MIDDLEWARE

Design et Autonomisation d'un kart

Projet

KART

Gwendal Priser

Paul-Antoine Le Tolguenec

Gwendal Priser

Gwendal Priser

Gwendal Priser

April 2, 2020

Contents

List of Figures

List of Tables

1 Introduction

2 Hardware

2.1 Main Board

For this project, we decided to use a *Raspberry Pi 3B+* as main board. It will let us plug some sensors and control the motors of the car according to the wanted behavior we have programmed.

On this Raspberry Pi, we need to choose an Operating System. our choice was to use Ubuntu Mate because of its simplicity to install and its polyvalence. It will let us do everything we want, like plug sensors, code any program to control our car, ... A preview of Ubuntu Mate is shown on the 1

As an imposed figure for our project, we decided to use the *Robot Operating System* (ROS) as Middleware for this car. It's going to offer us some practical tools to code our pragrams easily. There is also some usefull community shared tools like *rqt* or *key_telop* we will use in this project.

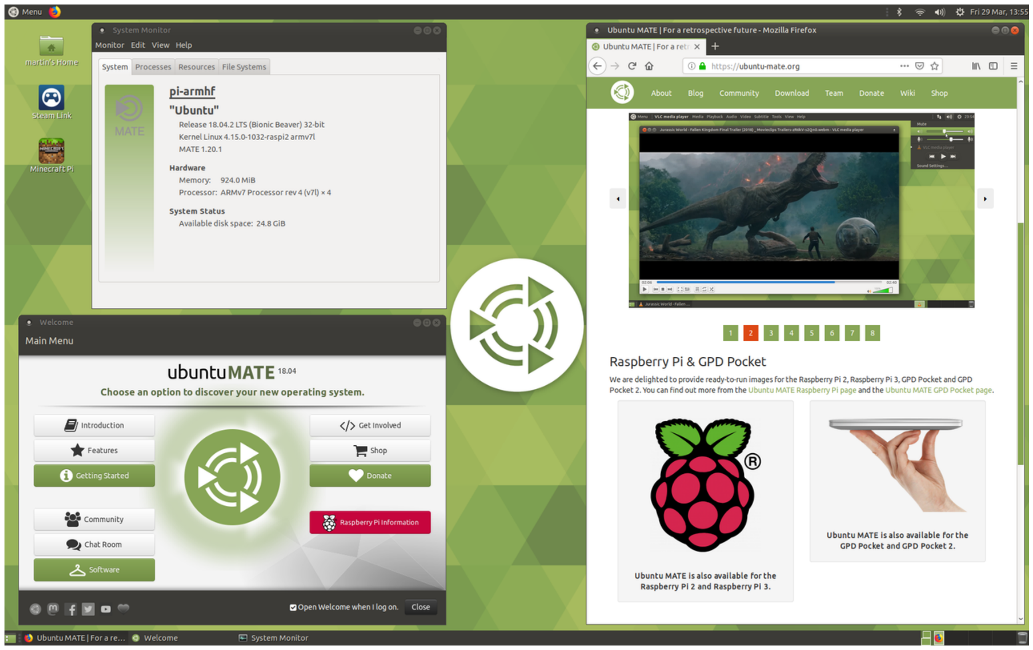
2.2 Sensors

This section present the main hardware configuration of the car. For this project we have to choose which sensors we want in our car in the following list 2.

As it's shown we decided to choose neither the GNSS nor the Inertial Units, mainly because of their accuracy.

Actually, for our problem we found that an accuracy of 1 meter for the *GNSS* is too large because the car need to run in a 0.8 meter wide racing lane, following a line. This sensors is alos not able to know if the car position is correct.

For the *Inertial Unit*, we found that this sensor is too noisy to give us any usefull informations about the state of our car. For instance the consecutive integration of the acceleration in order to get the speed and the position of



pi-armhf
"Ubuntu"
Release 18.04.2 LTS (Bionic Beaver) 32-bit
Kernel Linux 4.15.0-1032-raspi2 armv7l
MATE 1.20.1

Hardware
Memory: 924.0 MiB
Processor: ARMv7 Processor rev 4 (v7l) × 4

System Status
Available disk space: 24.8 GiB

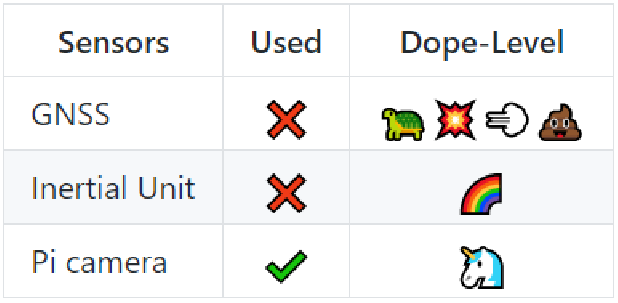
Raspberry Pi & GPD Pocket

We are delighted to provide ready-to-run images for the Raspberry Pi 2, Raspberry Pi 3, GPD Pocket and GPD Pocket 2. You can find out more from the Ubuntu MATE Raspberry Pi page and the Ubuntu MATE GPD Pocket page.

Ubuntu MATE is also available for the Raspberry Pi 2 and Raspberry Pi 3.

Ubuntu MATE is also available for the GPD Pocket and GPD Pocket 2.

Figure 1: Ubuntu Mate screen



Sensors	Used	Dope-Level
GNSS	❌	🐢💥💨💩
Inertial Unit	❌	🌈
Pi camera	✔️	🦄

Figure 2: The Sensors list available on our GitHub

the car leads to an important drift effect on our data. So this sensor is not currently able to to give any correct informations about the state of the car. Moreover, the acceleration of the car could be quite good after filtering if we only needed it. In our case the only usefull information is to know the position of our car in relation to the line.

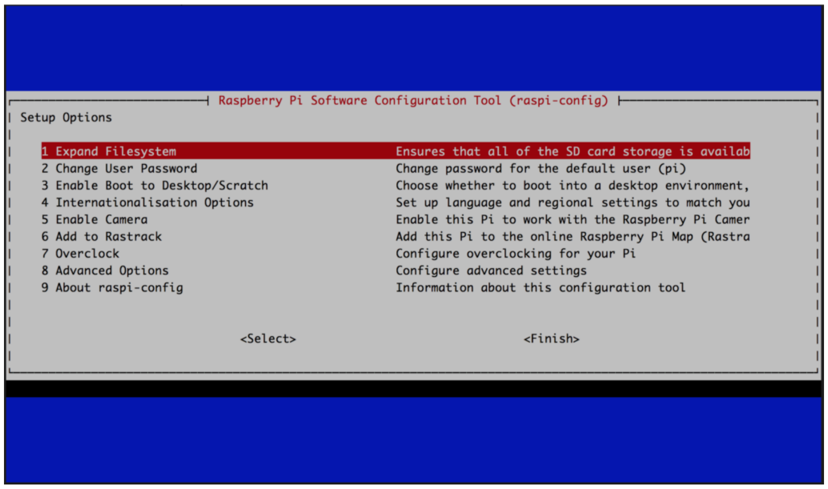
That's why we decided to focus our attention on the camera. Because we are using a *Raspberry Pi 3B+*, the camera we have choosen is the official camera which can be pluged on the dedicated port on the board. This sensor is perfectly suited to our problem, because with an appropriated image processing we will be able to detect the line and to correct the car trajectory.

2.3 Configuration

Now we will explain how we configured our sensors in our project, to let them comunicate with the software and with the car.

2.3.1 Pi Camera

The configuration of the camera on the raspberry pi is relatively simple. We used the *raspi-config* utility to configure the camera. That's how we set up the camera on the Raspberry Pi.



```
Raspberry Pi Software Configuration Tool (raspi-config)
Setup Options

   1 Expand Filesystem                 Ensures that all of the SD card storage is availab
   2 Change User Password              Change password for the default user (pi)
   3 Enable Boot to Desktop/Scratch    Choose whether to boot into a desktop environment,
   4 Internationalisation Options      Set up language and regional settings to match you
   5 Enable Camera                     Enable this Pi to work with the Raspberry Pi Camer
   6 Add to Rastrack                   Add this Pi to the online Raspberry Pi Map (Rastra
   7 Overclock                         Configure overclocking for your Pi
   8 Advanced Options                  Configure advanced settings
   9 About raspi-config                Information about this configuration tool

                  <Select>                          <Finish>
```

Figure 3: raspi-config utiliy on the Raspberry Pi

2.3.2 Hardware PWM

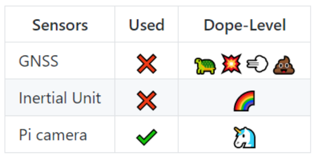
On Raspberry Pi board there is a lot of way to generate pwm signals. The most of the time, these methods are software based and so they are not accurate. With a lot of searches, we found a website who speak about the raspberry pi's hardware pwm signals. There is apparently an hardware pwm generator used by the bord to generate sounds. It's better to use hardware generated pwm, because if the processor has a slow down and the interrupt is not correctly handled, the pwm duty cycle will not be very accurate and the car will not be able for instance to follow a straight line, because the bearing of the car is controled by a servomotor with a pwm signal.

So we decided to use this tutorial : `https://disconnected.systems/blog/pi-zero-w-rover-setup/#moving-the-robot`, which explain us how to setup pwm signals on the Raspberry Pi, and how to correctly configure the files to have a standard pwm signal which is generated. Then we need to give the rights to users for reading and writing in these files. All these bash command are in *init_pwm.sh*.

Then we have to add some automation. So we created a *crontab* rule. That will automatically create all the required files and allow the permissions to every users. We just have to write in the file *duty_cycle* a value between 1.000.000 and 2.000.000, and the Raspberry Pi will read and adapt pwm signals in real time.

3 Image Processing

3.1 Data logging



Sensors	Used	Dope-Level
GNSS	❌	🐢💥💨💩
Inertial Unit	❌	🌈
Pi camera	✔️	🦄

Figure 4: collecting data

For the first part of this image processing, we had to collect images. So first we went into the environment in which the robot was going to evolve.

3.2 pre binarization treatment

In this part, we first had to perform a pre-binarization treatment in order to reduce post-binarization noise. So we used a Gaussian filter to blur the image.

3.3 binarization

Since the line we wanted to mark is white. An effective treatment is simply to switch to grey level. So for binarization, we switch the image to a grey level and threshold for a grey level that we have determined empirically.

3.4 post binarization treatment

In this part we performed a morphological treatment. There was still a lot of noise after binarization. So we made an opening. With a kernel in the shape of a rectangle (since it was the most efficient for this treatment). At the end of this treatment we obtain a well defined line which crosses the screen.

3.5 find the center of the line

In this last part, the contours are marked using a gradient method. Then the contours are sorted from the smallest to the largest. We recover the largest contour. And we recover the coordinates of the barycentre of the contour. Then the error is the difference between the center of the image and the coordinates of the pixel.

4 Explications

5 Resultats

6 Discussions et Conclusion

A Appendix