Larbits CryptocurrencyMinerCar

Introduction

This gitlab accompanies the second version of the LarbitsSisters robot. This includes a round design with TOF sensors around the chassis. The robot is able to drive and charge fully autonomously. Both Arduino code for the OpenCR board, Python code on the RPi and the Fusion360 model of the chassis are saved in this repository.

Installation

For each robot, start with a clean install of Raspbian. For our implementation we used Buster (with Desktop - release April 4th, 2022 - download here). Follow the next steps:

First, install required depencies for the TOF sensors:

```
sudo pip install smbus2
sudo pip install vl53l1x
```

Next, we setup some basic RPi things: change the password, enable ssh, I2C and OpenVNC. Change the name of the Pi to the correct robot, and set the timezone. Furthermore set the resolution fixed (otherwise, if you boot the RPi without screen attached and then use OpenVNC to login the resolution is only 640x480). All of this is done through raspi-config. Since this is basic RPi configuration, I won't go into detail. None of this is actually required for the robot to function,

except for the I2C bus (which should be enabled).

```
sudo raspi-config
```

A reboot might be necessary after the above changes. Next, clone the repository. For the startup-script on boot, it is required to clone it on the Desktop. It is possible to change the location, for this only the systemctl startup script should be changed (start_robot.service - not tested):

```
cd ~/Desktop
git clone https://gitlab.com/EAVISE/larbits_v2.git
```

Furthermore we need to install/change some config files. First we increase the speed of the I2C bus (default = 100000Hz).

```
sudo nano /boot/config.txt
```

Find the line containing dtparam=i2c_arm=on. Add i2c_arm_baudrate=400000 to this line, right after the dtparam. The line should contain:

```
dtparam=i2c_arm=on,i2c_arm_baudrate=400000
```

Reboot the RPi.

Next, we install a config file that ensure that the serial communication with the OpenCR board is always mapped to /dev/OpenCR. This is needed because sometimes the OpenCR board crashes, restarts and the default device mapping might be different (e.g. /dev/ACM0, /dev/ACM1,...).

```
cd ~/Desktop/larbits_v2
sudo mv 98-my-rules.dev /etc/udev/rules.d/
sudo chown root /etc/udev/rules.d/98-my-rules.dev
sudo chgrp root /etc/udev/rules.d/98-my-rules.dev
```

Disconnect and plugin the USB to the OpenCR board. It should now show up as /dev/OpenCR (when doing ls /dev).

Let's test the software now. Everything should work:

```
cd ~/Desktop/larbits_v2
python3 autonomous_robot.py
```

You can also use the restart-on-error script, which catches OSErrors and reboots the script. This is

required because sometimes the I2C TOF sensors tend to crash. A restart fixed this issue:

```
cd ~/Desktop/larbits_v2
./restart-on-error.sh python3 autonomous_robot.py
```

If everything work, you can continue. Kill with CTRL+C. Next we ensure that the robot automatically starts at boot. For this, create a systemctl entry (copy it from the gitlab repository):

```
cd ~/Desktop/larbits_v2
sudo cp start_robot.service /lib/systemd/system/
sudo chmod 644 /lib/systemd/system/start_robot.service
sudo systemctl daemon-reload
sudo systemctl enable start_robot.service
```

The robot should now work after boot. You can create an alias to easily access the log of the robot. Open ~/.bashrc and add the following line to the end (make sure the path to the log file is correct, if you installed the repository to a different location than the Desktop):

```
nano ~/.bashrc
alias status_robot=cat ~/Desktop/larbits_v2/robot.log
```

Usage

Since the robot is automatically started with a script, you have to shut this down at boot if required. You can also restart and get the status with one of the following commands:

```
sudo systemctl stop start_robot
sudo systemctl start start_robot
sudo systemctl status start_robot
```

You can read the automatically generated logs of the robot using:

```
status_robot
```

Hardware

The robot consists of the following hardware:

- RPi3 for I2C sensor readout, vector computation, logging and communication of the required action through serial communication with the OpenCR board

- OpenCR board: drives the motors through RS484 communication. Readout the magnetic sensor through interrupt routine. Voltage conversion from battery voltage to stable 12V and 5V for RPi
- 7 TOF sensors, type VL53L1X. Readout through I2C
- Battery currently is a NiMH 12V (10-cell) 2000mAh type
- Motors: Dynamixel XC430-W240 - see datasheet here
- Magnetic hall sensor based on Iduino 1485303
- Two 12V fans to actively cool the motors. Powered at 8V through step-down convertor (to lower noise and power consumption)
- Additional power on/off switch and two charging points on the front

How the robot works

IMPORTANT NOTICE: on power up the robot is in drive mode. I.e. do NOT power up the robot while it is in front of (due to the magnetic line), or on the charging station. Switch on power and place the robot somewhere on the track where it is able to drive.

Computation on RPi

The robot uses the 7 TOF sensors to detect the optimal driving direction. The sensors are configured in low-range mode, to scan every 20ms. The sensor values are recalculated into a force: if there is enough space this force pulls the robot towards the empty space. Below a specific distance the distance is used to calculate a force which repels the robot (a bit spring-like). Based on the fixed angles of the sensor, a final rotation and force is calculated during every scan. By default (can be changed with the parameters) the robot scans every 500ms. First, the robot performs the required rotation, then continues to drive with a speed which depends on the calculated force. If there is a large open space, the robot thus moves fast. If there is only few open space left, the robot starts to move slower. The contribution of each sensor is weighted: sensors on the side have a slightly lower weight (0.5). The front sensor has maximal weight (1.0). Sensors in between vary (i.e. 0.75 - 0.85). The required action is communicated serially to the OpenCR board (which actually drives the motors). During every scan, the RPi requests the status from the OpenCR board. This includes the battery voltage, temperature of both motors and the state of the magnetic sensor (if a magnetic line was detected).

A magnetic line is hidden under the track, about 5-8 cm in front of the charging station. If the magnetic line is detected, the robot checks the battery voltage. If this is above a specific value (default 12.6V) the robot rotates 180 degrees and start to drive away again. If this value is lower than the threshold, the robot tries to dock as follows: he will drive slowly forward (no correction based on sensors) for 5 seconds. Normally, the robot is centered enough such that it drives itself onto the charging station. Next, the robot wait for 15 seconds (the battery charger takes some

time to increase the voltage). Next, the voltage is sampled 25 times with 200ms pauses (5 seconds). The maximum voltage is determined. This is required, because the charger block-charges (only increases voltage a specific intervals). If a voltage which is significantly higher than the previous battery voltage is detected (400mV by default), the robot goes into charging mode. Otherwise docking is failed, the robot drives backwards, rotates 180 degrees and continues its way). In charging mode, the robot wait for one of two things: either the battery voltage above a specific threshold (14.7V by default) or a maximum charging time (25 min by default). The latter is required since when the battery gets older, the maximum voltage might not be reached. When either of these to conditions are met, the robot peforms the same manoever as when docking was failed (continue driving).

The robot logs its state (and battery voltage + motor temperatures) about every 10 seconds. See above to readout state. Logging is implemented with the Python logging library. Log file is robot.log. The maximum filesize is 200kB. When the log file's size is larger, the log file is stored as *.log.1. A total of 5 such log files are stored (after all 5 log files are written, the oldest log file gets deleted).

OpenCR board

The OpenCR board runs Arduino-code to drive the motors. If waits for serial commands from the RPi. The usable commands are the following:

- drive_cnt,L,R ==> drives the motors with a speed of L for left and R for right motor. Speed is limited to 306. The robot keeps on drive for a maximum of 800ms (default). So, every 800ms a new command must be send from the RPi. This is usefull, since if the RPi Python script crashes, the robot stops (and does not continue driving). Command is acknowledged with OK.
- beep ==> beeps (e.g. for fault indication of low battery voltage). Command is acknowledged with OK.
- status ==> returns the status in the form e.g.: 12.8V,0,52,53 ==> battery voltage, status magnet sensor, temperature left motor and temperature right motor
- rst_sens ==> the magnetic detection sensor should be resetted by the RPi. Acknowledged with OK.

The OpenCR board monitors the battery voltage every 5 seconds. If the voltage drops below 11.4V (fixed value in Arudino code), a beep is generated every 5 seconds. The robot keeps on driving. If the voltage drops below 11.0V (critical voltage) the robot stops driving. In this case, the Arduino response to every command with "LOW_BAT". The RPi should check for this, and shuts down to save battery power. Note that the sensors are still active (5V from RPi is no table to shutdown) so power consumption is still significant (estimated around 1W for the RPi part + sensors). A beep is generated every 0.5 seconds. The robot should be turned of immediately, otherwise the battery might be damaged beyond repair.

Concerning the magnetic line detection, this is done in an interrupt routine. This runs at about

60Hz. The sensor is connected to an analog pin of the OpenCR board. An ADC reads the value. A running median is used to filter outliers from the sensor readouts. The threshold are fixed in Arduino code.

The OpenCR has a number of usefull uswer-programmable LEDs. They are currently used as follows:

- User1 LED on: initialisation done, ready for serial communication; blinks: error on initialisation, check serial terminal for error message
- User2 LED on: magnetic line detected
- User3 LED on: processing command
- User4 LED on: low battery alarm active

Routines for fail safe

There are several fail-safes built into the code, since the robot should be able to drive for a long time on its own:

- The OpenCR board checks for low battery voltage: details see above
- The OpenCR board requires a command at least every 800 ms
- The RPi should receive a response from the OpenCR board, no later than 100ms after the command was send (serial time-out). If the response is not as expected (or empty), the RPi resets the OpenCR board.
- The RPi shuts down in case of critical low battery voltage
- The robot beeps at low voltage
- The charging time is limited to a maximum of 25 mins
- When docking, the robot checks for an increase in voltage
- The other way around: if a significant increase in voltage is detected during driving (normally not possible), the robot might have driven into the charging station without magnetic line detection. In this case, the robot turns around.
- The magnetic line should be detected at least every 8 min (default value). If this is not the case (the robot might be stuck at the charging station without touching the charging points), turn around.
- Sometimes the I2C TOF sensors fail. In such case, the script is automatically restarted.

Access Point

A RPi access point in included with the robots. This is only required for debugging. It runs RaspAP. This is router software, with a web interface to configure it. Currently the SSID is cmc-rpi. The router runs DHCP, but performs static IP address lease to the robots based on their MAC address:

- Etherium: 192.168.1.200
- Litecoin: 192.168.1.201
- Monero: 192.168.1.202
- Zcash: 192.168.1.203

If you connect to the same network with a laptop, you can ssh to the robots (or use VNCViewer), and check the status of each of them.