 Model B
- Bluetooth module HC-05
- 4-Phase 5V Stepper motor + controller board
- Active Buzzer
- LED
- Push-button
- Raspberry Pi Camera module
- Power source, e.g. a power bank

## Pin Connections

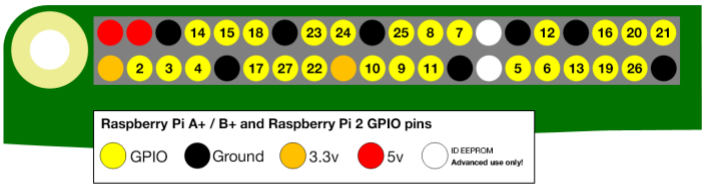



Fig. 3: RPi pinout [6]

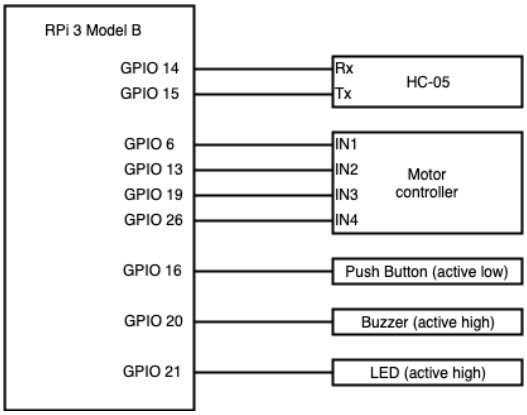



Fig. 4: Pin connections

Figure 4 shows schematic representation of pin connections omitting powering. It is implied that all grounds are connected, and each peripheral device needs a corresponding power input in accordance with its datasheet. In general, all used peripherals can be fed from 5V pins of the RPi, **except for the Motor Controller**, which requires higher current. The RPi camera is connected using its dedicated socket on the RPi board.

# Development

## Raspberry Pi setup

This section describes the steps performed in order to setup the RPi to work as intended. It can also serve as a brief tutorial in case a new setup is required.

### OS installation

In order to install OS on Raspberry Pi, please follow the steps described in the official documentation [7]. Hint: for this project, **Raspbian** was installed using **NOOBS**.

### Configuration

After installation of the OS, RPi has to be configured to enable features required for this project. Comprehensive information on configuration is provided in the official documentation [8].

1. **raspi-config**
   Follow steps described in the section *raspi-config* [9] to change the *Interfacing Options* such that:
   - Camera is ENABLED
   - SSH is ENABLED
   - Serial: login shell over serial – DISABLED, serial port hardware - ENABLE

2. **UART**
   Familiarize with steps described in the section *UART configuration* [10]. Here 2 essential measures have to be ensured:
   - Disabling Linux's use of console UART – this step is the same as configuring Serial in previous step (raspi-config). If it was done before, it can be skipped.
   - Add the **pi3-disable-bt** tree overlay in order to connect UART to GPIO pins. In brief, add a line **dtoverlay=pi3-disable-bt** to the file **/boot/config.txt** as described in the documentation.

3. **Wireless access point**
   Set up the RPi as an access point following instructions in the documentation [11]. The step "Add routing and masquerade" and entire section "Using the Raspberry Pi as an access point to share an internet connection (bridge)" can be omitted as irrelevant for the purpose of this project. At this stage, please note down the static IP assigned to the RPi as well as ssid and passphrase.

4. **Run script on boot**
   In order to run the script on startup of the system, several approaches exist. Within this project, the **rc.local** was chosen. Please refer to documentation [12] for brief instructions on how to use rc.local.

   Hint: pay attention on the *Warning* part of the documentation as the script runs continuously.

   Hint: use *python* instead of *python3.*

## Android App

The App that serves as the Remote Control was developed with the following features:

- Send Buzzer, LED, halt/reboot, video on/off commands using buttons and Motor commands using an X-Y joystick via Bluetooth;
- Implement a Frame format for transmission via Bluetooth (further – Tx frame);
- Display strings received via Bluetooth in the Log;
- Display Video stream received via Wi-Fi (TCP/IP);
- React to events such as button press on the EGD by vibrating and displaying the event in the log;
- Additional features: settings, share log, etc.

Implementation of Wi-Fi for video streaming was the only option, whereas implementation of Bluetooth was the decision made intentionally. The reason is to make the Remote Control easy to communicate with other potential remote devices, e.g. Arduino or any other microcontroller. Implementation of Bluetooth communication on a microcontroller is straight forward (UART).

The core classes:

- Class **MainActivity.java**
  This is the entry point of the app. Main activity pops up when the App is opened and displays its content consisting of a video player in the background and the following 3 fragments in the foreground.

- Class **ControlsFragment.java**
  The left-handed fragment of the app, which encapsulates buttons for toggling LED as well as Buzzer commands. When a button is pressed, corresponding command is sent to the *DataTransmitter* which in turn will send the command via Bluetooth within the next transmitted Tx frame.

- Class **JoystickFragment.java**
  The right-handed fragment. Serves a similar function as *ControlsFragment*, but with a custom X-Y joystick instead of buttons – is used to control the motor. In fact, only the X-axis is used since there is only a single motor, however, transmission of the Y-axis value is also implemented. Transmission of X-Y values is performed in a similar manner as described for the *ControlsFragment*.

- Class **ServiceFragment.java**
  The middle fragment. Serves for several purposes:
  - Video settings (IP, port, resolution, fps)
  - Logger settings (log on/off, autoscroll on/off, clear log, share log)
  - Sends service commands such as reboot/halt and video on/off
  - Controls Bluetooth connection (connect, disconnect)
  - Handles strings received via Bluetooth (adds them to the Log)

  Sending the commands is performed in the same manner as in previous 2 classes.

- Class **BluetoothCommunicationHandler.java**
  This class handles Bluetooth communication:
  - takes a Bluetooth device object and performs connection to it.
  - provides methods for sending bytes, receiving frames (IRxFrame)
  - provides callbacks for connect/disconnect/failed connection and frame received events.

- Class **BluetoothHelper.java**
  This is a helper class to deal with the *BluetoothCommunicationHandler*. It provides a useful method *pickBTDeviceAndConnect()*, which pops up a dialog with a list of paired Bluetooth devices such that the user can pick a device to connect to.

- Class **DataTransmitter.java**
  This class is responsible for transmission of Tx frames. It encapsulates a Thread that runs in background and with a specific frequency (e.g. every 50 ms) packs the commands into a frame and transmits using the *BluetoothCommunicationHandler*.

  The format of the **Tx frame** is as follows:

  byte 1: S - start of sequence (0x33)
  byte 2: joystick_X - joystick X value (value range 0…127)
  byte 3: joystick_Y - joystick Y value (value range 0…127)
  byte 4: command - some command, e.g. LED, RPi reboot… (value range 0…127)
  byte 5: E - end of sequence (0x77)

- Class **VideoStreamDecoder.java**
  This class filters TCP/IP input stream, detects video frames, decodes them and displays in the Video Player.

- Class **TcpIpReader.java**
  This class provides TCP/IP connection to the given IP:port and corresponding input stream.

- Interface **IRxFrame.java**
  This is an interface that is used for compatibility of the *BluetoothCommunicationHandler* with other classes such as *FrameLogAdapter* as well as various possible Frame formats that are supposed to be received via Bluetooth. The concept is as follows: the *BluetoothCommunicationHandler* receives frames, which are the implementations of *IRxFrame*. The *FrameLogAdapter* also accepts frames which are the implementations of *IRxFrame*. For each particular needed frame format, a corresponding implementation of the *IRxFrame* interface should be created. Within this project, only a single possible implementation exists, which is described next. However, other implementations can further be added.

- Class **RxFrameString.java**
  The implementation of the *IRxFrame* interface for receiving plain Strings via bluetooth. The frame format is specified as follows: N (any number) bytes of data

followed by the stop-sequence ['\r', '\n']. The bytes preceding the stop sequence will be treated as a char[] and converted to a String once the stop sequence is detected.

- Class **FrameLogAdapter.java**
  Provides basic functionality such as adding *IRxFrame* frames into the log, clearing log, setting/resetting autoscroll mode and getting the entire log in a single string for the sharing purpose (e.g. sending via Email).

- Class **Settings.java**
  This class handles saving and retrieving settings such as video IP/port, resolution, fps.

Following diagram shows simplified structure of the app including data flow between the core classes:

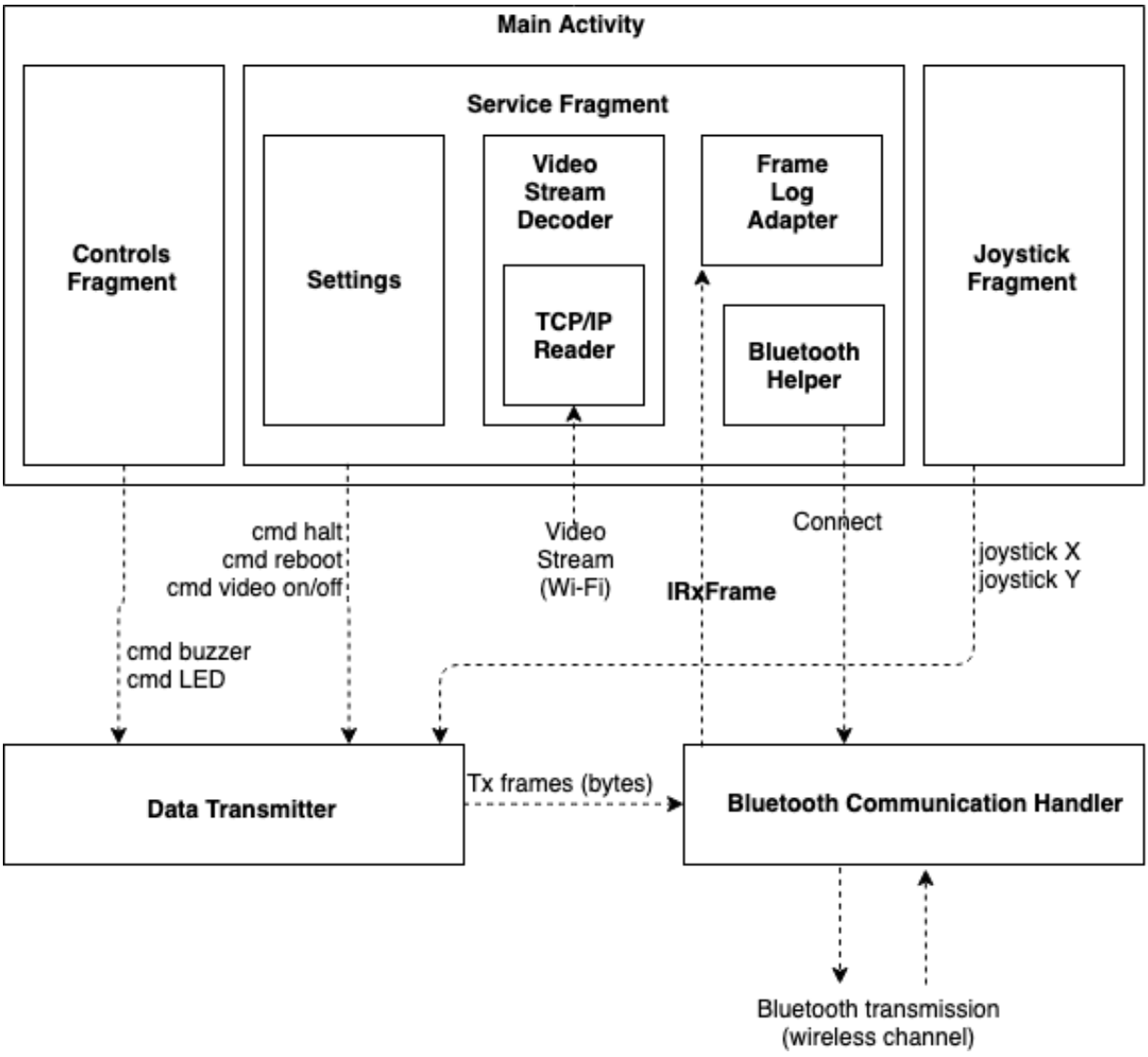



Fig. 5: Structure of the App and data flow diagram

For the details on implementation please refer to the provided source code which has necessary comments. The most recent .apk installation file was also provided.

## RPi script

A python script was developed that serves as the interface between the Remote Control and the periphery of the EGD. This script is the one that has to be executed on system startup, which is mentioned in the *step 4* of the *Raspberry Pi Configuration* subsection (p. 5).

Specification of the script is defined as follows:

- Receive commands via Bluetooth (UART) in the **Tx frames** mentioned before. After receiving the command, call corresponding function, e.g. toggled LED, rotate the motor, start/stop video streaming;

- Detect push button press and send corresponding message to the Remote Control via Bluetooth;

- Send debug messages to Remote Control whenever new command is received, button is pressed, etc., e.g. “Program started”, “Command: led”…;

- Messages sent to the Remote Control should be preceded by the prefix **“RPi: ”** in order to distinguish them from the other Log data at the Remote Control side;

- Messages sent to the Remote Control have to end with suffix **“\r\n”** in order to be compliant with the **RxFrameString** mentioned before;

- Messages that should bring the attention of the controlling person should have additional prefix **“cmd:”**, e.g. when the push button is pressed by the testing person. The reason is that messages having that additional prefix will cause the Remote Control to vibrate.

  Examples:
  “**RPi:** Command: led**\r\n**” – standard message (will appear in the log);
  “**RPi: cmd:**btn**\r\n**” – command message (will appear in the log and cause vibration).

- Transmit video using built in tool **raspivid** [13] through netcat [14].
  This approach is selected as the optimal one since it provides the smallest latency and the opportunity of integration with the OpenCV. This conclusion is based on comparisons of various methods of streaming video from Raspberry Pi [15]. Compatibility with the OpenCV may come in handy at later stages when computer vision capabilities will be integrated into this mockup.

For the source code and more details on implementation of the script please refer to the provided **script.py** file. It includes necessary comments.

## Testing

1. Since the stepper motor does not have any encoder, the position of the wheel should be manually adjusted to the neutral (straight). This is needed because when the system starts, the current position of the stepper is treated as the reference (neutral) position and each rotation is performed with respect to that position.
   Adjust the wheel by hand.

2. Switch the RPi and all the peripherals ON.

3. Start the Android App:

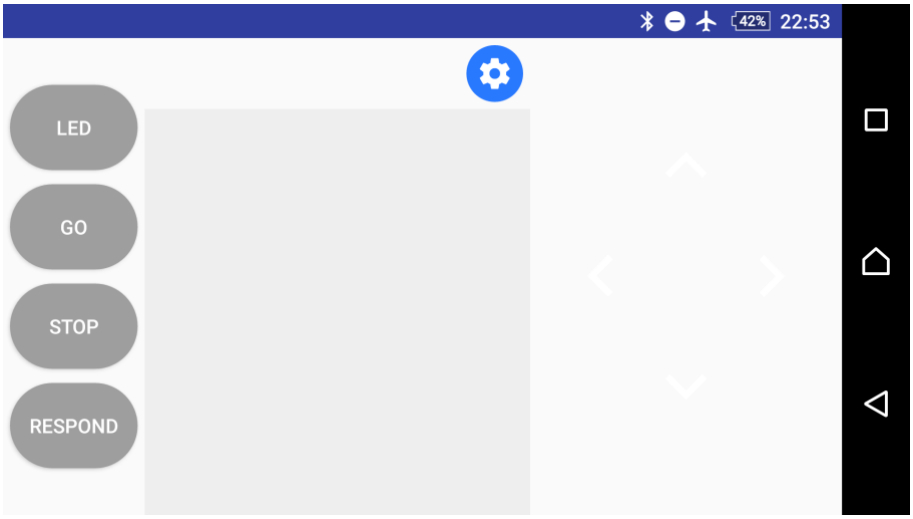



Fig. 6: Start screen of the App

The area at the left-hand side (with gray buttons) is the **Controls Fragment** mentioned before; the right-hand-sided white rectangle is the **Joystick Fragment**; in between is the **Service Fragment** with the blue button “Configuration” and the **Log** (gray area).

4. Press the blue button “configuration”. The settings and commands dialog will pop up:

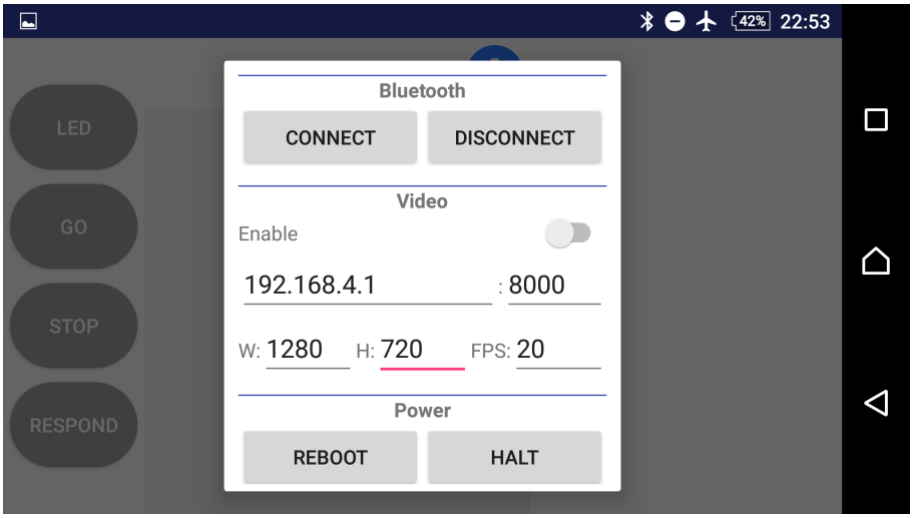



Fig. 7: Settings and commands dialog

5. Press “CONNECT”. A dialog with the list of paired Bluetooth devices will pop up:

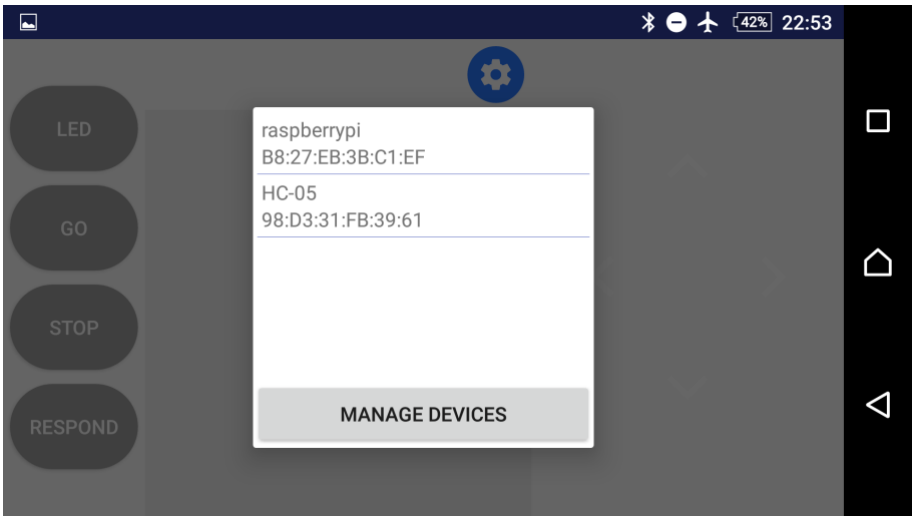



Fig. 8: Paired Bluetooth devices dialog

If you don’t see the Bluetooth device of the RPi (in this case HC-05, since the built-in Bluetooth module is not used by the script), you have to add to the list of paired devices. In order to do so go to Bluetooth settings of the Android device, search for the Bluetooth adapter and pair with it, afterwards, go back to step 4. A shortcut to the Bluetooth settings of the Android device is through pressing the button “MANAGE DEVICES”.

6. Find the target Bluetooth device and tap on it. The connection should be attempted.

   If connected and if RPi is properly configured, switched on and the python script is running, you should see at least the first message from the RPi:

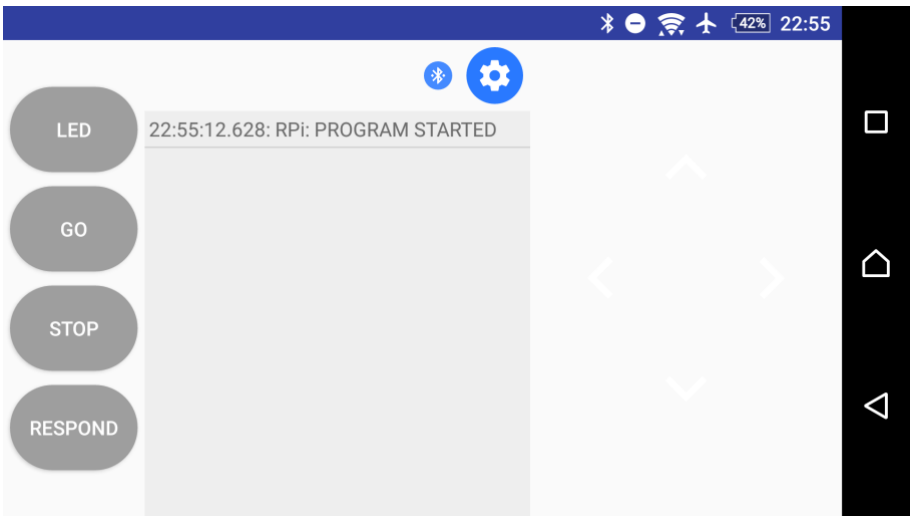



Fig. 9: First message sent by the Cane when python script starts

7. Press command buttons, e.g. LED, GO… The Cane should respond correspondingly, e.g. toggle LED, play the “GO” command with the buzzer, etc. Press the push button on the Cane – the Android device should vibrate. After each action, the Cane should send corresponding messages:

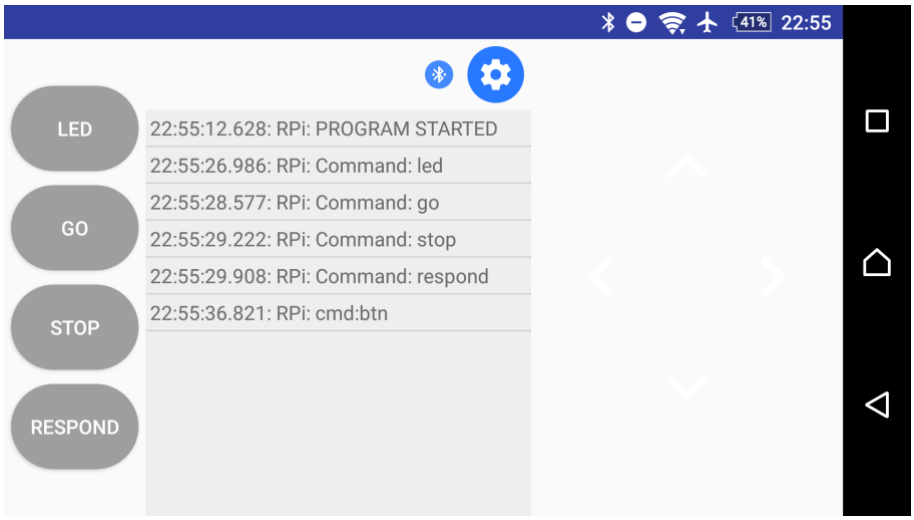



Fig. 10: Messages from the Cane when commands are received

8. Touch the Joystick area and move the finger, then lift the finger up. You should see the joystick tip popping up, following the finger up to a certain limit and vanishing when the finger is removed. At the same time, you should spectate corresponding behavior of the Cane:
    - Moving the joystick along the X axis (horizontally) will rotate the wheel proportional to the distance of the joystick tip from the point where the finger first touched it
    - Releasing the joystick will bring the wheel to the neutral position

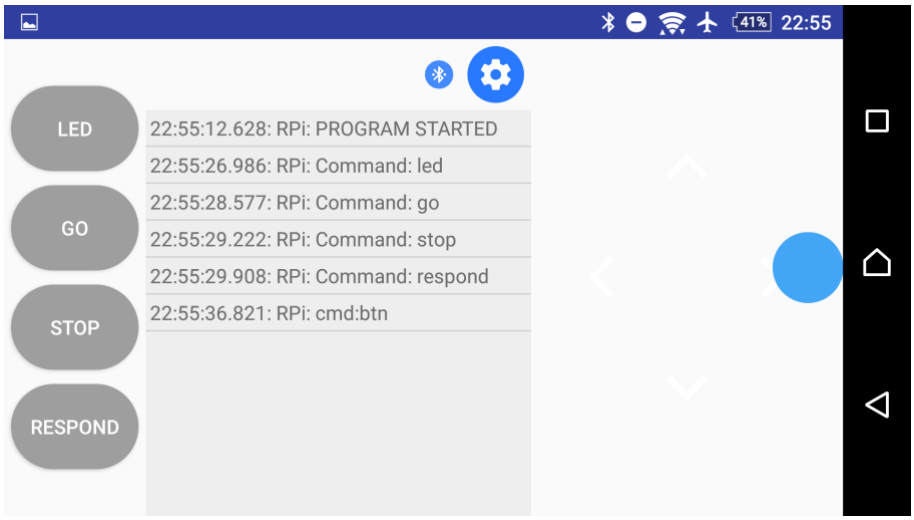



Fig. 11: Joystick tip pops up

9. Open the settings and commands dialog again (as in the step 4):

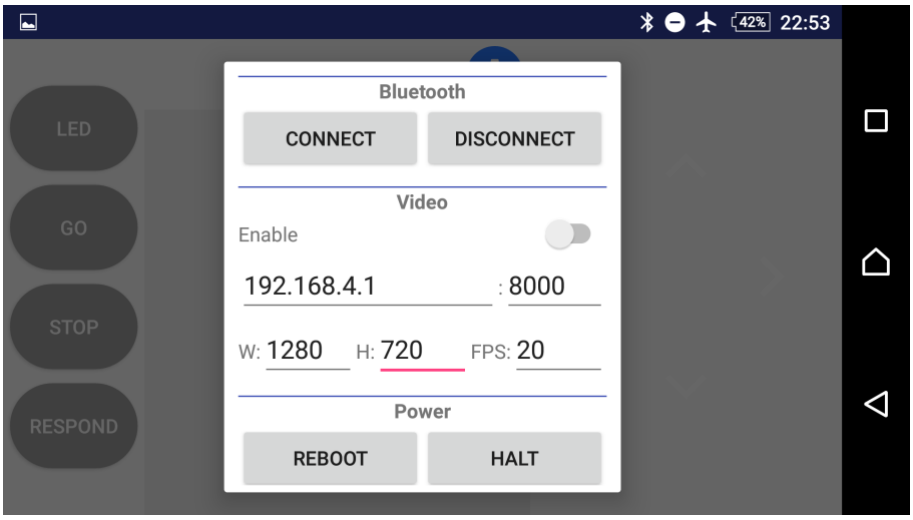



Fig. 12: Settings and commands dialog

At this stage:

- Make sure the IP is correct: it has to match the static IP of the RPi set during the access point configuration.
- Make sure the port number is 8000.
- Make sure the Android device is connected to the access point of the RPi (Android's Wi-Fi is connected to the RPi).

Enable video transmission (turn the "Enable" switch under the "Video" section ON). After some time, you should see video in the background:

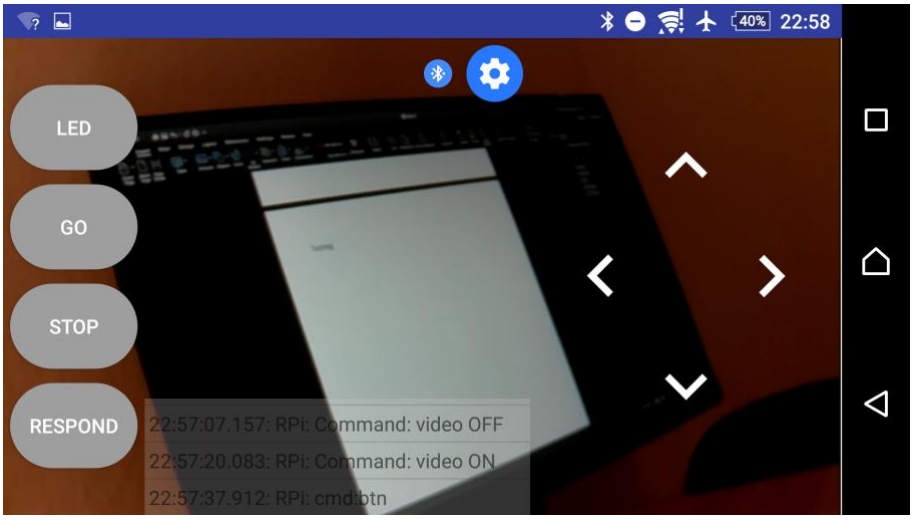



Fig. 13: Video streaming is ON

Please also note that the Log area is shrinked to release more space for the video. In a similar manner, Video can be switched OFF followed by extension of the Log to a normal size. Besides, each command should be accompanied by corresponding message from the RPi which will appear in the Log. If video fails to start, try switching on/off few times. If that does not help, probably, this feature does not work for this particular Android device (this problem is described further, in the Open Issues sec.).

10. Open the settings and commands dialog again (as in the step 4) and scroll down to see other commands:

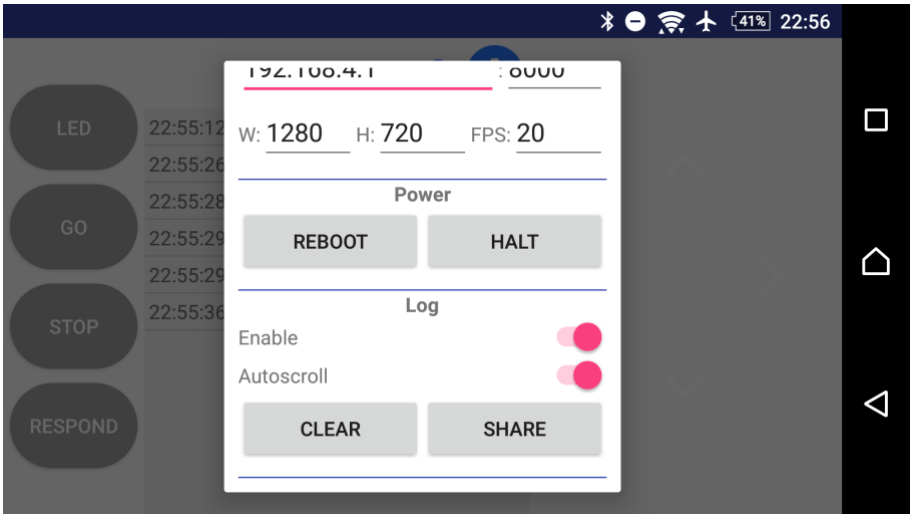



Fig. 14: other commands

Try commands:

- SHARE – sharing the content of the Log as plain text e.g. using Email.
- CLEAR – will clear the Log
- Enable – hides/shows the Log. Disabling the Log will not prevent it from tracking the messages, thus, once enabled, all messages received so far will be displayed.
- Autoscroll – enables/disables automatic scrolling of the Log to the last received message whenever new message is received.
- REBOOT – will reboot the RPi. This will not cause the Bluetooth to disconnect, thus, after rebooting, the script will start again and send the first message (as in step 6), which will be displayed in the Log. This means, rebooting does not require repeated connection via Bluetooth.
- **HALT** – same as REBOOT, but RPi will shut down without starting again. This should be used to shut the RPi properly down.

## Delivered configuration

Wi-Fi access point on RPi:

| | |
|---|---|
| ssid: | **RPi** |
| password: | **raspberry** |

SSH:

| | |
|---|---|
| user: | **pi** |
| IP: | **192.168.4.1** |
| Password: | **egd** |

## Open issues

Some issues still remain unresolved and need more investigation:

1. Using RPi built-in Bluetooth module instead of the external module for serial communication.

2. Higher lagging of video streaming when RPi is set up as a wireless access point compared to that when the RPi and the Android device are connected through an external hotspot (router).

3. On some Android devices video streaming does not work. Suggested starting point for debugging: class VideoStreamDecoder.java, line 157:

   ```
   int len = reader.read(buffer);
   ```

   **len** is -1, which means for some reason there are no bytes available in the input stream of the TCP/IP reader although they are being transmitted by the RPi.
   This problem occurs on some devices and does not on the others.

4. Stepper motor does not have an absolute encoder, thus has to be adjusted manually to neutral position before powering up.

5. Issue irrelevant for the mockup, but relevant for the final product: computational power of the RPi.

## Conclusion

During this project, extensive work was performed in cooperation with the other team members in order to develop a mockup of the Electronic Guide Dog – a device that will help to analyze the concept of the EGD from the usability perspective. The device has closely approached the state of the minimum viable product for testing with the target users involved. The current state of the device is a good starting point for the future teams to polish it through improvement of its hardware and appearance.

# References


[1] https://developer.android.com/studio/
[2] https://www.raspberrypi.org/products/raspberry-pi-3-model-b/
[3] https://www.python.org/
[4] https://www.raspberrypi.org/downloads/raspbian/
[5] https://en.wikipedia.org/wiki/IDLE
[6] https://www.raspberrypi.org/documentation/usage/gpio/
[7] https://www.raspberrypi.org/documentation/installation/
[8] https://www.raspberrypi.org/documentation/configuration/
[9] https://www.raspberrypi.org/documentation/configuration/raspi-config.md
[10] https://www.raspberrypi.org/documentation/configuration/uart.md
[11] https://www.raspberrypi.org/documentation/configuration/wireless/access-point.md
[12] https://www.raspberrypi.org/documentation/linux/usage/rc-local.md
[13] https://www.raspberrypi.org/documentation/usage/camera/raspicam/raspivid.md
[14] https://en.wikipedia.org/wiki/Netcat
[15] https://www.youtube.com/watch?v=sYGdge3T30o