Voiture Assistée

Travail de diplôme 2021, CFPT-Informatique
V1.0

Ackermann Gawen

Sommaire

01
PROJET

Voiture Assistée

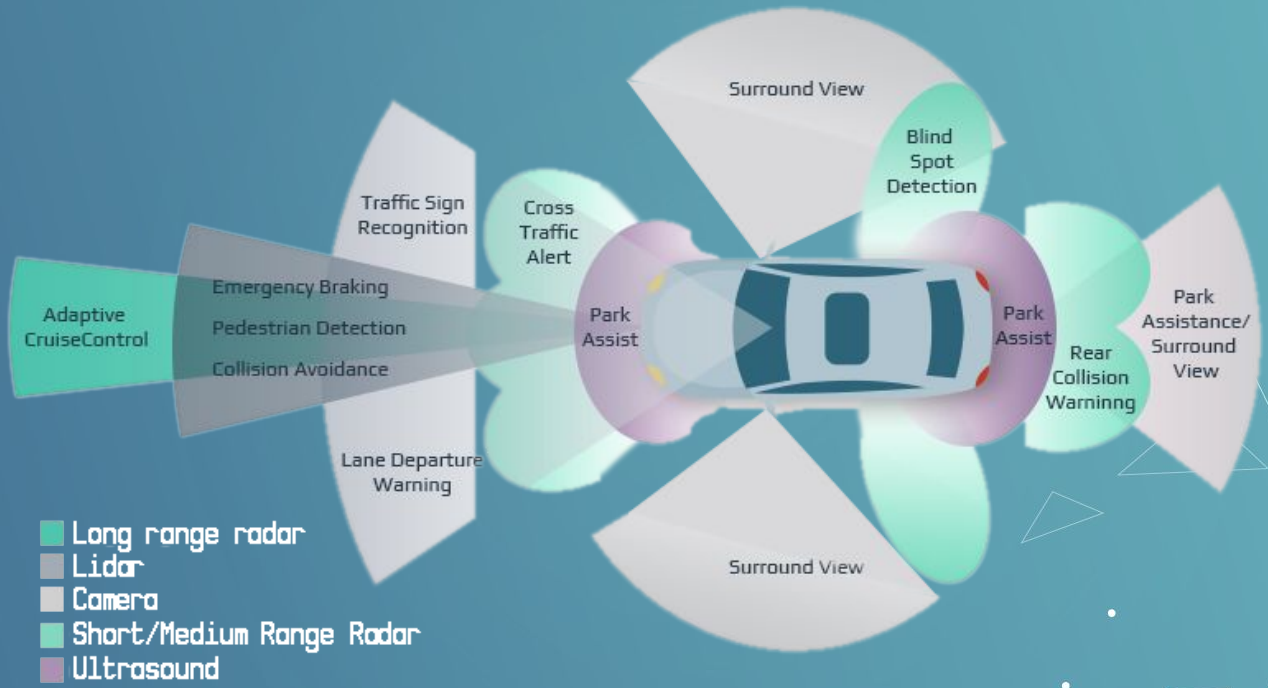
Le but du projet est réaliser une voiture autonome sachant se déplacer en évitant les obstacles sur son chemin. À l'aide d'une interface web, on peut gérer et voir les données des capteurs installés

Enseignant

- M. Bonvin

Technologie existante

Une voiture autonome est souvent constituée des mêmes capteurs



Technologies utilisées

Camera and Headlight
Pi Camera — Bright Pi

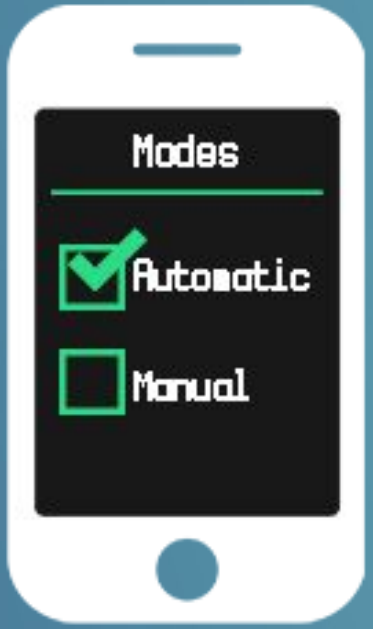
360° Laser scanner
RPLidar A2M8

Ground detection
Flying-Fish

Web interface
Python Flask

Remote control
Python Bluetooth

Technologies utilisées



WiFi connection to the server

Bluetooth connection to the car

02
DÉMONSTRATION

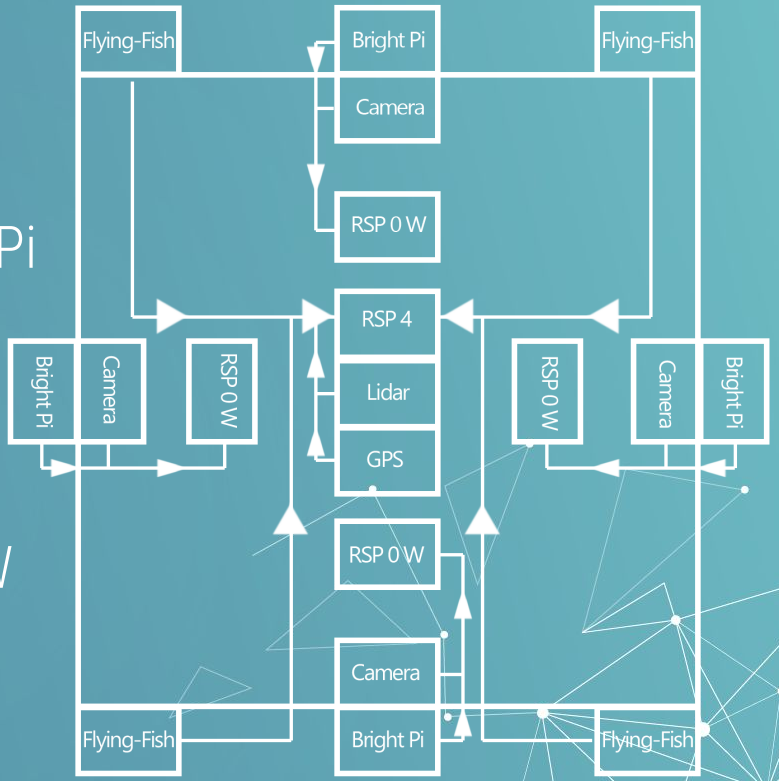
03

FONCTIONNEMENT DE L'APPLICATION

Schéma général

Plan de la voiture et des capteurs attachés :

- Les phares sont représentés par le Bright Pi
- Les détecteurs de sol par les Flying-Fish
- Le radar 360° par le Lidar
- Le raspberry pi principal par RSP 4
- Les raspberry pi des capteurs par RSP 0 W



L’application

L’utilisateur peut depuis l’interface web :

- Se (dé)connecter à la voiture par bluetooth
  - Télécommander la voiture
- Gérer l’état des divers capteurs installés
- Avoir accès aux données fournies par les capteurs
- Gérer l’activation du mode automatique

L'application

L'application est divisé en 2 serveurs différents :

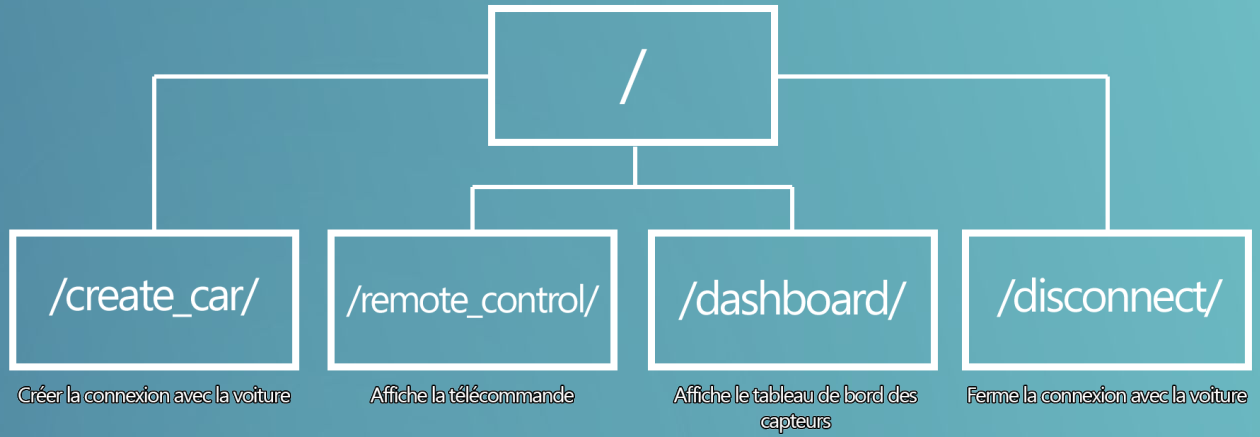
- Serveur principal (connexion à la voiture, télécommande, tableau de bord, …)
- Serveur des capteurs

Le serveur principal tourne sur un Raspberry Pi 4

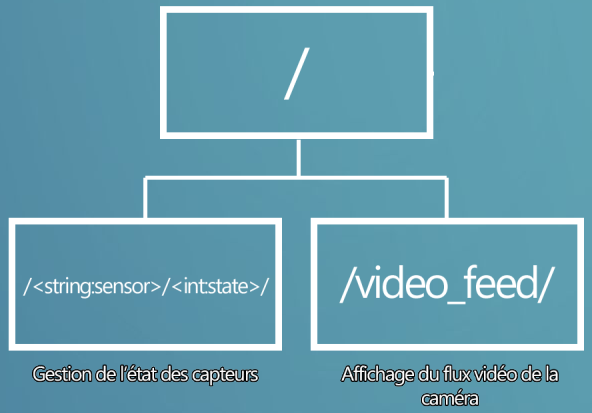
Le serveur des capteurs tourne indépendamment sur chaques Raspberry Pi 0 WiFi

L'application

- Serveur principal



- Serveur des capteurs



04

ÉLÉMENTS TECHNIQUES

Récupération des données du Lidar

- Utilisation de l'API C++ fourni par Slamtec

- Branchement à l'adaptateur

Exécution de l'API C++ avec des processus python

↓

L'API C++ effectue un scan continu, ensuite lisse les données en faisant la moyennes des données précédentes avec les données à l'instant T, puis les affichent dans la console système

↓

Récupération de manière asynchrone (Asyncio) des données émises par l'API dans la console système

↓

Formatage et traitement des données, puis insertion des distances dans un tableau contenant 360 éléments

```python
async def _read_stream(stream, callback):
    """
    Will read the text in the console from the process simple_grabber

    stream : The streaming of the data in the console
    callback : The method to call when data has been received

    """
    while True:
        line = await stream.readline()
        if line:
            callback(line.split(b","))
        else:
            break

def get_radar_data(row):
    """
    Will parse the data received in text by the Lidar

    row : Row to read and to add or modify in the array of angles
    """
    global rows
    # row normaly is like [angle, distance]
    tmp = row
    if len(tmp) == 2:
        angle = int(tmp[0])
        # remove the line return
        dist = tmp[1].replace(b"\n", b"")
        rows[angle] = float(dist)

async def run(should_scan):
    """
    Will run the subprocess and bind the async method

    should_scan : The code to know if the program should scan or not
    """
    command = ("./scanner/simple_grabber /dev/ttyUSB0 " + should_scan).split()
    process = await create_subprocess_exec(*command, stdout=PIPE, stderr=PIPE)
    await asyncio.wait([_read_stream(process.stdout, lambda x: {get_radar_data(x)})])
    await process.wait()


async def main(should_scan):
    """
    The main function which calls the run loop async

    should_scan : The code to know if the program should scan or not
    """
    await run(should_scan)



@app.route("/bg_processing_lidar/<string:state>", methods=["POST"])
def bg_process_lidar(state=None):
    """Process the values passed by Javascript"""

    try:
        loop = asyncio.get_running_loop()
    except RuntimeError:  # no event loop running:
        loop = asyncio.new_event_loop()
    finally:
        loop.run_until_complete(main(state))

    return ""
```

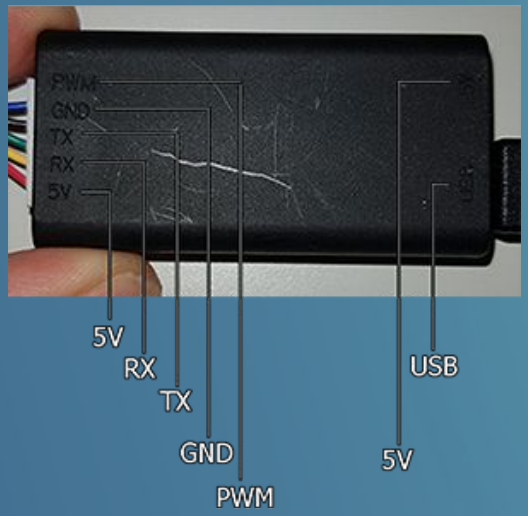
Code C++ modifié

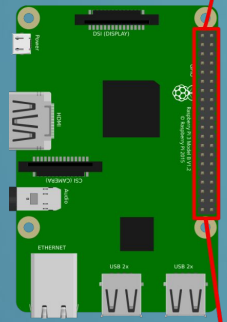
```
int max_size_arr_angle_dist = 360;
int nb_scan = 10;
float angle_dist_tmp[max_size_arr_angle_dist];
float angle_dist[max_size_arr_angle_dist];
int size_arr_angle_dist = (sizeof(angle_dist) / sizeof(angle_dist[0]));

const int MAX_RANGE_LIDAR = 16000.0f;
const int MIN_RANGE_LIDAR = 0.0f;
// used as a code in order to treat the data below in another process
//printf("<<<\n");
while (run_scan)
{
    //reset the array to empty (empty represented by the 0)
    std::fill_n(angle_dist, max_size_arr_angle_dist, 0.0f);
    capture_and_display(angle_dist, drv);
    if (size_arr_angle_dist == 0)
    {
        fprintf(stderr, "Error, cannot grab scan data.\n");
        break;
    }
    /* Correct the errors */
    for (size_t j = 0; j < max_size_arr_angle_dist; j++)
    {
        if (angle_dist[j] < MIN_RANGE_LIDAR && angle_dist[j] >= MAX_RANGE_LIDAR)
        {
            angle_dist[j] = MIN_RANGE_LIDAR;
        }
    }
    /* make the average of distance */
    capture_and_display(angle_dist_tmp, drv);
    for (size_t j = 0; j < max_size_arr_angle_dist; j++)
    {
        if (angle_dist_tmp[j] > MIN_RANGE_LIDAR && angle_dist_tmp[j] < MAX_RANGE_LIDAR)
        {
            angle_dist[j] = (angle_dist[j] + angle_dist_tmp[j]) / 2;
        }
    }
    //system("clear");
    for (size_t j = 0; j < max_size_arr_angle_dist; j++)
    {
        printf("%d,%f\n", j, angle_dist[j]);
    }
}
```

```
float *capture_and_display(float angle_dist[], RPlidarDriver *drv)
{
    u_result ans;
    int old_angle = 0;
    float threshold_range = 2000.0f; // equal to 1 meter 50
    float max_range = 16000.0f;      // equal to the max range of the sensor which is 16 meters
    rplidar_response_measurement_node_t nodes[8192];
    size_t count = _countof(nodes);

    // fetch extactly one 0-360 degrees' scan
    ans = drv->grabScanData(nodes, count);
    if (IS_OK(ans) || ans == RESULT_OPERATION_TIMEOUT)
    {
        drv->ascendScanData(nodes, count);
        for (int pos = 0; pos < (int)count; ++pos)
        {
            float angle = (nodes[pos].angle_q6_checkbit >> RPLIDAR_RESP_MEASUREMENT_ANGLE_SHIFT) / 64.0f;
            float distance = nodes[pos].distance_q2 / 4.0f;
            int rounded_angle = (int)round(angle);
            (distance <= 0) ? distance = 0 : distance;
            (distance > threshold_range || distance > max_range) ? distance = 0 : distance;
            angle_dist[rounded_angle] = distance;
        }
    }
    else
    {
        printf("error code: %x\n", ans);
    }
    return angle_dist;
}
```

Récupération des données du Lidar

- Dessouder les câbles branchés au port série pour les souder sur le GPIO du Raspberry Pi 4



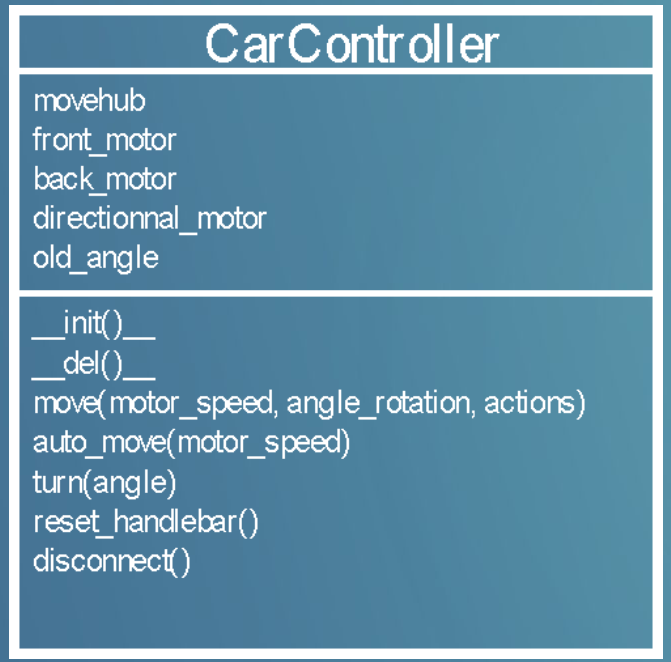


Connexion à la voiture avec le Bluetooth

- Tentative de connexion à l'aide de gatt
  - Connexion établie, mais toujours pas télécommandable
- Tentative d'envoi de commandes en bytes

```python
import gatt

LEGO_Hub_Service = "00001623-1212-EFDE-1623-785FEABCD123"
LEGO_Hub_Characteristic = "00001624-1212-EFDE-1623-785FEABCD123"

class AnyDevice(gatt.Device):

    def connect_succeeded(self):
        """Print the mac adress of the connected device
        """
        super().connect_succeeded()
        print("[%s] Connected" % (self.mac_address))

    def connect_failed(self, error):
        """Print the mac adress of the device that failed to connect
        """
        super().connect_failed(error)
        print("[%s] Connection failed: %s" % (self.mac_address, str(error)))

    def disconnect_succeeded(self):
        """Print the mac adress of the device disconnected
        """
        super().disconnect_succeeded()
        print("[%s] Disconnected" % (self.mac_address))

    def services_resolved(self):
        """Print the services and its characteristics if they are similar to the Lego Hub UUID
        """
        super().services_resolved()
        print("[%s] Resolved services" % (self.mac_address))
        for service in self.services:
            if service == LEGO_Hub_Service:
                print("[%s]  Service [%s]" % (self.mac_address, service.uuid))
                for characteristic in service.characteristics:
                    if str(characteristic.uuid) == LEGO_Hub_Characteristic:
                        print("[%s]    Characteristic [%s]" % (self.mac_address, characteristic.uuid))



# Initialize the manager with the hci0 adaptater
manager = gatt.DeviceManager(adapter_name='hci0')
# Create the device
device = AnyDevice(mac_address='90:84:2B:50:36:43', manager=manager)
# Connect self to device
device.connect()
# The main loop that is necessary to receive Bluetooth events from the Bluetooth adapter
manager.run()
# It can be stopped by executing the stop() method
```

Connexion à la voiture avec le Bluetooth




```python
def __init__(cls):
    """Create the connection with the car"""
    cls.connection = get_connection_gatt(hub_mac=cls.MY_MOVEHUB_ADD)
    try:
        # The motors
        cls.movehub = MoveHub(cls.connection)
        cls.front_motor = Motor(cls.movehub, cls.movehub.PORT_A)
        cls.back_motor = Motor(cls.movehub, cls.movehub.PORT_B)
        cls.directionnal_motor = EncodedMotor(cls.movehub, cls.movehub.PORT_C)
        cls.old_angle = cls.DEFAULT_ANGLE
    except:
        cls.movehub = None
        cls.front_motor = None
        cls.back_motor = None
        cls.directionnal_motor = None
        cls.instance = None
        cls.connection = None
        cls.old_angle = None
```

```python
def get_connection_gatt(controller='hci0', hub_mac=None, hub_name=None):
    from pylgbst.comms.cgatt import GattConnection

    return GattConnection(controller).connect(hub_mac, hub_name)
```

05
CONCLUSION

Ce qui devait être fait

Réaliser une interface web accessible depuis le téléphone qui permet :

- De contrôler la voiture
  - D’activer le mode automatique
- De gérer les capteurs installés
  - De voir les données des capteurs
- D’enregistrer les données GPS
- Afficher les obstacles encadrés avec leurs distance par rapport à la voiture sur les caméras
- D’enregistrer les actions menées par la voiture

Ce qui a été réalisé

- Interface web accessible depuis le téléphone
- Contrôle de la voiture
  - D’activer le mode automatique
- De gérer les capteurs installés
  - De voir les données des capteurs

Ce qu’il reste à faire

- D’enregistrer les données GPS
- Afficher les obstacles encadrés avec leurs distance par rapport à la voiture sur les caméras
- D’enregistrer les actions menées par la voiture

06

Merci de votre attention,
Avez-vous des questions ?


slidesgo