Horo

Horst Dumcke

Feb 05, 2023

CONTENTS:

CHAPTER

ONE

OVERVIEW

Horo is a project to build a low cost wheeled robotics platform using second hand hoverboards. The hoverboard robot (Horo) is an educational tool to learn about robotics and the robotic operating system ROS.

TODO: insert picture

Building the platform by using scrap material should cost less than 100 EUR/USD, adding sensors will incur additional costs but these sensors can also be used in other projects.

1.1 High Level Architecture

The high level architecture is shown in :numref:hla.

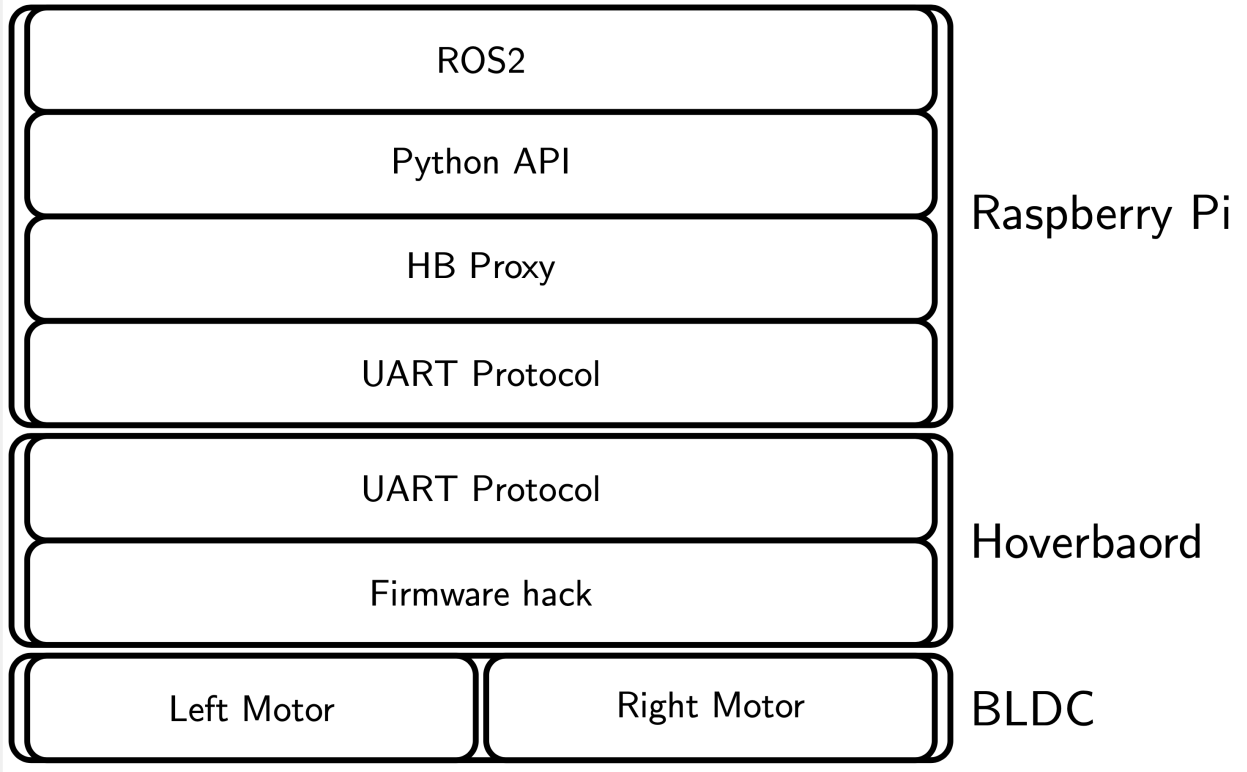

Fig. 1: High Level Architecture

We will modify the firmware on the hoverboard to be able to control the motors from software running on a Raspberry Pi. The communication between the hoverbaord and the Raspberry Pi will be done with a protocol running over UART. We will run a proxy on the Raspberry Pi that will manage the protocol on the UART interface but also allow a Python module to interact with the protocol and thus being capable to controlling the motors and reading the motor encoders.

We will use this Python API in a ROS2 node to abstract the hardware.

1.2 Key Building Blocks

The key building blocks for Horo are as follows:

Hardware: The hardware consists of a hoverboard that is physically modified (some pars are cut with a saw), a wooden board or any other material to build a platform, a Raspberry Pi 4B as the main controller and added sensors (see below)

Hoverboard Firmware We will flash the boards that are shipped with the hoverboard hardware to run customized firmware

Hoverboard Interface We will run code on the Raspberry Pi that will interface with the hoverboard hardware over UART and provide interfaces to ROS by sending and receiving messages to and from ROS topics

ROS We will use ROS2 to implement the robots functionality

1.3 Sensors

IMU We use an inexpensive MPU6050 board

Camera The camera will be used for computer vision. Having a neural network processor integrated into the camera will free resources on the main computer. We will test with OAK-D from Luxonis and other non-ai web cams.

Lidar A light detection and ranging sensor will be used to scan the environment of the robot. We will test with a relative inexpensive LD06

Joystick controller To move the robot by hand we will use a PS4 joystick

Wheel encoders We will use the hal sensor that are part of the BLDC motors of the hoverboard to sensor the wheel movement

1.4 Applications

Navigation/SLAM

Line following

Teleop with joystick or keybaord

CHAPTER

TWO

HOVERBOARD

We will leverage the work done and documented in the following projects:

https://github.com/lucysrausch/hoverboard-firmware-hack.git

https://github.com/flo199213/Hoverboard-Firmware-Hack-Gen2.git

We are using a Gen2 hoverboard and assembled firmware from different forks of the original repo.

2.1 Building the firmware

The original (hacked) firmware requires Keil to compile the code. As Keil is only available on Windows we have build a procedure that allows to build the SW on many OS: Windows, Linux and Mac OS. As the cross compiler runs on X86 we currently do not support ARM based PCs.

The build procedure consists of starting a Ubuntu VM, downloading the compiler and the source code and then building the firmware for the master and the slave board.

2.1.1 Install Multipass

Follow the guide at https://multipass.run to install Multipass for your operating system

2.1.2 Install Multipass Orchestrator

Follow the README in https://github.com/hdumcke/multipass-orchestrator

2.1.3 Deploy the Build VM

Clone https://github.com/hdumcke/multipass-orchestrator-configurations and use mpo-deploy to deploy the configuration in multipass-orchestrator-configurations/armtoolchain/config.yaml

This will start a VM, install the build system, clones the repository with the source code and compiles the firmware for the master and the slave board. You will find the firmware in the directories ~/master and ~/slave in the VM. Use SCP to transfer the firmware to your host PB

2.2 Flashing the boards

You will find the schematics of the board here: https://github.com/hdumcke/linorobot2_hoverboard/blob/main/Hoverboard-Firmware-Hack-Gen2/Schematics/HoverBoard_CoolAndFun.pdf

It is convenient to solder pins into the holes for the STLink connector. Do only connect GDN, SWDIO and SWCLK, do not connect power otherwise you risk to break the board.

st-flash –reset write firmware.bin 0x8000000

Make sure you use the correct firmware for the board you are flashing (master or slave).

2.3 Future directions

There are different ways to control the phases of a BLDC motor: trapezoidal, sinusoidal or FOC (field oriented control)

There are open source projects that implement BLDC control algorithm but we have not found any project that applies to Gen2 boards that we are using.

We should investigate and write our own control algorithm.

CHAPTER

THREE

RASPBERRY

This project requires multiple components being installed on the Raspberry:

Operating System As we will use ROS2 Humble we requires Ubuntu 22.02 We will opt for the server version but desktop version should also work.

HB Proxy We requires a proxy that manages the communication with the hoverbaord. This proxy is written in C and is controlled by systemd

Python Module The hoverbaord hardware is abstracted by a Python API

ROS ROS2 Humble and ROS2 code that is part of this project needs to be installed

3.1 Installation

To simplify the installation of the Raspberry Pi we provide a cloud-init based installation procedure.

Prepare a SD card with Ubuntu 22.04

https://cdimage.ubuntu.com/releases/20.04/release/

ubuntu-22.04.1-preinstalled-server-arm64+raspi.img.xz is recommended.

Clone https://github.com/hdumcke/linorobot2_hoverboard on the PC where you have created your SD card. Make sure the SD card is mounted. Run

prepare_sd.py

And answer the questions. When done, eject your SD card, stick it into the Raspberry Pi and boot. At the end the Raspberry Pi will reboot and you should have a complete installation.

During installation the output of the setup script will be written to ~/.setup_err.log and ~/.setup_out.log These file can be used to monitor the installation process

CHAPTER

FOUR

CUSTOMIZATION

4.1 URDF

TODO

4.2 RSO2 control parameters

TODO

4.3 Pair Joystick

Pair bluetooth controller:

```
bluetoothctl
scan on
pair 00:26:5C:06:ED:B3
connect 00:26:5C:06:ED:B3
trust 00:26:5C:06:ED:B3
exit
```

CHAPTER

FIVE

OPEN ISSUES

Current main open issue for me is flashing the firmware. It sometimes works but mostly it is not working. I have ordered a new STlink and will see if this will solve the issue

Patrick has changed the firmware, now with speed for each motor as input and wheel encoder data as output. The proxy needs to be adapted as well as the python API.

Linorobot2_hoverboard/ros2/linorobot2_hoverboard_control/linorobot2_hoverboard_control/hoverboard_interface.py must be updated to use wheel encoder information to generate ROS messages for TF and odom

I am currently looking at ROS control if that can be used.