Indoor real-time navigation for robot vehicles

Final year project for the specialization program in Embedded Systems Engineering

Submitted by:
Liu Qin
Niu Siyuan

Tutor: Sébastien Bilavarn

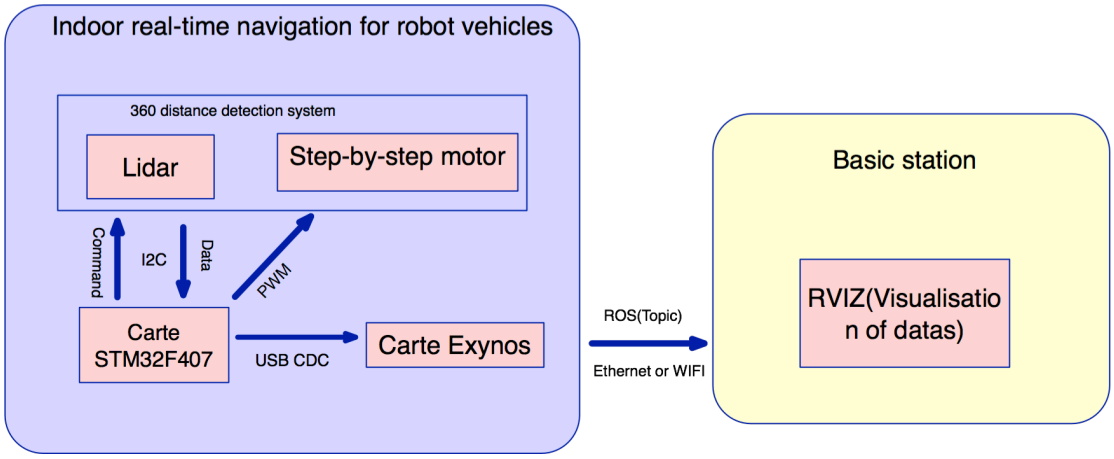
1.Introduction:

The goal of this project is to design a real-time navigation system for a robot vehicle. The acquisition device is based on a Lidar controlled by a STM32 MCU and an Exynos board. They can both communicate through a serial / USB connection and we use UART to do this. The Lidar data is streamed to a PC base station running SLAM and visualization software. The acquisition device and the base station communicate using Robot Operating System (ROS) and a Wifi connection. The SLAM algorithm is based on Google Cartographer and a 3D visualization tool called RVIZ, both integrated in ROS. We want to realize that when we move the Lidar around and collect data with it, we can watch the map get created/updated in real time on a remote base station. Firstly, we install ROS and Google Cartographer on both PC workstation and Exynos board. Then, we configure Lidar, UART, motor and Wifi module. Finally, we connect them together and watch the map by using RVIZ.

2.Specification

We use a lidar model that can scan its 360 surroundings with a motor and an infrared laser. The lidar will be controlled by the STM32f407 development board, an ARM Cortex-M microcontroller that is a high-performance microcontroller. Its development environment supports Keil MDK-ARM and ST-Link/V2. We use the I2C protocol for communication on this board. Among them, the lidar data will be retrieved by the STM32f407 card and transmitted to the Exynos card using the USB protocol (uart). The Exynos card and base station will communicate by using the ROS (Robot Operating System) operating system over a Wi-Fi connection. On a desktop computer used as a base station, we use the Google Cartographer and RVIZ visualization tools of the ROS operating system for map visualization.

3.Overall view of the system



The figure above illustrates the design architecture of the real-time navigation system. The central controller is STM32F4Discovery, the purpose of the STM32 board is to configure the lidar as we needed.
And then it can read the distance and convert it to an integer value in centimeters. It includes the UART unit, the PWM unit, and the I2C unit. The scanning lidar module allows the lidar to realize a whole scan of the robot environment (360°) in a two-dimensional space. It consists of a lidar and a step-by-step motor. These two parts are synchronized and the step-by-step motor is in one Rotate two turns in seconds and use it to scan 360 data in a circle. Finally, for laser scanner equipment (Lidar), which uses I2C for data conversion, I2C needs to be configured firstly, and so it is convenient to transmit/receive data and design step-by-step motors through STM32. The main function of Exynos is to receive data from STM32 and use ROS to transfer data to PC via WiFi, so we need to install ROS on Exynos board (only Ubuntu system is supported). Use UART serial transmission between the STM32 and Exynos boards. On the PC base station, we also installed ROS, so that the two stations can use WiFi to communicate before. Finally, the published values of the sub-software RVIZ that starts ROS on the base station are mapped.

4. Advanced conception

The entire project is divided into two departments, hardware and software. Here we will discuss hardware and software in detail and their respective contributions to the project. There are four kinds of hardware devices on the robot vehicle, which are laser radar, stepper motor, STM32 and Exynos board. STM32 is used to control sensor data transmission, PWM, Exynos and PC base stations communicate remotely via ROS and WiFi.

4.1 Hardware

• Lidar Lite v3<Appendix1>
Lidar is a sensor that measures the distance by laser reflection, and the distance is calculated from the time difference between transmission and reception. The lidar used in this project is Lidar-Lite v3HP. This is a laser module for measuring distances from 0 to 40 meters. The module can communicate with the Arduino microcontroller via a PWM or I2C protocol. An operating mode corresponding to the tradeoff between accuracy, measurement range and acquisition time can be set. In this system, it is fixed on the step-by-step motor and measures the distance between the vehicle and the surrounding obstacles in real time as the motor moves. In order to achieve I2C communication, the four lines on the lidar need to be connected to the STM32: power, ground, SCL and SDA. The lidar transmits data to the STM32 via I2C.

• NEMA17 stepper motor <Appendix2>
The stepper motor module consists of a motor (NEMA17) and a motor driver

(A4988). The drive controls the stepper motor by inputting the direction of rotation and stepping. For stepper motors, the step size is 1.8 and requires additional power at 12V. The bipolar stepper motor Nema 17 has good accuracy and extremely low vibration and noise levels. It will be used as a lidar support for a full 360° scanning environment.

• Hardware cabling <Appendix3>

4.2 Software

•µVision programming
Our main software development platform is keil µVision, which is a software development system compatible with single-chip C language. Keil offers a complete development solution including C compiler, macro assembly, linker, library management and a powerful emulator debugger, which are combined through an integrated development environment (µVision).

• Configure Lidar to transfer and receive data <Appendix4>
We use the µVision environment to configure the lidar, I2C link between the lidar and the STM32 board to control the lidar in order to perform measurements. Once the I2C link is configured, we start to configure the lidar. This is done by writing a hexadecimal value corresponding to the lidar register. There are several modes of operation. In order to optimize the performance of our system, we will use the "fast mode of Lidar"<Appendix 4> indicated in the document, allowing 400kbit/s. The distance measured by the lidar is saved to two registers of the lidar, including the high and low bits of the data.
The specific code can be found in Appendix6-Lidar 2D-(2)

• Motor configuration
After Lidar's operation and configuration are verified, we add the configuration of the step-by-step motor. A lidar will be attached to perform a full scan. Since our NEMA 17 engine has 200 steps, and we want to realize that the speed of motor is 2 turns/second, so that there is an interruption last 2.5ms between each step. In the final procedure, the distance returned by the lidar is retrieved at the same frequency as the motor is rotated, in other words, a step corresponds to a distance. Each time the motor rotates, the distance data detected by lidar is sent to the Exynos card via UART.
See the Appendix6(3) for specific code.

• Start button configuration
In order for the system to start and end work under manual control. We add the configuration of the button.
The specific code is detailed in Apeendix6- IV

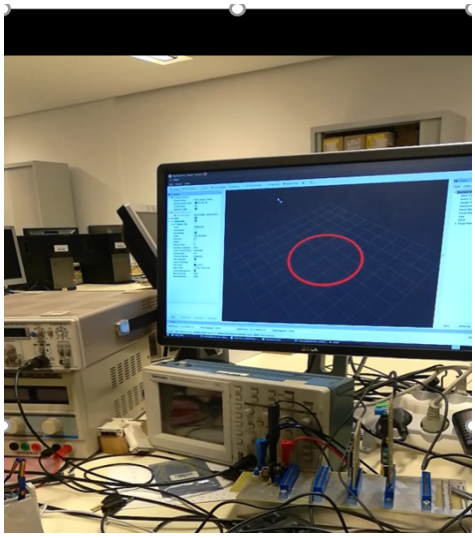
• ROS + Exynos
First, we downloaded the demo of fakeLaserScan on git. This demo is ready to use. Based on our own situation, we changed the parameters of the data published by lasrscan so that the actual data could be released. It should be emphasized that angle_min and angle_max correspond to positive and negative PI (3.14), that corresponds 360 degrees around. It can be noticed that the default values of ranges[] and intensitys[] are constant by default, so we added the part that reads the transmitted data from the usb port and puts the data in range[]. This enables real-time measurement of data via ROS on the Exynos board.

• Set ros / rviz
See Appendix6-(3) for details of the command.
Firstly, we need to launch fakelasersacn, then open rviz by typing the commands in the terminal, the rviz visual interface appears. Then we click on join node and add the laserscan node. It should be noted that we need to change the name of the id frame to be the same as the name set in the demo. Finally, adjust the size of the displayed points.

• wifi configuration
See <Appendix6-3.SLAM> for configuration details.
First, we tried the Exynos connection wifi, which is transmitted between the base station PC connected to the Ethernet (wired). But it didn't work, so we connected Exynos to Ethernet and then communicated with the PC. We suspect that it may be the reason for network incompatibility. Finally, we connected Exynos and PC to campus wifi, and then communicated between the two stations, which was equally successful.

5.Test and Results

We have realized to show Lidar data directly in software RVIZ. However, each circle shown in the following pictures correspond to an obstacle detected by Lidar not each point corresponds to an obstacle. Except this problem, we can assure that the communication with ROS between Lidar, STM32 card and Exynos works well. Besides, we have achieved the remote communication between Exynos and PC base workstation via Ethernet and via Wifi.
Following problem of association Lidar + motor we could not test our RVIZ display settings corresponded with the actual operation of our system. And we don’t have enough time to integrate the Lidar device with a battery such that the system can be fully autonomous and mobile.



Link of the video of our demo:
https://drive.google.com/file/d/1RP6c2uct3u-JYB1dpVSFE8M1L1agLF-D/view?usp=sharing

6. Conclusion

Although we haven't finished our project, we have reached most of our objectives and validated most of the project concerning the display of Lidar data on the RVIZ software. During the project, we have learned to communicate between Exynos and PC base station using ROS. We were constrained by lack of time and the problem of connection between Lidar and motor. The next step we will do is to integrate the Lidar device with a battery to realize a fully autonomous and mobile system.

Appendix :

Appendix1 : Lidar Lite v3 Specifications

Electrical

Specification	Measurement
Power	5 Vdc nominal 4.5 Vdc min., 5.5 Vdc max.
Current consumption	105 mA idle 135 mA continuous operation

Performance

Specification	Measurement
Range (70% reflective target)	40 m (131 ft)
Resolution	+/- 1 cm (0.4 in.)
Accuracy < 5 m	±2.5 cm (1 in.) typical*
Accuracy ≥ 5 m	±10 cm (3.9 in.) typical Mean ±1% of distance maximum Ripple ±1% of distance maximum
Update rate (70% Reflective Target)	270 Hz typical 650 Hz fast mode** >1000 Hz short range only
Repetition rate	~50 Hz default 500 Hz max

*Nonlinearity present below 1 m (39.4 in.)

**Reduced sensitivity

Interface

Specification	Measurement
User interface	I2C PWM External trigger
I2C interface	Fast-mode (400 kbit/s) Default 7-bit address 0x62 Internal register access & control
PWM interface	External trigger input PWM output proportional to distance at 10 µs/cm

Appendix 2: NEMA 17 stepper Motor Specification

ELECTRICAL

parameter	conditions/description	min	typ	max	units
power supply	VDD	4.5	5	5.5	V
start up time			200		ms
current consumption	with unloaded output		16		mA
output high level		VDD-0.1			V
output low level				0.1	V
output current (per channel)				15	mA
rise/fall time			8		ns

INCREMENTAL CHARACTERISTICS

parameter	conditions/description	min	typ	max	units
channels	CMOS Voltage: A, B, Z				
waveform	CMOS voltage square wave				
phase difference	A leads B for CCW rotation (viewed from front)				
quadrature resolutions[1]	48, 96, 100, 125, 192, 200, 250, 256, 360, 384, 400, 500, 512, 768, 800, 1000, 1024, 1600, 2000, 2048, 2500, 4096				PPR
index[2]	one pulse per 360 degree rotation				
accuracy			0.2		degrees
quadrature duty cycle			50		%

Notes: 1. Resolution programmed with AMT Viewpoint™ PC software. Default resolution set to 400 PPR.
2. Zero position alignment set with AMT One Touch Zero™ module, AMT Viewpoint™ PC software, or serial commands

Appendix 3: Component cabling

1) Lidar-stm32f4 card

Lidar	Stm32f4
Red wire	5v
Orange wire	No connect
Yellow wire	No connect

Green wire	PB6(SCL)
Blue wire	PB7(SDA)
Black wire	GND

2) Motor-stm32f4 card

Motor	**Stm32f4**
VDD	5v
GND	GND
Step	PB9
Direction	PB8

3) UART-stm32f4 card

Motor	**Stm32f4**
GND	GND
RXD	PA2
TXD	No connect

Appendix 4 : Lidar configuration in « fast mode »

Sending 0x00
Sending 0x01
Sending 0x04
Sending 0x08
Sending 0x02
Sending 0x1d
Pooling on bit 0 of the state register (0x01)
reading the data in 0x10
reading the data in 0x0f
Distance obtained by concatenation of the 2 values read

Appendix 5: Links

-References
LIDAR_Lite_v3_Operation_Manual_and_Technical_Specifications.pdf
https://www.cui.com/product/resource/nema17-amt112s.pdf
-git

Appendix 6: Steps and code

1. ROS cartographer

1) Install Robotic Operating System (ROS)

- Install ROS (we choose *ROS Kinetic Kame*)

```
$ sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu $(lsb_release -
```

```
sc) main" > /etc/apt/sources.list.d/ros-latest.list'
$ sudo apt-key adv --keyserver hkp://ha.pool.sks-keyservers.net:80 --recv-
key 421C365BD9FF1F717815A3895523BAEEB01FA116
$ sudo apt-get update
$ sudo apt-get install ros-kinetic-desktop-full
$ apt-cache search ros-kinetic
```

-Initialize ROS

```
$ sudo rosdep init
$ rosdep update
$ echo "source /opt/ros/kinetic/setup.bash" >> ~/.bashrc
$ source ~/.bashrc
$ sudo apt-get install python-rosinstall
```

-Test ROS

```
$ roscore
```

2) Install Google Cartographer

```
$ sudo apt-get update
$ sudo apt-get install -y python-wstool python-rosdep ninja-build
$ mkdir catkin_ws
$ cd catkin_ws
$ wstool init src
$ wstool merge -t src
$ https://raw.githubusercontent.com/googlecartographer/cartographer_ros/mast
er/cartographer_ros.rosinstall
$ wstool update -t src
$ src/cartographer/scripts/install_proto3.sh
$ sudo rosdep init
$ rosdep update
$ rosdep install --from-paths src --ignore-src --rosdistro=${ROS_DISTRO} -y
$ catkin_make_isolated --install --use-ninja
```

Note: We use command `$ sudo apt-get install gparted`
to install software which is used to resize the partition because google cartographer takes up too much space.

3) Run a demo

-Download a demo

```
$ mkdir fakelaserscan
$ cd fakelaserscan
$ mkdir src
$ cd src
$ git clone https://github.com/erenaud3/fakeLaserScan
```

```
$ cd ..
$ catkin_make
$ source ~/catkin_ws1/devel/setup.bash
$ source ~/fakelaserscan/devel/setup.bash
```

-Run the demo

```
$ roslaunch beginner_tutorials fakeLaserScan.launch
$ rostopic echo -n1 /fakeScan
$ rosrun rviz rviz
```

-Run a node

```
$ roscore
$ rosnode list
$ rosnode info /rosout
$ rosrun [package_name] [node_name]
$ rosrun turtlesim turtlesim_node
```

-Analyze the key parameters

```
float32 angle_min        # start angle of the scan [rad]

float32 angle_max        # end angle of the scan [rad]

float32 angle_increment  # angular distance between measurements [rad]

float32 time_increment   # time between measurements [seconds]

float32 scan_time        # time between scans [seconds]

float32 range_min        # minimum range value [m]

float32 range_max        # maximum range value [m]

float32[] ranges         # range data [m] (Note: values < range_min or >
range_max should be discarded)

float32[] intensities    # intensity data [device-specific units]
```

2. Lidar 2D

1) UART communication

I. Configure UART

```
/*la partie uart*/
void USART_Puts(USART_TypeDef* USARTx,volatile char *s)
{
 while(*s)
 {
  while(!(USARTx ->SR & 0x00000040));//envoyer->0x04
         USART_SendData(USARTx,*s);
         *s++;
 }
}


int USART2_transmitter_empty(void){
        return ( USART_GetFlagStatus(USART2, USART_FLAG_TXE) == SET );
}


void USART2_puts(char *s){
        while( *s ){
                if(USART2_transmitter_empty()) {
                                USART_SendData(USART2, *s);
                                s++;
                }
        }
}
```

```
void UART_Initialisation(){
  GPIO_InitTypeDef GPIO_InitStructure;
       USART_InitTypeDef USART_InitStructure;
              /*configuration uart*/

       RCC_APB1PeriphClockCmd(RCC_APB1Periph_USART2,ENABLE);
       RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOA,ENABLE);//P2A

       GPIO_InitStructure.GPIO_Mode=GPIO_Mode_AF;
       GPIO_InitStructure.GPIO_Pin=GPIO_Pin_2;
       GPIO_InitStructure.GPIO_PuPd=GPIO_PuPd_UP;
       GPIO_InitStructure.GPIO_Speed=GPIO_Speed_100MHz;

       GPIO_Init(GPIOA,&GPIO_InitStructure);
       //juste connecter PA2 avec rx  parce que c'est la carte qui envoie les donne■s , uart recoit les donne■s
       GPIO_PinAFConfig(GPIOA,GPIO_PinSource2,GPIO_AF_USART2);
       GPIO_PinAFConfig(GPIOA, GPIO_PinSource3, GPIO_AF_USART2);

       USART_InitStructure.USART_BaudRate=115200;
       USART_InitStructure.USART_HardwareFlowControl=USART_HardwareFlowControl_None;
       USART_InitStructure.USART_Mode=USART_Mode_Tx;
       USART_InitStructure.USART_Parity=USART_Parity_No;
       USART_InitStructure.USART_StopBits=USART_StopBits_1;
       USART_InitStructure.USART_WordLength=USART_WordLength_8b;
       USART_Init(USART2,&USART_InitStructure);
       USART_Cmd(USART2,ENABLE);
}
```

II.C program on the Exynos to read data via UART (uart.c)

```c
#include <stdio.h>
#include <fcntl.h>   /* File Control Definitions           */
#include <termios.h> /* POSIX Terminal Control Definitions */
#include <unistd.h>  /* UNIX Standard Definitions          */
#include <errno.h>   /* ERROR Number Definitions           */

int main(void){
        int fd;/*File Descriptor*/

               printf("\n +----------------------------------+");
               printf("\n |        Serial Port Read          |");
               printf("\n +----------------------------------+");

               /*------------------------------- Opening the Serial Port -------------------------------*/

               /* Change /dev/ttyUSB0 to the one corresponding to your system */

               fd = open("/dev/ttyUSB0",O_RDWR | O_NOCTTY);    /* ttyUSB0 is the FT232 based USB2SERIAL Converter   */
                                                               /* O_RDWR   - Read/Write access to serial port       */
                                                               /* O_NOCTTY - No terminal will control the process   */
                                                               /* Open in blocking mode,read will wait              */



               if(fd == -1)                                            /* Error Checking */
                  printf("\n  Error! in Opening ttyUSB0  ");
               else
                  printf("\n  ttyUSB0 Opened Successfully ");


               /*---------- Setting the Attributes of the serial port using termios structure --------- */

               struct termios SerialPortSettings;      /* Create the structure                          */

               tcgetattr(fd, &SerialPortSettings);     /* Get the current attributes of the Serial port */


              /* Setting the Baud rate */
              cfsetispeed(&SerialPortSettings,B115200); /* Set Read  Speed as 9600                       */
              cfsetospeed(&SerialPortSettings,B115200); /* Set Write Speed as 9600                       */

              /* 8N1 Mode */
              SerialPortSettings.c_cflag &= ~PARENB;   /* Disables the Parity Enable bit(PARENB),So No Parity   */
              SerialPortSettings.c_cflag &= ~CSTOPB;   /* CSTOPB = 2 Stop bits,here it is cleared so 1 Stop bit */
              SerialPortSettings.c_cflag &= ~CSIZE;    /* Clears the mask for setting the data size             */
              SerialPortSettings.c_cflag |=  CS8;      /* Set the data bits = 8                                 */

              SerialPortSettings.c_cflag &= ~CRTSCTS;       /* No Hardware flow Control                         */
              SerialPortSettings.c_cflag |= CREAD | CLOCAL; /* Enable receiver,Ignore Modem Control lines       */


              SerialPortSettings.c_iflag &= ~(IXON | IXOFF | IXANY);          /* Disable XON/XOFF flow control both i/p and o/p */
              SerialPortSettings.c_iflag &= ~(ICANON | ECHO | ECHOE | ISIG);  /* Non Cannonical mode                            */

              SerialPortSettings.c_oflag &= ~OPOST;/*No Output Processing*/

              /* Setting Time outs */
              SerialPortSettings.c_cc[VMIN] = 10; /* Read at least 10 characters */
              SerialPortSettings.c_cc[VTIME] = 0; /* Wait indefinetly   */


              if((tcsetattr(fd,TCSANOW,&SerialPortSettings)) != 0) /* Set the attributes to the termios structure*/
                  printf("\n  ERROR ! in Setting attributes");
              else
             intf("\n  BaudRate = 115200 \n  StopBits = 1 \n  Parity   = none");

              /*------------------------------- Read data from serial port -----------------------------*/

              tcflush(fd, TCIFLUSH);   /* Discards old data in the rx buffer            */

              char read_buffer[32];   /* Buffer to store the data received              */

              int  bytes_read = 0;    /* Number of bytes read by the read() system call */
              int i = 0;
              int tempo;

                while(1){
                bytes_read = read(fd,&read_buffer,32); /* Read the data                   */

                printf("\n\n  Bytes Rxed -%d", bytes_read); /* Print the number of bytes read */
                printf("\n\n  ");

                for(i=0;i<bytes_read;i++)        /*printing only the received characters*/
                    printf("%c",read_buffer[i]);

                printf("\n +----------------------------------+\n\n\n");
        //usleep(500000);
                tempo=600000;//la vitess pour recevoir les donnees de lidar
                while(tempo)
                        tempo--;
                }

                close(fd); /* Close the serial port */

        }
```

2) Lidar control with STM32

I. Configure Lidar

```
/*la partie I2C*/
void I2C1_Initialize(void)
{
        I2C_InitTypeDef I2C_init1;
        GPIO_InitTypeDef GPIO_InitStructure1;
        GPIO_InitTypeDef GPIO_InitStructure2;

        I2C_init1.I2C_ClockSpeed = 100000;
        I2C_init1.I2C_Mode = I2C_Mode_I2C;
        I2C_init1.I2C_DutyCycle = I2C_DutyCycle_2;
        I2C_init1.I2C_OwnAddress1=0;
        I2C_init1.I2C_Ack = I2C_Ack_Disable;
        I2C_init1.I2C_AcknowledgedAddress = I2C_AcknowledgedAddress_7bit;

        RCC_APB1PeriphClockCmd(RCC_APB1Periph_I2C1, ENABLE);

        RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOB, ENABLE);

        //GPIO_PinConfigure(GPIOB, 6, GPIO_MODE_AF, GPIO_OUTPUT_OPEN_DRAIN, GPIO_GPOUTPUT_SPEED_50MHz, GPIO_PULL_UP);


         GPIO_InitStructure1.GPIO_Pin=GPIO_Pin_6;
         GPIO_InitStructure1.GPIO_Mode=GPIO_Mode_AF;
         GPIO_InitStructure1.GPIO_OType=GPIO_OType_OD;
         GPIO_InitStructure1.GPIO_PuPd=GPIO_PuPd_UP;
         GPIO_InitStructure1.GPIO_Speed=GPIO_Fast_Speed;
         GPIO_Init(GPIOB,&GPIO_InitStructure1);
                //GPIO_PinConfigure(GPIOB, 7, GPIO_MODE_AF, GPIO_OUTPUT_OPEN_DRAIN, GPIO_OUTPUT_SPEED_50MHz, GPIO_PULL_UP);

         GPIO_InitStructure2.GPIO_Pin=GPIO_Pin_7;
         GPIO_InitStructure2.GPIO_Mode=GPIO_Mode_AF;
         GPIO_InitStructure2.GPIO_OType=GPIO_OType_OD;
         GPIO_InitStructure2.GPIO_PuPd=GPIO_PuPd_UP;
         GPIO_InitStructure2.GPIO_Speed=GPIO_Fast_Speed;
         GPIO_Init(GPIOB,&GPIO_InitStructure2);


        //GPIO_PinAF(GPIOB, 6, GPIO_AF_I2C1);   SCL  green
        //GPIO_PinAF(GPIOB, 7, GPIO_AF_I2C1);     SDA  blue
        GPIO_PinAFConfig(GPIOB, 6,GPIO_AF_I2C1);
        GPIO_PinAFConfig(GPIOB, 7,GPIO_AF_I2C1);

        I2C_Init(I2C1, &I2C_init1);

        I2C_Cmd(I2C1, ENABLE);
}
```

```
void I2C_start(I2C_TypeDef* I2Cx, uint8_t address, uint8_t direction)
{
        while(I2C_GetFlagStatus(I2Cx, I2C_FLAG_BUSY) == SET);
        I2C_GenerateSTART(I2Cx, ENABLE);
        while(I2C_CheckEvent(I2Cx, I2C_EVENT_MASTER_MODE_SELECT) == ERROR);
        I2C_Send7bitAddress(I2Cx, address, direction);
        if (direction == I2C_Direction_Transmitter){
                while(I2C_CheckEvent(I2Cx, I2C_EVENT_MASTER_TRANSMITTER_MODE_SELECTED) == ERROR);
        } else if (direction == I2C_Direction_Receiver){
                while(I2C_CheckEvent(I2Cx, I2C_EVENT_MASTER_RECEIVER_MODE_SELECTED) == ERROR);
        }
}

uint16_t get_distance(){
        uint8_t mask;
  uint8_t receive;
        uint8_t distance_faible;
        uint8_t distance_fort;
        uint16_t distance;

  I2C_start(I2C1, 0x62<<1, I2C_Direction_Transmitter);
        I2C_SendData(I2C1, 0x00);
        while(I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_BYTE_TRANSMITTED) == ERROR);
        I2C_SendData(I2C1, 0x01);
        I2C_GenerateSTOP(I2C1, ENABLE);

        I2C_start(I2C1, 0x62<<1, I2C_Direction_Transmitter);
        I2C_SendData(I2C1, 0x04);
        while(I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_BYTE_TRANSMITTED) == ERROR);
        I2C_SendData(I2C1, 0x08);
        while(I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_BYTE_TRANSMITTED) == ERROR);
        I2C_GenerateSTOP(I2C1, ENABLE);

        I2C_start(I2C1, 0x62<<1, I2C_Direction_Transmitter);
        I2C_SendData(I2C1, 0x01);
        while(I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_BYTE_TRANSMITTED) == ERROR);
        I2C_GenerateSTOP(I2C1, ENABLE);


        I2C_start(I2C1, 0x62<<1, I2C_Direction_Receiver);
        mask=0x00000001;
        while(I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_BYTE_RECEIVED) == ERROR);
        receive=I2C_ReceiveData(I2C1);
        I2C_GenerateSTOP(I2C1, ENABLE);
```

```
        while((receive&mask)!=0){
              I2C_start(I2C1, 0x62<<1, I2C_Direction_Transmitter);
              I2C_SendData(I2C1, 0x01);
              while(I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_BYTE_TRANSMITTED) == ERROR);
              I2C_GenerateSTOP(I2C1, ENABLE);

              I2C_start(I2C1, 0x62<<1, I2C_Direction_Receiver);
              while(I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_BYTE_RECEIVED) == ERROR);
              receive=I2C_ReceiveData(I2C1);
              I2C_GenerateSTOP(I2C1, ENABLE);
        }

        //while((receive&mask)!=0);
        I2C_start(I2C1, 0x62<<1, I2C_Direction_Transmitter);
        I2C_SendData(I2C1, 0x10);
        while(I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_BYTE_TRANSMITTED) == ERROR);
        I2C_GenerateSTOP(I2C1, ENABLE);
        I2C_start(I2C1, 0x62<<1, I2C_Direction_Receiver);
        while(I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_BYTE_RECEIVED) == ERROR);
        distance_faible=I2C_ReceiveData(I2C1);
        I2C_GenerateSTOP(I2C1, ENABLE);

        I2C_start(I2C1, 0x62<<1, I2C_Direction_Transmitter);
        I2C_SendData(I2C1, 0x0f);
        while(I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_BYTE_TRANSMITTED) == ERROR);
        I2C_GenerateSTOP(I2C1, ENABLE);
        I2C_start(I2C1, 0x62<<1, I2C_Direction_Receiver);
        while(I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_BYTE_RECEIVED) == ERROR);
        distance_fort=I2C_ReceiveData(I2C1);

        distance= (distance_fort<<8)+distance_faible;
        I2C_GenerateSTOP(I2C1, ENABLE);


        return distance;

}
```

3) Control the stepper motor and scan 360

Configure motor

```
//pwm
__IO uint32_t usTick=2500;
void SysTick_Handler(){
usTick--;
        GPIO_SetBits(GPIOB,GPIO_Pin_9);
        if(usTick==0){
          GPIO_ResetBits(GPIOB,GPIO_Pin_9);
                usTick=2500;
        }
}


void PWMinit(){
         GPIO_InitTypeDef GPIO_InitStructure1;
         GPIO_InitTypeDef GPIO_InitStructure2;

         RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOB, ENABLE);
         GPIO_InitStructure1.GPIO_Pin=GPIO_Pin_9;
         GPIO_InitStructure1.GPIO_Mode=GPIO_Mode_OUT;
         GPIO_InitStructure1.GPIO_OType=GPIO_OType_OD;
         GPIO_InitStructure1.GPIO_PuPd=GPIO_PuPd_UP;
         GPIO_InitStructure1.GPIO_Speed=GPIO_Fast_Speed;
         GPIO_Init(GPIOB,&GPIO_InitStructure1);

         GPIO_InitStructure2.GPIO_Pin=GPIO_Pin_8;
         GPIO_InitStructure2.GPIO_Mode=GPIO_Mode_OUT;
         GPIO_InitStructure2.GPIO_OType=GPIO_OType_OD;
         GPIO_InitStructure2.GPIO_PuPd=GPIO_PuPd_UP;
         GPIO_InitStructure2.GPIO_Speed=GPIO_Fast_Speed;
         GPIO_Init(GPIOB,&GPIO_InitStructure2);

}
```

4) Lidar device demo

I.fakeLaserScan.cpp

```
#include <ros/ros.h>
#include <sensor_msgs/LaserScan.h>
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>   /* File Control Definitions           */
#include <termios.h> /* POSIX Terminal Control Definitions */
#include <unistd.h>  /* UNIX Standard Definitions          */
#include <errno.h>   /* ERROR Number Definitions           */
#include "std_msgs/String.h"

#include <sstream>

int main(int argc, char** argv){
  ros::init(argc, argv, "fake_laser_scan_publisher");// ros::init(argc, argv, "talker");

  ros::NodeHandle n;//ok
  ros::Publisher Scan_pub = n.advertise<sensor_msgs::LaserScan>("fakeScan", 50);
  ros::Publisher chatter_pub = n.advertise<std_msgs::String>("chatter", 1000);


  unsigned int num_readings = 200;
  double laser_frequency = 400;
  double ranges[num_readings];
  double intensities[num_readings];

  int count = 0;
//-----------------------------------------------------------------------------------------------------

               int fd;/*File Descriptor*/

       fd = open("/dev/ttyUSB0",O_RDWR | O_NOCTTY);    /* ttyUSB0 is the FT232 based USB2SERIAL Converter   */
               struct termios SerialPortSettings;      /* Create the structure                          */

               tcgetattr(fd, &SerialPortSettings);     /* Get the current attributes of the Serial port */

               /* Setting the Baud rate */
               cfsetispeed(&SerialPortSettings,B115200); /* Set Read  Speed as 9600                       */
               cfsetospeed(&SerialPortSettings,B115200); /* Set Write Speed as 9600                       */

               /* 8N1 Mode */
               SerialPortSettings.c_cflag &= ~PARENB;   /* Disables the Parity Enable bit(PARENB),So No Parity   */
               SerialPortSettings.c_cflag &= ~CSTOPB;   /* CSTOPB = 2 Stop bits,here it is cleared so 1 Stop bit */
               SerialPortSettings.c_cflag &= ~CSIZE;    /* Clears the mask for setting the data size             */
               SerialPortSettings.c_cflag |=  CS8;      /* Set the data bits = 8                                 */

               SerialPortSettings.c_cflag &= ~CRTSCTS;       /* No Hardware flow Control                         */
               SerialPortSettings.c_cflag |= CREAD | CLOCAL; /* Enable receiver,Ignore Modem Control lines       */


                SerialPortSettings.c_iflag &= ~(IXON | IXOFF | IXANY);          /* Disable XON/XOFF flow control both i/p and o/p */
                SerialPortSettings.c_iflag &= ~(ICANON | ECHO | ECHOE | ISIG);  /* Non Cannonical mode                            */

                SerialPortSettings.c_oflag &= ~OPOST;/*No Output Processing*/

                /* Setting Time outs */
                SerialPortSettings.c_cc[VMIN] = 10; /* Read at least 10 characters */
                SerialPortSettings.c_cc[VTIME] = 0; /* Wait indefinetly   */



                /*------------------------------- Read data from serial port -----------------------------*/

                tcflush(fd, TCIFLUSH);   /* Discards old data in the rx buffer            */

//-----------------------------------------------------------------------------------------------------
                char read_buffer[32];   /* Buffer to store the data received              */
                int  bytes_read = 0;    /* Number of bytes read by the read() system call */
                int i = 0;
                //int tempo;
                //while(1){



//-----------------------------------------------------------------------------------------------------
  ros::Rate r(400);//  ros::Rate loop_rate(10);
  while(n.ok()){
//------------------------------------------------------------------------
 std_msgs::String msg;
 std::stringstream ss;
//------------------------------------------------------------------
bytes_read = read(fd,&read_buffer,32); /* Read the data
                */
//for(i=0;i<bytes_read;i++)      /*printing only the received characters*/
        //          printf("fonction get_lidar: %c\n",read_buffer[i]);

                double distance=atoi(read_buffer);
    //generate some fake data for our laser scan
    for(unsigned int i = 0; i < num_readings; ++i){

          ranges[i] =distance/100.0;
      intensities[i] = 100 + count;
    }
    ros::Time scan_time = ros::Time::now();
```

```
  //populate the LaserScan message
  sensor_msgs::LaserScan scan;
  scan.header.stamp = scan_time;
  scan.header.frame_id = "fake_laser_frame";
  scan.angle_min = -3.14;
  scan.angle_max = 3.14;
  scan.angle_increment = 6.28 / num_readings;
 // scan.time_increment = (1 / laser_frequency) / (num_readings);
      scan.time_increment = 1 / laser_frequency;
  scan.range_min = 0.0;
  scan.range_max = 100.0;

  scan.scan_time = (1/2);

  scan.ranges.resize(num_readings);
  scan.intensities.resize(num_readings);
  for(unsigned int i = 0; i < num_readings; ++i){
    scan.ranges[i] = ranges[i];
    scan.intensities[i] = intensities[i];
  }
  ss <<count<< "distance mesure " << distance;
  msg.data = ss.str();
  ROS_INFO("%s", msg.data.c_str());
  chatter_pub.publish(msg);

  ros::spinOnce();

  Scan_pub.publish(scan);
  ++count;
  r.sleep();//


}
```

II. talker.cpp

```
#include <sstream>

/**
 * This tutorial demonstrates simple sending of messages over the ROS system.
 */
int main(int argc, char **argv)
{
  /**
   * The ros::init() function needs to see argc and argv so that it can perform
   * any ROS arguments and name remapping that were provided at the command line.
   * For programmatic remappings you can use a different version of init() which takes
   * remappings directly, but for most command-line programs, passing argc and argv is
   * the easiest way to do it.  The third argument to init() is the name of the node.
   *
   * You must call one of the versions of ros::init() before using any other
   * part of the ROS system.
   */
  ros::init(argc, argv, "talker");

  /**
   * NodeHandle is the main access point to communications with the ROS system.
   * The first NodeHandle constructed will fully initialize this node, and the last
   * NodeHandle destructed will close down the node.
   */
  ros::NodeHandle n;

 /**
  * The advertise() function is how you tell ROS that you want to
  * publish on a given topic name. This invokes a call to the ROS
  * master node, which keeps a registry of who is publishing and who
  * is subscribing. After this advertise() call is made, the master
  * node will notify anyone who is trying to subscribe to this topic name,
  * and they will in turn negotiate a peer-to-peer connection with this
  * node.  advertise() returns a Publisher object which allows you to
  * publish messages on that topic through a call to publish().  Once
  * all copies of the returned Publisher object are destroyed, the topic
  * will be automatically unadvertised.
  *
  * The second parameter to advertise() is the size of the message queue
  * used for publishing messages.  If messages are published more quickly
  * than we can send them, the number here specifies how many messages to
  * buffer up before throwing some away.
  */
 ros::Publisher chatter_pub = n.advertise<std_msgs::String>("chatter", 1000);

 ros::Rate loop_rate(10);

 /**
  * A count of how many messages we have sent. This is used to create
  * a unique string for each message.
  */
 int count = 0;
```

```
  while (ros::ok())
  {
    /**
     * This is a message object. You stuff it with data, and then publish it.
     */
    std_msgs::String msg;

    std::stringstream ss;
    ss << "hello world " << count;
    msg.data = ss.str();

    ROS_INFO("%s", msg.data.c_str());

    /**
     * The publish() function is how you send messages. The parameter
     * is the message object. The type of this object must agree with the type
     * given as a template parameter to the advertise<>() call, as was done
     * in the constructor above.
     */
    chatter_pub.publish(msg);

    ros::spinOnce();

    loop_rate.sleep();
    ++count;
  }


  return 0;
}
```

III. listener.cpp

```
#include "ros/ros.h"
#include "std_msgs/String.h"

/**
 * This tutorial demonstrates simple receipt of messages over the ROS system.
 */
void chatterCallback(const std_msgs::String::ConstPtr& msg)
{
  ROS_INFO("I heard: [%s]", msg->data.c_str());
}

int main(int argc, char **argv)
{
  /**
   * The ros::init() function needs to see argc and argv so that it can perform
   * any ROS arguments and name remapping that were provided at the command line.
   * For programmatic remappings you can use a different version of init() which takes
   * remappings directly, but for most command-line programs, passing argc and argv is
   * the easiest way to do it.  The third argument to init() is the name of the node.
   *
   * You must call one of the versions of ros::init() before using any other
   * part of the ROS system.
   */
  ros::init(argc, argv, "listener");

  /**
   * NodeHandle is the main access point to communications with the ROS system.
   * The first NodeHandle constructed will fully initialize this node, and the last
   * NodeHandle destructed will close down the node.
   */
  ros::NodeHandle n;


  /**
   * The subscribe() call is how you tell ROS that you want to receive messages
   * on a given topic.  This invokes a call to the ROS
   * master node, which keeps a registry of who is publishing and who
   * is subscribing.  Messages are passed to a callback function, here
   * called chatterCallback.  subscribe() returns a Subscriber object that you
   * must hold on to until you want to unsubscribe.  When all copies of the Subscriber
   * object go out of scope, this callback will automatically be unsubscribed from
   * this topic.
   *
   * The second parameter to the subscribe() function is the size of the message
   * queue.  If messages are arriving faster than they are being processed, this
   * is the number of messages that will be buffered up before beginning to throw
   * away the oldest ones.
   */
  ros::Subscriber sub = n.subscribe("chatter", 1000, chatterCallback);

  /**
   * ros::spin() will enter a loop, pumping callbacks.  With this version, all
   * callbacks will be called from within this thread (the main one).  ros::spin()
   * will exit when Ctrl-C is pressed, or the node is shutdown by the master.
   */
  ros::spin();

  return 0;
}
```

IV. Start button

```
//Start button
void buttonInitialization(){

        GPIO_InitTypeDef GPIO_InitDef;
        GPIO_InitDef.GPIO_Pin=GPIO_Pin_0;
        GPIO_InitDef.GPIO_Mode=GPIO_Mode_IN;
        GPIO_InitDef.GPIO_OType=GPIO_OType_PP;
        GPIO_InitDef.GPIO_PuPd=GPIO_PuPd_DOWN;
        GPIO_InitDef.GPIO_Speed=GPIO_Speed_50MHz;
        RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOA,ENABLE);
        GPIO_Init(GPIOA,&GPIO_InitDef);
}
```

3. SLAM

ROS network setup
On choose:
master: Exynos (name:odroid)
slave: pc (name: administreur-OptiPlex-9020)

-Preparatory work

```
$ ifconfig
$ hostname
$ gksu gedit /etc/hosts
$ chmod -R 777 /etc/hosts
$ sudo /etc/init.d/networking restart
```

-Realize communication between two computers

```
$ sudo apt-get install chrony
$ sudo apt-get install openssh-server
$ ps -e|grep ssh

$ ssh odroid (master)
$ ssh administreur-OptiPlex-9020 (slave)

$ ping odroid (master)
$ ping administreur-OptiPlex-9020 (master)

$ ping administreur-OptiPlex-9020 (slave)
$ ping odroid (slave)

$ netcat -l 12345 (master)
$ netcat odroid 1234 (slave)

$ export ROS_MASTER_URI=http://odroid:11311
$ cd catkin_ws1

$ source ~/catkin_ws1/devel/setup.bash
```

```
$ rosrun beginner_tutorials talker
$ rosrun beginner_tutorials listener
```

4. Complete code(trans.c)

```
#include "stm32f4xx.h"                  // Device header
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include "tm_stm32f4_delay.h"


char str[50];
uint32_t i;

uint16_t distance;

/*la partie uart*/
void USART_Puts(USART_TypeDef* USARTx,volatile char *s)
{
 while(*s)
 {
  while(!(USARTx ->SR & 0x00000040));//envoyer->0x04
         USART_SendData(USARTx,*s);
         *s++;
 }
}


int USART2_transmitter_empty(void){
        return ( USART_GetFlagStatus(USART2, USART_FLAG_TXE) == SET );
}


void USART2_puts(char *s){
        while( *s ){
                if(USART2_transmitter_empty()) {
                                USART_SendData(USART2, *s);
                                s++;
                }
        }
}


void UART_Initialisation(){
  GPIO_InitTypeDef GPIO_InitStructure;
        USART_InitTypeDef USART_InitStructure;
                /*configuration uart*/

        RCC_APB1PeriphClockCmd(RCC_APB1Periph_USART2,ENABLE);
        RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOA,ENABLE);//P2A

        GPIO_InitStructure.GPIO_Mode=GPIO_Mode_AF;
        GPIO_InitStructure.GPIO_Pin=GPIO_Pin_2;
        GPIO_InitStructure.GPIO_PuPd=GPIO_PuPd_UP;
        GPIO_InitStructure.GPIO_Speed=GPIO_Speed_100MHz;

        GPIO_Init(GPIOA,&GPIO_InitStructure);
        //juste connecter PA2 avec rx  parce que c'est la carte qui envoie les donne , uart recoit les donne
        GPIO_PinAFConfig(GPIOA,GPIO_PinSource2,GPIO_AF_USART2);
        GPIO_PinAFConfig(GPIOA, GPIO_PinSource3, GPIO_AF_USART2);

        USART_InitStructure.USART_BaudRate=115200;
        USART_InitStructure.USART_HardwareFlowControl=USART_HardwareFlowControl_None;
        USART_InitStructure.USART_Mode=USART_Mode_Tx;
        USART_InitStructure.USART_Parity=USART_Parity_No;
        USART_InitStructure.USART_StopBits=USART_StopBits_1;
        USART_InitStructure.USART_WordLength=USART_WordLength_8b;
        USART_Init(USART2,&USART_InitStructure);
        USART_Cmd(USART2,ENABLE);
}
```

```c
/*la partie I2C*/
void I2C1_Initialize(void)
{
	I2C_InitTypeDef I2C_init1;
	GPIO_InitTypeDef GPIO_InitStructure1;
	GPIO_InitTypeDef GPIO_InitStructure2;

	I2C_init1.I2C_ClockSpeed = 100000;
	I2C_init1.I2C_Mode = I2C_Mode_I2C;
	I2C_init1.I2C_DutyCycle = I2C_DutyCycle_2;
	I2C_init1.I2C_OwnAddress1=0;
	I2C_init1.I2C_Ack = I2C_Ack_Disable;
	I2C_init1.I2C_AcknowledgedAddress = I2C_AcknowledgedAddress_7bit;

	RCC_APB1PeriphClockCmd(RCC_APB1Periph_I2C1, ENABLE);

	RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOB, ENABLE);

	//GPIO_PinConfigure(GPIOB, 6, GPIO_MODE_AF, GPIO_OUTPUT_OPEN_DRAIN, GPIO_GPOUTPUT_SPEED_50MHz, GPIO_PULL_UP);


	 GPIO_InitStructure1.GPIO_Pin=GPIO_Pin_6;
	 GPIO_InitStructure1.GPIO_Mode=GPIO_Mode_AF;
	 GPIO_InitStructure1.GPIO_OType=GPIO_OType_OD;
	 GPIO_InitStructure1.GPIO_PuPd=GPIO_PuPd_UP;
	 GPIO_InitStructure1.GPIO_Speed=GPIO_Fast_Speed;
	 GPIO_Init(GPIOB,&GPIO_InitStructure1);
		//GPIO_PinConfigure(GPIOB, 7, GPIO_MODE_AF, GPIO_OUTPUT_OPEN_DRAIN, GPIO_OUTPUT_SPEED_50MHz, GPIO_PULL_UP);

	 GPIO_InitStructure2.GPIO_Pin=GPIO_Pin_7;
	 GPIO_InitStructure2.GPIO_Mode=GPIO_Mode_AF;
	 GPIO_InitStructure2.GPIO_OType=GPIO_OType_OD;
	 GPIO_InitStructure2.GPIO_PuPd=GPIO_PuPd_UP;
	 GPIO_InitStructure2.GPIO_Speed=GPIO_Fast_Speed;
	 GPIO_Init(GPIOB,&GPIO_InitStructure2);


	//GPIO_PinAF(GPIOB, 6, GPIO_AF_I2C1);   SCL  green
	//GPIO_PinAF(GPIOB, 7, GPIO_AF_I2C1);     SDA  blue
	GPIO_PinAFConfig(GPIOB, 6,GPIO_AF_I2C1);
	GPIO_PinAFConfig(GPIOB, 7,GPIO_AF_I2C1);

	I2C_Init(I2C1, &I2C_init1);

	I2C_Cmd(I2C1, ENABLE);
}


void I2C_start(I2C_TypeDef* I2Cx, uint8_t address, uint8_t direction)
{
	while(I2C_GetFlagStatus(I2Cx, I2C_FLAG_BUSY) == SET);
	I2C_GenerateSTART(I2Cx, ENABLE);
	while(I2C_CheckEvent(I2Cx, I2C_EVENT_MASTER_MODE_SELECT) == ERROR);
	I2C_Send7bitAddress(I2Cx, address, direction);
	if (direction == I2C_Direction_Transmitter){
		while(I2C_CheckEvent(I2Cx, I2C_EVENT_MASTER_TRANSMITTER_MODE_SELECTED) == ERROR);
	} else if (direction == I2C_Direction_Receiver){
		while(I2C_CheckEvent(I2Cx, I2C_EVENT_MASTER_RECEIVER_MODE_SELECTED) == ERROR);
	}
}

uint16_t get_distance(){
	uint8_t mask;
  uint8_t receive;
	uint8_t distance_faible;
	uint8_t distance_fort;
	uint16_t distance;

  I2C_start(I2C1, 0x62<<1, I2C_Direction_Transmitter);
	I2C_SendData(I2C1, 0x00);
	while(I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_BYTE_TRANSMITTED) == ERROR);
	I2C_SendData(I2C1, 0x01);
	I2C_GenerateSTOP(I2C1, ENABLE);

	I2C_start(I2C1, 0x62<<1, I2C_Direction_Transmitter);
	I2C_SendData(I2C1, 0x04);
	while(I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_BYTE_TRANSMITTED) == ERROR);
	I2C_SendData(I2C1, 0x08);
	while(I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_BYTE_TRANSMITTED) == ERROR);
	I2C_GenerateSTOP(I2C1, ENABLE);

	I2C_start(I2C1, 0x62<<1, I2C_Direction_Transmitter);
	I2C_SendData(I2C1, 0x01);
	while(I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_BYTE_TRANSMITTED) == ERROR);
	I2C_GenerateSTOP(I2C1, ENABLE);


	I2C_start(I2C1, 0x62<<1, I2C_Direction_Receiver);
	mask=0x00000001;
	while(I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_BYTE_RECEIVED) == ERROR);
	receive=I2C_ReceiveData(I2C1);
	I2C_GenerateSTOP(I2C1, ENABLE);
```

```
          while((receive&mask)!=0){
                I2C_start(I2C1, 0x62<<1, I2C_Direction_Transmitter);
                I2C_SendData(I2C1, 0x01);
                while(I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_BYTE_TRANSMITTED) == ERROR);
                I2C_GenerateSTOP(I2C1, ENABLE);

                I2C_start(I2C1, 0x62<<1, I2C_Direction_Receiver);
                while(I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_BYTE_RECEIVED) == ERROR);
                receive=I2C_ReceiveData(I2C1);
                I2C_GenerateSTOP(I2C1, ENABLE);
        }

        //while((receive&mask)!=0);
        I2C_start(I2C1, 0x62<<1, I2C_Direction_Transmitter);
        I2C_SendData(I2C1, 0x10);
        while(I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_BYTE_TRANSMITTED) == ERROR);
        I2C_GenerateSTOP(I2C1, ENABLE);
        I2C_start(I2C1, 0x62<<1, I2C_Direction_Receiver);
        while(I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_BYTE_RECEIVED) == ERROR);
        distance_faible=I2C_ReceiveData(I2C1);
        I2C_GenerateSTOP(I2C1, ENABLE);

        I2C_start(I2C1, 0x62<<1, I2C_Direction_Transmitter);
        I2C_SendData(I2C1, 0x0f);
        while(I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_BYTE_TRANSMITTED) == ERROR);
        I2C_GenerateSTOP(I2C1, ENABLE);
        I2C_start(I2C1, 0x62<<1, I2C_Direction_Receiver);
        while(I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_BYTE_RECEIVED) == ERROR);
        distance_fort=I2C_ReceiveData(I2C1);

        distance= (distance_fort<<8)+distance_faible;
        I2C_GenerateSTOP(I2C1, ENABLE);


        return distance;

}
```

```
//Start button
void buttonInitialization(){

        GPIO_InitTypeDef GPIO_InitDef;
        GPIO_InitDef.GPIO_Pin=GPIO_Pin_0;
        GPIO_InitDef.GPIO_Mode=GPIO_Mode_IN;
        GPIO_InitDef.GPIO_OType=GPIO_OType_PP;
        GPIO_InitDef.GPIO_PuPd=GPIO_PuPd_DOWN;
        GPIO_InitDef.GPIO_Speed=GPIO_Speed_50MHz;
        RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOA,ENABLE);
        GPIO_Init(GPIOA,&GPIO_InitDef);
}

//partie pwm

__IO uint32_t usTick=2500;
void SysTick_Handler(){
usTick--;
        GPIO_SetBits(GPIOB,GPIO_Pin_9);
        if(usTick==0){
          GPIO_ResetBits(GPIOB,GPIO_Pin_9);
                usTick=2500;
        }
}
```

```
void PWMinit(){
        GPIO_InitTypeDef GPIO_InitStructure1;
        GPIO_InitTypeDef GPIO_InitStructure2;

        RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOB, ENABLE);
        GPIO_InitStructure1.GPIO_Pin=GPIO_Pin_9;
        GPIO_InitStructure1.GPIO_Mode=GPIO_Mode_OUT;
        GPIO_InitStructure1.GPIO_OType=GPIO_OType_OD;
        GPIO_InitStructure1.GPIO_PuPd=GPIO_PuPd_UP;
        GPIO_InitStructure1.GPIO_Speed=GPIO_Fast_Speed;
        GPIO_Init(GPIOB,&GPIO_InitStructure1);

        GPIO_InitStructure2.GPIO_Pin=GPIO_Pin_8;
        GPIO_InitStructure2.GPIO_Mode=GPIO_Mode_OUT;
        GPIO_InitStructure2.GPIO_OType=GPIO_OType_OD;
        GPIO_InitStructure2.GPIO_PuPd=GPIO_PuPd_UP;
        GPIO_InitStructure2.GPIO_Speed=GPIO_Fast_Speed;
        GPIO_Init(GPIOB,&GPIO_InitStructure2);

}
```

```c
int main(void)
{
        buttonInitialization();
        UART_Initialisation();
        SystemCoreClockUpdate();
        GPIO_SetBits(GPIOB,GPIO_Pin_8);

        //configuration de I2C
 while(1){
        if(GPIO_ReadInputDataBit(GPIOA,GPIO_Pin_0)){
        I2C1_Initialize();

      while(1){
         //mesure distance
         SysTick_Config(SystemCoreClock / 1000000);
         distance=get_distance();
         sprintf(str,"%d\n",distance);
         USART_Puts(USART2,str);

                        }
                }
        }
}
```