Programming of mobile robots

Thomas MAUGIN and Adrien LIOTARD

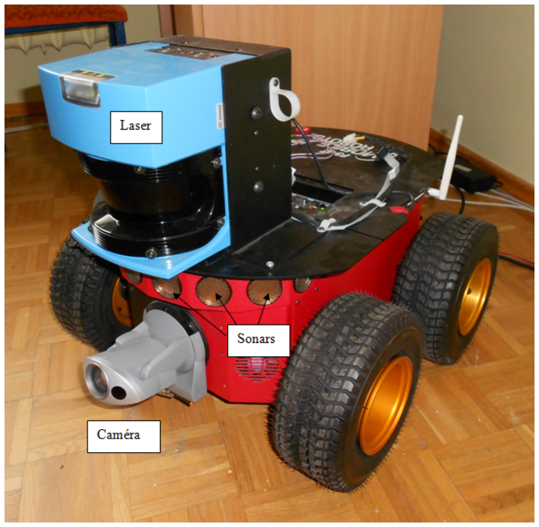
May - July 2013

Contents

Introduction

The main goal of our internship at the Institute of Control and Computation Engineering of the University of Zielona Gora was to learn how to control different mobile robots. The two platforms we used during these 3 months are an AmigoBot and a Pioneer 3-AT, both manufactured by Adept MobileRobots (`http://www.mobilerobots.com/Mobile_Robots.aspx`). Most of the programs we created to control the robots were written in C++ using Microsoft Visual Studio and Aria libraries. Aria is the special set of tools provided by Adept MobileRobots to program and control their robots and all related hardware such as sonar, laser or camera. You can read further information and download it here: `http://robots.mobilerobots.com/wiki/ARIA`.

Amigobot



Pioneer 3-AT with camera and laser

During the first half of the internship, we programmed several movement commands on the AmigoBot, like lines, circles, squares, and moving to a specified position. We also used a webcam connected to a computer running Matlab to implement a color tracking algorithm. Then in the second half of the project, we adapt those programs to run on the P3AT, along with the use of new features like the integrated pan/tilt/zoom camera and the laser rangefinder.

Camera Canon VC-C50i

Laser Sick LMS 200

In the following chapters, you will see in details how we implement each task, with the corresponding pieces of code and further explanation. To use these codes inside Microsoft Visual Studio C++, remember to first open an existing example project in `C:\ProgramFiles\MobileRobots\Aria\examples`, and then replace the .cpp source code by the one you want. You must do this in order to keep the dependencies with Aria libraries.

Chapter 1

Basic movement functions

1.1 Linear motion

The first things we need to control the Amigobot are basic movement tasks. In fact any trajectory made by the robot can be decomposed in two simple movements: translation and rotation. That's why we have to firstly implement these two kinds of movement as functions, so we can then build more complex paths by combining those two functions. Before coding a function, we have to wonder what parameters we need for it. In the case of the translation (named "Go"), we want the robot to move forward to a specified distance with a specified speed. So we have 3 inputs for this task: distance and speed values, and an ArRobot parameter for the robot itself. So the prototype of the Go function is:

int Go(ArRobot *robot,double Speed,double Dist)

Now we can start the programming of this task. One important thing to know is that every robot's motors command must be set between lock() and unlock() calls, and after enableMotors() call. The integrated function to make the robot to move forward is setVel(Speed), and we want to do this while the robot has not reached the end of the required distance. So we put this command inside a "while" loop with the condition "current travelled distance is inferior to final distance". Obviously we must continuously actualize the value of the travelled distance, so we do this by using getVel(). As getVel() only gives the current speed, we need to multiply it by the sample time "dt" in order to obtain the sample distance. Once the robot has reached the goal it must stop, so we set setVel() to 0 after the loop.

When we tested this program on the real robot we noticed strange behaviours due to a bad estimation of the current robot's position. We found that it was caused by the acceleration and deceleration phases at the beginning and end of the task. That's why we added conditions inside the loop to adjust the robot's speed in terms of the travelled distance. We also added some "printf" calls to monitor this task in the console. You can see the final code below:

```
int Go(ArRobot *robot,double Speed,double Dist)//go forward at the specified distance with the
    specified speed
{
    printf("Go forward : %f mm at Speed : %f mm/s\n",Dist,Speed);
    double Dist0=0;
    double dt=200;
    while(Dist0<Dist)
    {
        robot->lock();
        robot->enableMotors();
        if((Dist-Dist0)<300)
        {
            robot->setVel((Dist-Dist0)/300*Speed+5);//adjusting the speed
        }
        else if(Dist0<300 && !((Dist-Dist0)<300))
        {
            robot->setVel(Dist0/300*Speed+5);
        }
        else
        {
            robot->setVel(Speed);
        }
        Dist0+=robot->getVel()* dt/1000;//current travelled distance
        printf("D=%f\tS=%f\r",Dist0,robot->getVel());
        robot->unlock();
        ArUtil::sleep(dt);
    }
    robot->lock();
    robot->setVel(0);
    robot->unlock();
    return 1;
}
```

1.2 Rotation

The rotation task (named ”Turn”) is quite similar to ”Go”. Instead of linear speed and distance, we use rotation speed and angle as inputs, giving the following prototype:

int Turn(ArRobot *robot,double Speedr,double Angle)

Then we use exactly the same approach as in the previous task, with the use of setRotVel(Speedr) to command the robot’s spinning speed. Again we had to adjust the speed, this time in terms of the current rotation angle. The final code of this function is shown below:

```
int Turn(ArRobot *robot,double Speedr,double Angle)//the robot turns at the specified angle with
    the specified speed
{
    printf("Turn : %f deg at Speed : %f deg/s\n",Angle,Speedr);
    double Angle0=0;
    double dt=200;
    double direction=0;
    while(Angle0<Angle)
    {
        robot->lock();
        robot->enableMotors();
        if((Angle-Angle0)<45)//adjusting rotational speed
        {
            if(Speedr<0)
                robot->setRotVel((Angle-Angle0)/45*Speedr-2);
            else
                robot->setRotVel((Angle-Angle0)/45*Speedr+2);
        }
        else if(Angle0<30 && !((Angle-Angle0)<45))
        {
            if(Speedr<0)
                robot->setRotVel(Angle0/45*Speedr-2);
            else
                robot->setRotVel(Angle0/45*Speedr+2);
        }
        else
        {
            robot->setRotVel(Speedr);
        }
        if(robot->getRotVel()<0)
            Angle0+=-robot->getRotVel()* dt/1000;
        else
            Angle0+=robot->getRotVel()* dt/1000;
        printf("A=%f\tSr=%f\r",Angle0,robot->getRotVel());
        robot->unlock();
        ArUtil::sleep(dt);
    }
    robot->lock();
    robot->setRotVel(0);
    robot->unlock();
    return 1;
}
```

1.3 Move to point

Now that we have our two basic movement tasks, we can combine them to make the robot to move at any particular relative position. This is the purpose of the "GoTo" function. Its prototype is:

int GoTo(ArRobot *robot,double x, double y)

with x and y the goal's coordinates relative to the robot's initial position.

The code of this function is composed of two phases: first we calculate the distance between the robot and the destination, along with the angle between the robot's initial facing direction and the virtual line between the robot and the goal. Then we use the results of these calculations as inputs parameters for Go and Turn functions. You can see the final code below:

```
int GoTo(ArRobot *robot,double x, double y)//combination of the 2 previous functions
{
    //printf("Posteta = %f ",Posteta);
    //system("pause");
    double Angle;
    double Dist;
    if((x-Posx)>0 && (y-Posy)>=0)//determinig the angle between robot initial direction and goal
        Angle = atan((y-Posy)/(x-Posx));
    else if((x-Posx)<=0 && (y-Posy)>0)
        Angle = atan((y-Posy)/(x-Posx))+ 3.14159;
    else if((x-Posx)<0 && (y-Posy)<=0)
        Angle = atan((y-Posy)/(x-Posx))+ 3.14159;
    else if((x-Posx)>=0 && (y-Posy)<0)
        Angle = atan((y-Posy)/(x-Posx))+ 2*3.14159;

    double SaveAngle;
    SaveAngle=Angle;
    Angle-=Posteta;
    Posteta=SaveAngle;

    Dist = sqrt((x-Posx)*(x-Posx)+(y-Posy)*(y-Posy));//calcul of the distance to reach the goal
    point
    if(Angle>3.14159)//turn to right side
        Turn(robot,-20,360-Angle/3.14159*180);
    else if(Angle<0 && Angle>-3.14159)
        Turn(robot,-20,2*3.14159-Angle/3.14159*180);
    else if(Angle<0)
        Turn(robot,20,-Angle/3.14159*180);
    else
        Turn(robot,20,Angle/3.14159*180);

    Go(robot,250,Dist);//go forward to the goal with a speed of 250mm/s
    Posx=x;//save the new robot's position
    Posy=y;
    return 1;
}
```

Chapter 2

Mapping

The mapping program uses the sonars to detect objects and draw a map of the room. The robot move autonomously in the room and do the entire job.

First we have to direct the robot and avoid obstacles. To do that, when the sonars detect an obstacle in front of the robot, the robot turns on the side where there is no obstacle.

```
if(f<=MinDist)
        {
            printf("Turn\r");
            robot->lock();
            robot->setVel(0);
            double direction=0;
            double Speedr=20;
            if(l<r)
                direction=1;
            else
                direction=-1;
            Speedr*=direction;
            robot->setRotVel(Speedr);
            robot->unlock();
        }
        else
        {
            printf("Go\r");
            robot->lock();
            robot->setVel(200);
            robot->setRotVel(0);
            robot->unlock();
        }
```

To remember all obstacles, we use a 2D array (generated dynamically). Then a background task scans the room with the sonars and put the coordinates of each obstacle in this array. To do that we need to know the position of the robot, the distance and the angle of the obstacle from the robot. With a trigonometric calculus we can determinate the position of the obstacle in the array.

```
double th;
double d;

    d = sonar.currentReadingPolar(-45, 45, &th); // get the range of the nearest object at the front of the robot
    f=d;
    if(d<5000)
    {
        //system("cls");
        double Wallx=robot->getX(); // Calculate the position of this object and convert it to put it in the grid
        double Wally=robot->getY();
        Wallx+= d*cos((th+robot->getTh())*PI/180); // Calcul to place the object on the map
        Wally+= d*sin((th+robot->getTh())*PI/180);
        Wallx/=res; // Put the value to an appropriate size
        Wally/=res;
        //printf("%d %d\n",(int)Wallx,(int)Wally);
        if( ((int)Wallx+Wide/2)<Wide && ((int)Wally+Height/2)<Height  && ((int)Wallx+Wide/2)>-1 &&
     ((int)Wally+Height/2)>-1) // Be sure to not be out of the grid
            Grille[(int)Wallx+Wide/2][(int)Wally+Height/2]=1; // Place the object on the grid
    }
```

The starting position of the robot is the center of the array, that's why we have some initialization variables at the beginning.

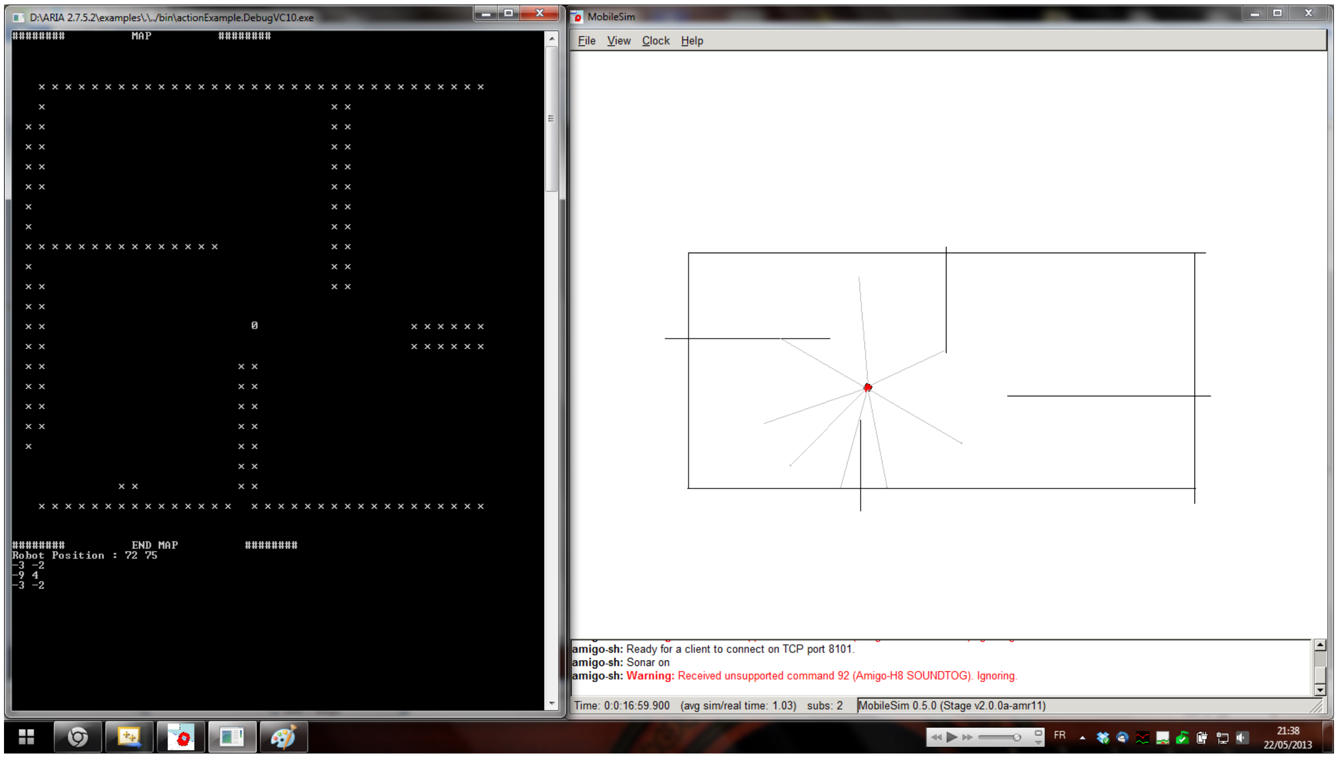
```
double Posx=0; // Position x of the robot
double Posy=0; // Position y of the robot
double Posteta=0; // Angle of the robot
double MinDist=500; // Minimum distance to turn
double f,l,r; // Distance at Left Right and Front of the robot (from sonar).

int Wide=150; // Number of cells of the map in the console
int Height=150; // Idem
int res=500; // Resolution 1000 => 1m
int** Grille=NULL; // Grid for mapping

int ConsoleWide=36;// Number of cells to show in console
int ConsoleHeight=25;

int PositionMapX=Wide/2-ConsoleWide/2;// Starting position in console
int PositionMapY=Height/2-ConsoleHeight/2;
```

By using the arrow key we can move the view in the console.

```
if (GetAsyncKeyState(VK_UP))// Use directional key to navigate in the console map
        {
            PositionMapY++;
            ArUtil::sleep(100);
        }
        if (GetAsyncKeyState(VK_DOWN))
        {
            PositionMapY--;
            ArUtil::sleep(100);
        }
        if (GetAsyncKeyState(VK_LEFT))
        {
            PositionMapX--;
            ArUtil::sleep(100);
        }
        if (GetAsyncKeyState(VK_RIGHT))
        {
            PositionMapX++;
            ArUtil::sleep(100);
        }
```

There is a drawing function, it draw the map of the room in the console using the array. Walls are represented with "X", the robot by "0". The prototype is the following : int DisplayMap(int** Grille, int Wide, int Height);

This function is called by a background task.


D:\ARIA 2.7.5.2\examples\.\../bin\actionExample.DebugVC10.exe

```
########          MAP          ########


    x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x
    x                                           x x
  x x                                           x x
  x x                                           x x
  x x                                           x x
  x x                                           x x
  x                                             x x
  x                                             x x
  x x x x x x x x x x x x x x x                 x x
  x                                             x x
  x x                                           x x
  x x
  x x                               0                       x x x x x x
  x x                                                       x x x x x x
  x x                             x x
  x x                             x x
  x x                             x x
  x x                             x x
  x                               x x
                                  x x
                x x               x x
    x x x x x x x x x x x x x x x   x x x x x x x x x x x x x x x x x x


########          END MAP          ########
Robot Position : 72 75
-3 -2
-9 4
-3 -2
```

MobileSim
File
View
Clock
Help

```
amigo-sh: Ready for a client to connect on TCP port 8101.
amigo-sh: Sonar on
amigo-sh: Warning: Received unsupported command 92 (Amigo-H8 SOUNDTOG). Ignoring.
Time: 0:0:16:59.900   (avg sim/real time: 1.03)   subs: 2   MobileSim 0.5.0 (Stage v2.0.0a-amr11)
```

Example of simulation

This program is very helpful to understand how to use console and sonars. It also provides an example of using a background task.

```cpp
class Mapping // Callback of the mapping display
{
public:
    // Constructor. Adds our 'user task' to the given robot object.
    Mapping(ArRobot *robot);

    // Destructor. Does nothing.
    ~Mapping(void) {}

    // This method will be called by the callback functor
    void doTask(void);
protected:
    ArRobot *myRobot;

    // The functor to add to the robot for our 'user task'.
    ArFunctorC<Mapping> myTaskCB;
};
// the constructor
Mapping::Mapping(ArRobot *robot):
    myTaskCB(this, &Mapping::doTask)
{
    myRobot = robot;
    // just add it to the robot
    myRobot->addSensorInterpTask("PrintingTask", 50, &myTaskCB);
}

void Mapping::doTask(void)
{
    // print out some info about the robot and the map
    system("cls");
    DisplayMap(Grille,Wide,Height);

    fflush(stdout);
}
```

And in main function :

Mapping pt(robot); // Start Display mapping callback function

The simulation is working perfectly according to the precision of the sonars, but in reality it's not working as well. Sonars's precision is not so good, it result a wrong position of obstacles on the map. Furthermore, the odometry is not exact it also imply a shift in obstacles positions. Maybe it will be better to use the laser scanner in the future.

Chapter 3

Smooth Trajectory

3.1 Visual Studio C++

The smooth trajectory algorithm is use to move the robot from a point to another by doing a ”smooth trajectory”, avoiding turning while not going forward. We need to pass the coordinates of the destination (x,y,teta) to make it work.

The algorithm is based on the following calculus:

Cartesian to Polar coordinates:

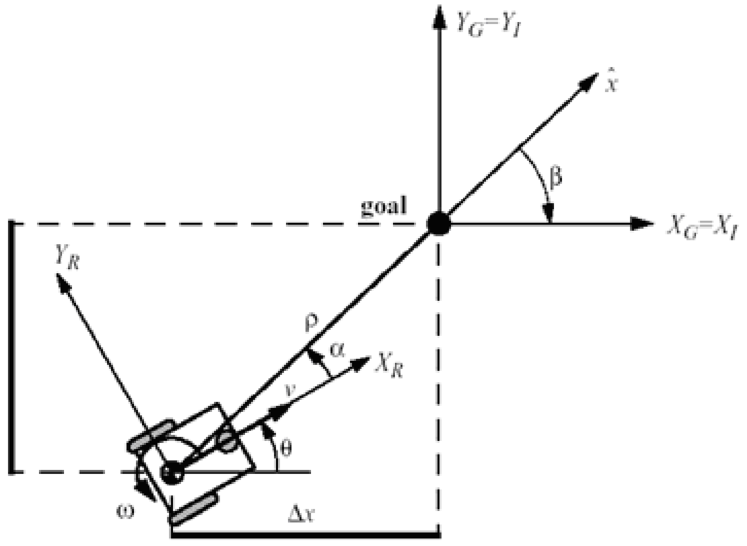
$$\rho = \sqrt{\Delta x^2 + \Delta y^2} \tag{3.1}$$

$$\alpha = -\theta + \arctan^2(\Delta x, \Delta y) \tag{3.2}$$

$$\beta = -\theta - \alpha \tag{3.3}$$

Speed and speed rotation:

$$v = k_\rho \rho \times \omega = k_\alpha \alpha + k_\beta \beta \tag{3.4}$$



The k coefficients depend on the robot size, speed etc. and must be changed according to the objective of the program. In the C++ program the speed and the speed rotation of the robot is calculated continuously and provide to the robot the appropriate trajectory. When the robot reaches his goal it just stops. With some adjustment, the robot can do the same thing but going backward (see the other source file). However this program has some constraints, the goal point must be in front of the robot and the orientation must be from minus pi/2 to pi/2 relatively to the starting orientation.

The algorithm is divided in 3 parts:

1. Initialization
2. Calculus
3. Commands

Note that 2 and 3 are in the same loop. It calculate the new speed and apply it to the robot.

During the initialization we provide to the program the coefficients k1, k2 and k3. K1 control the speed of the robot. K2 and k3 the rotation speed. To have sharp turns we must change k2 and k3 and increase their values. We also need to enter the destination point relatively to the starting position of the robot.

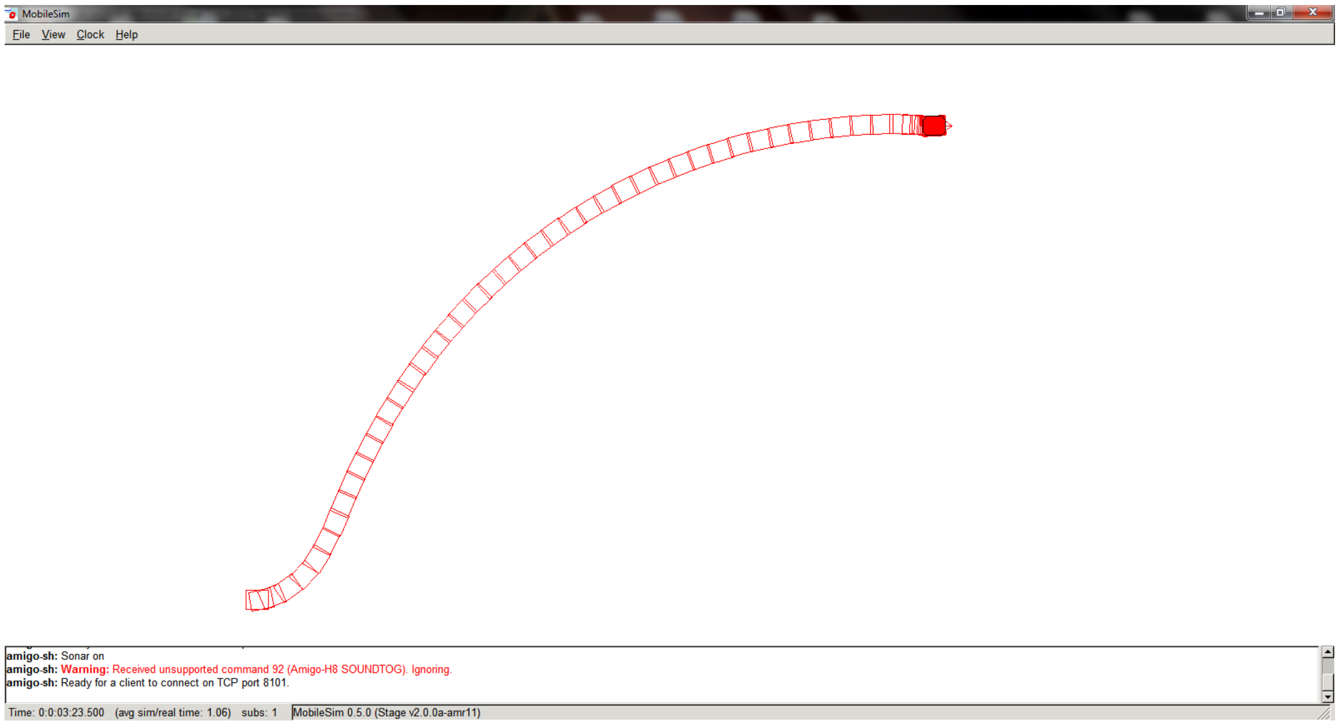
```
k1=0.6;   // Define coefficients  k1>0  k2-k1>0  k3<0
k2=50;
k3=-50;

xg=1000;  // Destination point
yg=-1000;
tg=-PI/2;
```

During the calculus it get the new coordinates of the robot in polar and calculate the new speed and speed rotation according to k1, k2 and k3.

Then it sends the command to the robot.

```
for(;;)
{
    ArUtil::sleep(100);
    x0 = robot->getX();
    y0 = robot->getY();
    t0 = robot->getTh()*PI/180 + tg;

    ro = sqrt((xg-x0)*(xg-x0)+(yg-y0)*(yg-y0));   // Calculus according to robot position
    alpha = -t0 + atan2((yg-y0),(xg-x0));
    beta = -t0 - alpha;

    v=k1*ro;
    if(v>150)// Limit speed and rotation speed
        v=150;
    w=k2*alpha+k3*beta;
    if(w>45)
        w=45;
    if(w<-45)
        w=-45;

    robot->lock();
    robot->setVel(v);
    robot->setRotVel(w);
    robot->unlock();

    if( abs(robot->getX()-xg)<2 && abs(robot->getY()-yg)<2 && abs(robot->getTh()-tg*180/PI)<2)
  // Stop the robot if it reach his destination point
    {
        break;
    }
}
```

Results are quite good, even in the reality the precision of movements are respected.



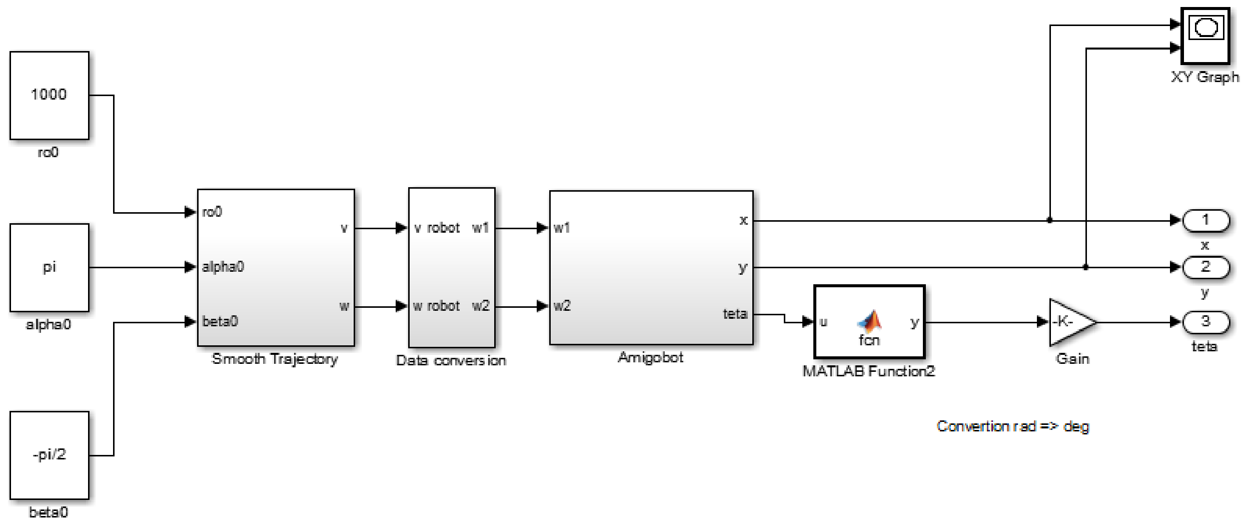
```
amigo-sh: Sonar on
amigo-sh: Warning: Received unsupported command 92 (Amigo-H8 SOUNDTOG). Ignoring.
amigo-sh: Ready for a client to connect on TCP port 8101.
Time: 0:0:03:23.500   (avg sim/real time: 1.06)   subs: 1   MobileSim 0.5.0 (Stage v2.0.0a-amr11)
```

Simulation of the algorithm with MobileSim.

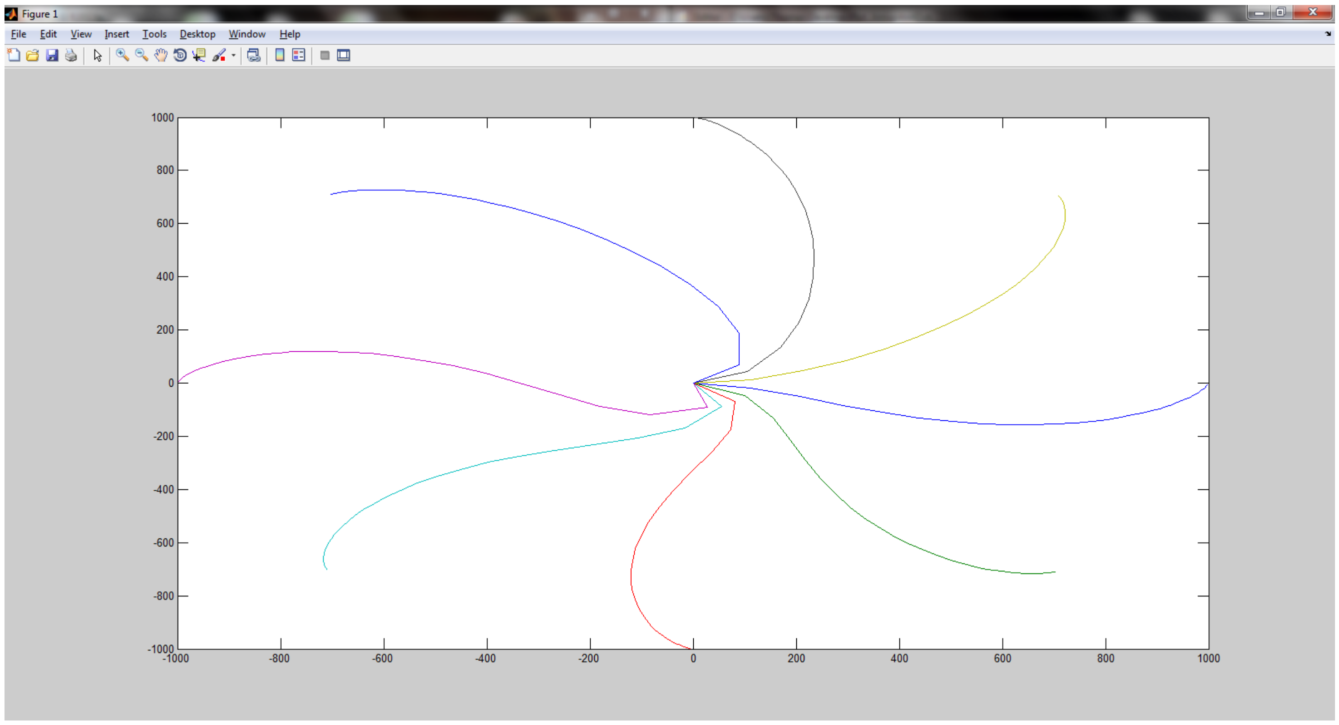
3.2 Matlab and Simulink

Simulink is a software in which we can create and simulate dynamic system, such as the Amigobot. We use the same algorithm into Matlab with Simulink. We had to adapt the algorithm to make it work with blocs.

The script is composed by the smooth trajectory bloc, and the simulation bloc. The first one calculates the speed of the robot. The second one applies it to the model of the robot and provides the new position.



We put different destination goals with the smooth trajectory algorithm. Here is the result:



Simulink allow us to simulate the robot without MobileSim and using the power of Matlab. We also compile the ARIA library for Matlab and control the robot via Matlab. Using ARIA with Matlab is not so different from Visual Studio C++, see the following source code to understand how to use it.

```
arrobot_disconnect
clear all
% initialize with default arguments: (connect to robot connected to this
% computer via COM1 serial port)
aria_init

% initialize aria to connect to a remote computer (e.g. a robot with a wifi
% interface instead of onboard computer, or a simulator running on another
% computer):
%aria_init -rh 192.168.1.13


% connect to the robot:
arrobot_connect
disp 'use arrobot_stop to stop'

% make the robot drive in a small circle:


%x0=input('x0 ?');
%y0=input('y0 ?');
%t0=input('t0 ?');
x0=0;
y0=0;
t0=0;

xg=input('xg ?');
yg=input('yg ?');
%tg=input('tg ?');
tg=0;

k=[0.6 50 -20]'

k0=[x0 y0 t0]'
kg=[xg yg tg]'

ro = sqrt((xg-x0)^2+(yg-y0)^2)
```

```
alpha = -t0 + atan2((yg-y0),(xg-x0))
beta = -t0 - alpha

coeff=[ro alpha beta]'

v=k(1)*coeff(1)
w=k(2)*coeff(2)+k(3)*coeff(3)



while arrobot_getx ~= xg || arrobot_gety ~= yg % CTRL + C to get out of the loop
    x0 = arrobot_getx;
    y0 = arrobot_gety;
    t0 = arrobot_getth*pi/180;

    ro = sqrt((xg-x0)^2+(yg-y0)^2);
    alpha = -t0 + atan2((yg-y0),(xg-x0));
    beta = -t0 - alpha;

    coeff=[ro alpha beta]';

    v=k(1)*coeff(1);
    w=k(2)*coeff(2)+k(3)*coeff(3);

    if v>500
        v=500;
    end
    if w>45
        w=45;
    end
    if w<-45
        w=-45;
    end
    arrobot_setvel(v);
    arrobot_setrotvel(w);
    pause(0.001);
    clc
    disp(['X = ' num2str(arrobot_getx) ' Y = ' num2str(arrobot_gety) ' Speed = ' num2str(v) ' Rot
    Speed = ' num2str(w)])
end
arrobot_setvel(0);
arrobot_setrotvel(0);
```

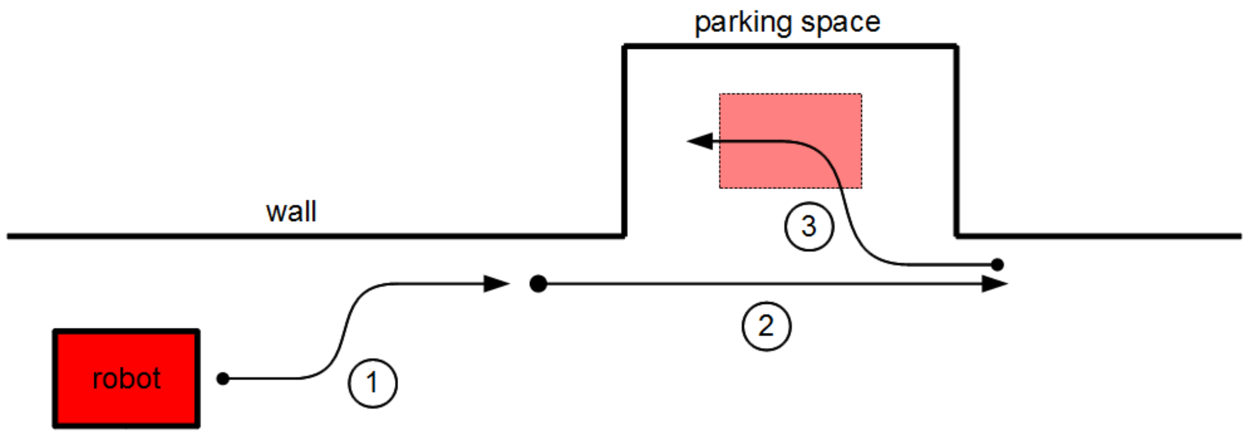
Using this program results will be the same.

Chapter 4

Parking

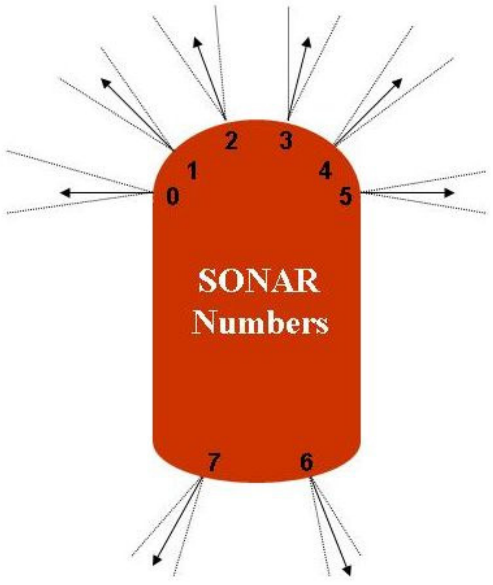
Parking:

The aim of this program is to make the robot to park in a place using its sonar to detect its environment. The full process is described in the picture below:

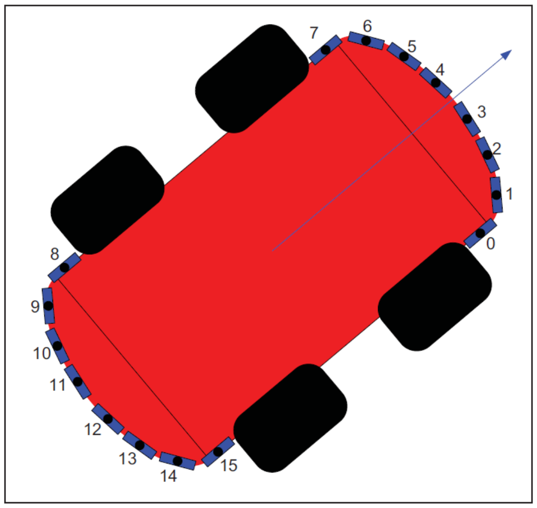


1. the robot measures the distance between itself and the wall, and then get closer to it if necessary
2. the robot move forward until it detects the end of the parking space
3. then the robot moves to the center of the place

So the first thing we need in our program is to obtain the values from the different sonar around the robot. The Amigobot and the Pioneer 3-AT have two different sonar configurations (shown in the pictures below), but in the two cases we are interested in the values from left and right sonar. So it means sonar number 0 and 5 for the Amigobot, and sonar number 0 and 7 for the P3AT.



Amigobot



Pioneer 3-AT

That's why we created a function named "GetSonar", which record the values from the sonar in the array "ListSonar". It uses the integrated function getSonarRange(number) to obtain the value from the given sonar. The corresponding code is visible below:

```
void GetSonar(void)
{
    for(int i=0; i<8; i++)
    {
        ListSonar[i] = robot->getSonarRange(i);//make an array of the values from the 8 sonars
        //printf("%d\n",ListSonar[i]);
    }
}
```

The next function is sonarPrinter(), which saves the ranges of the nearest obstacles in front, back, left and right of the robot respectively in f, b, l and r parameters. This task uses currentReadingPolar to obtain the sonar value in the given range in degrees.

```
void sonarPrinter(void) // Get the range at the front left right and back of the robot (and
    mapping)
{
    double th;
    double d;

    d = sonar.currentReadingPolar(-45, 45, &th); // get the range of the nearest object at the
    front of the robot
    f=d;

    d = sonar.currentReadingPolar(-135, -45, &th); // Idem with left ...
    l=d;

    d = sonar.currentReadingPolar(45, 135, &th); // Idem with right
    r=d;

    d = sonar.currentReadingPolar(-135, 135, &th); // Idem with back
    b=d;
    GetSonar();//call previous function
    fflush(stdout);
}
```

Then we have the Alignment function. It have to input parameters: the side where the wall is (left or right), and the current distance between the wall and the robot. If this distance is too far from the constant MINDIST, then the robot moves to the right distance using a smooth trajectory.

```
void Alignement(int Side, double Distance)//robot goes to the specified distance from the wall if necessary
{
    if(Distance < 0.9 * MINDIST  || Distance > 1.1 * MINDIST)//if the robot is not at the right distance from the wall
    {
        double x0,y0,t0,xg,yg,tg,k1,k2,k3,ro,alpha,beta,v,w;
        printf("\n\nWaiting for good alignment...\n");
        k1=0.6;
        k2=150;
        k3=-50;

        xg=robot->getX()+ AMIGOBOTLENGTH * cos(robot->getTh()*PI/180)+ (MINDIST-Distance) * sin(robot->getTh()*PI/180) ;
        yg=robot->getY()+ AMIGOBOTLENGTH * sin(robot->getTh()*PI/180) - (MINDIST-Distance) * cos(robot->getTh()*PI/180) + Side * 2 * (MINDIST-Distance) * cos(robot->getTh()*PI/180) ;;
        tg=robot->getTh()*PI/180;

        for(;;)
        {
            ArUtil::sleep(100);
            x0 = robot->getX();
            y0 = robot->getY();
            t0 = robot->getTh()*PI/180 + tg;

            ro = sqrt((xg-x0)*(xg-x0)+(yg-y0)*(yg-y0));
            alpha = -t0 + atan2((yg-y0),(xg-x0));
            beta = -t0 - alpha;

            v=k1*ro;
            if(v>150)
                v=150;
            w=k2*alpha+k3*beta;
            if(w>45)
                w=45;
            if(w<-45)
                w=-45;

            robot->lock();
            robot->setVel(v);
            robot->setRotVel(w);
            robot->unlock();

            printf("Pose=(%.2f,%.2f,%.2f) Goal=(%.2f,%.2f,%.2f)\r",robot->getX(), robot->getY(), robot->getTh(),xg,yg,tg*180/PI);


            if( abs(robot->getX()-xg)==0 && abs(robot->getY()-yg)==0 && abs(robot->getTh()-tg*180/PI)<2)
            {
                printf("\nRobot aligned !\n");
                break;
            }

            if (GetAsyncKeyState(VK_ESCAPE)) // To compile go to : Project => Property => Linker => Input => Additional Dependencies => Add kernel32.lib user32.lib gdi32.lib winspool.lib comdlg32.lib
            {
                break; // Now we are able to end the programme by pressing ESCAPE
            }
        }
    }
}
```

Now that the robot is correctly aligned, it can start the parking space detection. If it doesn't find a place to park before it travelled a certain distance (5 meters in our example), or if the place is too small then the process aborts.

To detects and measures the parking space, the robot moves forward in parallel to it. When the robot detect a change in its distance from the wall (corresponding to the beginning of the place), it checks if the depth of the space is large enough. If it's correct then the program records the ParkingSizeX and ParkingSizeY values according to the relative position of the robot (given by getX() and getY() integrated functions). When the robot detects the end of the place, the program records the new values of robot's coordinates, and then calculates the distance between those two points to obtain the space's width. If this value is large enough (which means superior to 1,3 times the robot length), then the function returns 0, otherwise 1. The code of this function, named "Evaluation", is available below:

```
int Evaluation(int Side,double Distance,double * ParkingSize)//measures the place's geometry
{
    double x0,y0,t0,xg,yg,tg,k1,k2,k3,ro,alpha,beta,v,w;
    double ParkingSizeX = 0;
    double ParkingSizeY = 0;
    int Evaluation=0;
    x0 = robot->getX();
    y0 = robot->getY();
    t0 = robot->getTh()*PI/180;

    while( sqrt(abs(x0-robot->getX())*abs(x0-robot->getX()) + abs(y0-robot->getY())*abs(y0-robot->getY()))< 5000)//keep going until the robot end the evaluation, or travelled for 5 meters
    {

        robot->lock();
        robot->setVel(200);
        robot->setRotVel(0);
        robot->unlock();
        if( ((Side==0 && ListSonar[0]>(Distance+MINIMUMPARKINGWIDTH)) || (Side==1 && ListSonar[5]>(Distance+MINIMUMPARKINGWIDTH))) && Evaluation==0)//detecting the beginning of the place
        {
            Evaluation = 1;//evaluation starts
            ParkingSizeX = robot->getX();
            ParkingSizeY = robot->getY();
            printf("Evaluation of the parking size...\n");
        }

        if(Evaluation==1)
        {
            printf("\rParking Size : %.0f",sqrt(abs(ParkingSizeX - robot->getX())*abs(ParkingSizeX - robot->getX())+abs(ParkingSizeY - robot->getY())*abs(ParkingSizeY - robot->getY())));
        }

        if( ((Side==0 && ListSonar[0]<=(Distance+MINIMUMPARKINGWIDTH)) || (Side==1 && ListSonar[5]<=(Distance+MINIMUMPARKINGWIDTH))) && Evaluation==1)//detecting the place's end
        {
            printf("\nEvaluation terminated\n");
            Evaluation = 0;
            ParkingSizeX = abs(ParkingSizeX - robot->getX());
            ParkingSizeY = abs(ParkingSizeY - robot->getY());
            *ParkingSize = sqrt(ParkingSizeX*ParkingSizeX+ParkingSizeY*ParkingSizeY);//calcul of the place's size according to robot measures
            if(*ParkingSize<AMIGOBOTLENGTH*1.3)//if the place is large enough
            {
                return 0;

            }
            else
            {
                return 1;
            }
        }


        if (GetAsyncKeyState(VK_ESCAPE)) // To compile go to : Project => Property => Linker => Input => Additional Dependencies => Add kernel32.lib user32.lib gdi32.lib winspool.lib comdlg32.lib
        {
            return 0;
        }
    }
}
```

If the evaluation task passed successfully, the robot cans now parks in the place. This is the purpose of the "Parking" function, which is quite similar to "Alignment", because it makes the robot moving to the desired position using a smooth trajectory again. It inputs parameters are the place's size, the side where to park and the distance from the place.

```
void Parking(double PlaceSize,int Side,double Distance)//park the robot according to the place's
    size and its position
{
    printf("Parking...\n");
    double x0,y0,t0,xg,yg,tg,k1,k2,k3,ro,alpha,beta,v,w,InitAngle,GoalAngle;

    //robot goes backward according to coordinates below:
    xg = robot->getX() - AMIGOBOTLENGTH/4*cos(robot->getTh()*PI/180)/2;
    yg = robot->getY() - AMIGOBOTLENGTH/4*sin(robot->getTh()*PI/180)/2;
    tg = robot->getTh()*PI/180;

    k1=0.3;
    k2=200;
    k3=-100;

    for(;;)
    {
        ArUtil::sleep(100);
        x0 = robot->getX();
        y0 = robot->getY();
        t0 = robot->getTh()*PI/180 + tg;

        ro = sqrt((-xg+x0)*(-xg+x0)+(-yg+y0)*(-yg+y0));
        alpha = -t0 + atan2((-yg+y0),(-xg+x0));
        beta = -t0 - alpha;

        v=-k1*ro;
        if(v>150)
            v=150;
        w=k2*alpha+k3*beta;
        if(w>45)
            w=45;
        if(w<-45)
            w=-45;

        robot->lock();
        robot->setVel(v);
        robot->setRotVel(w);
        robot->unlock();

        //system("cls");
        printf("Pose=(%.2f,%.2f,%.2f) Goal=(%.2f,%.2f,%.2f)\r",robot->getX(), robot->getY(), robot
    ->getTh(),xg,yg,tg*180/PI);

        if(b<200)
        {
            k1 = 0.1;
        }

        if( abs(robot->getX()-xg)<2 && abs(robot->getY()-yg)<2 && abs(robot->getTh()-tg*180/PI)
    <10)
        {
            break;
        }

        if (GetAsyncKeyState(VK_ESCAPE)) // To complie go to : Project => Property => Linker =>
    Input => Additional Dependencies => Add kernel32.lib user32.lib gdi32.lib winspool.lib
    comdlg32.lib
        {
            break; // Now we are able to end the programme by pressing ESCAPE
        }
    }

    //robot is parking on the right side according to place's size
    xg = robot->getX() - PlaceSize*cos(robot->getTh()*PI/180)/2 + (Distance+AMIGOBOTWIDTH/2)*sin(
    robot->getTh()*PI/180) ;
    yg = robot->getY() - PlaceSize*sin(robot->getTh()*PI/180)/2 + (Distance+AMIGOBOTWIDTH/2)*cos(
    robot->getTh()*PI/180) + Side*-2*(Distance+1.5*AMIGOBOTWIDTH)*cos(robot->getTh()*PI/180) ;
    tg = robot->getTh()*PI/180;

    k1=0.3;
    k2=200;
    k3=-100;

    for(;;)
    {
        ArUtil::sleep(100);
        x0 = robot->getX();
        y0 = robot->getY();
        t0 = robot->getTh()*PI/180 + tg;

        ro = sqrt((-xg+x0)*(-xg+x0)+(-yg+y0)*(-yg+y0));
        alpha = -t0 + atan2((-yg+y0),(-xg+x0));
        beta = -t0 - alpha;

        v=-k1*ro;
        if(v>150)
            v=150;
        w=k2*alpha+k3*beta;
        if(w>45)
            w=45;
        if(w<-45)
            w=-45;

        robot->lock();
        robot->setVel(v);
        robot->setRotVel(w);
        robot->unlock();

        //system("cls");
        printf("Pose=(%.2f,%.2f,%.2f) Goal=(%.2f,%.2f,%.2f)\r",robot->getX(), robot->getY(), robot
    ->getTh(),xg,yg,tg*180/PI);

        if(b<200)
        {
            k1 = 0.1;
        }

        if( abs(robot->getX()-xg)<2 && abs(robot->getY()-yg)<2 && abs(robot->getTh()-tg*180/PI)
    <10)
        {
            printf("\nParked !\n");
            break;
        }

        if (GetAsyncKeyState(VK_ESCAPE)) // To complie go to : Project => Property => Linker =>
    Input => Additional Dependencies => Add kernel32.lib user32.lib gdi32.lib winspool.lib
    comdlg32.lib
        {
            break; // Now we are able to end the programme by pressing ESCAPE
        }
    }

    robot->lock();
    robot->setVel(0);
    robot->setRotVel(0);
    robot->unlock();
}
```

Now we can called all these functions inside the main part of the program in order to execute the complete parking process. Firstly we compare the left and right sonar's values to determine if the parking space is located at the left or right of the robot, by using these lines of code:

```
if(ListSonar[0]>ListSonar[5]) // Parking place on right or left side ?
    {
        Side=1;
        Distance = ListSonar[5];//distance to right side of the robot
        printf("\nParking right side...\n\n");
    }
    else
    {
        Side=0;
        Distance = ListSonar[0];//distance to left side of the robot
        printf("\nParking left side...\n\n");
    }
```

Then we called in the order the Alignment, Evaluation and Parking functions to complete the process.

Chapter 5

iPhone control

The iPhone program allows us to control the P3-AT and its camera using the accelerometer of an iDevice. For that you need to have one your phone the application TouchOSC. Then you will be able to control the P3-AT if you upload the right remote and configure the application correctly.

First you need to connect to the wifi of the robot with your iDevice, and then you have to enter the IP address of it (192.168.1.10). Put the same port in the application and in the source code or else it won't work.

```
const int PORT_NUM = 9109; // INCOMING PORT
const int PORT_NUM_OUT = 9000; //OUTCOMING PORT
#define IP_OUT "192.168.1.3" // iDevice IP
```

In the first applet you will find:

- STOP: Stop the robot (but not the program).
- EXIT: Stop the program.
- ACC: Use the accelerometer instead of the trackpad.

In the second applet you will find:

- A fader to setup the zoom.
- A trackpad to control the camera.

This application use the OSC protocol to send data between the robot and the iphone. To dialog with the iPhone, the robot use the oscpkt library (very simple).

First we need to connect to the iPhone (one connection for incoming packets and the other for out coming packets).

```
UdpSocket sock;// Creat two OSC connections, one for send and the other to receive
sock.bindTo(PORT_NUM);
UdpSocket sock2;
sock2.connectTo(IP_OUT, PORT_NUM_OUT);
```

Then we have to catch data and store them into variables.

```
        if (!sock.isOk())
        {
            cerr << "Error opening port " << PORT_NUM << ": " << sock.errorMessage() << "\n";
        }
        else
        {
            PacketReader pr;
            PacketWriter pw;
            if (sock.receiveNextPacket(30 /* timeout, in ms */))
            {
                pr.init(sock.packetData(), sock.packetSize());
                oscpkt::Message *msg;
                if(pr.isOk() && (msg = pr.popMessage()) != 0)//Receive qll data from iDevice
                {
                    if (msg->match("/accxyz"))// Accelerometer (x,z,orietation)
                    {
                        msg->match("/accxyz").popFloat(x).popFloat(z).popFloat(orientation);//Save
    data into public variables
                        //cout << "Acc :" << x << "\t" << "\t" << z << "\t" << orientation /* << "
    from " << sock.packetOrigin()*/ << "\n";
                    }
                    else if (msg->match("/1/stop"))// Stop button
                    {
                        msg->match("/1/stop").popFloat(stop);
                        cout << "Stop :" << stop << "\n";
                    }
                    else if (msg->match("/1/acc"))// Enable accelerometer button
                    {
                        msg->match("/1/acc").popFloat(acc);
                        cout << "Acc :" << acc << "\n";
                    }
                    else if (msg->match("/1/control/1"))//  Touchpad
                    {
                        msg->match("/1/control/1").popFloat(forward).popFloat(direction);
                        cout << "Control :" << forward << " " << direction << "\n";
                    }
                    else if (msg->match("/2/multixy1/1"))// Camera Touchpad
                    {
                        msg->match("/2/multixy1/1").popFloat(tilt).popFloat(pan);
                        cout << "Camera :" << tilt << " " << pan << "\n";
                    }
                    else if (msg->match("/2/fader1"))// Camera zoom
                    {
                        msg->match("/2/fader1").popFloat(zoom);
                        cout << "Zoom :" << zoom << "\n";
                    }
                    else if (msg->match("/1/exit").popFloat(exit))// Exit button
                    {
                        if(exit==1)
                            break;
                    }
                    else// Unhandled messages
                    {
                        cout << "Server: unhandled message: " << *msg << "\n";
                    }
                }
            }
        }
```

Each button or other control is represented by an address (ex : "/2/fader1") We have to check if the incoming packets correspond to this address, and then store its value.

To control the robot we just send a commend proportional to the tilt of the iPhone or the position in the trackpad (with a threshold).

```
robot.lock();
        if(acc)// Command the robot with accelerometer
        {
            if(fabs(x)>THRESHOLD)
                Speed = double(x*MAX_SPEED);
            else
                Speed=0;

            if(fabs(z)>THRESHOLD)
                SpeedR = double(z*MAX_ROT_SPEED);
            else
                SpeedR=0;
        }
        else// Command with touchpad
        {
            if(fabs(forward)>THRESHOLD_PAD)
                Speed=double(forward*MAX_SPEED);
            else
                Speed=0;

            if(fabs(direction)>THRESHOLD_PAD)
                SpeedR=double(-direction*MAX_ROT_SPEED);
            else
                SpeedR=0;
        }

        if(stop==1)// Stop the robot if button "STOP" is enable
        {
            SpeedR=0;
            Speed=0;
        }

        robot.setVel(Speed);// Set speed and speedr according to previous command
        robot.setRotVel(SpeedR);
        robot.unlock();
```

The command of the camera is more complicated. We pick an example of controlling it and adapt it to this program. You can see a lot of objects to control it, but the important is:

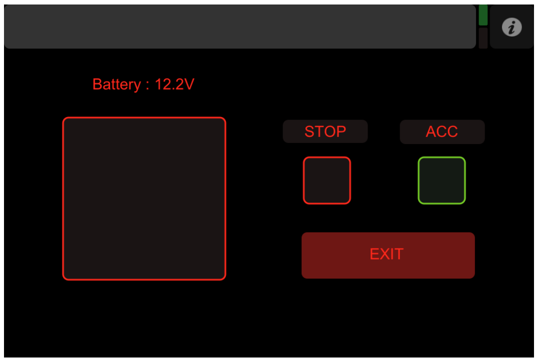
```
TiltAngle=int((1-tilt)*myPTU.getMaxPosTilt());//Convert data
PanAngle=int((-pan*2+1)*myPTU.getMaxPosPan());
Zoom=int((zoom-0.001)*myPTU.getMaxZoom());

myPTU.panTilt(PanAngle,TiltAngle);// Send commands to the camera
myPTU.zoom(Zoom);
```

We convert data to the right value and send it to the camera.
To send data to the iPhone we do that:

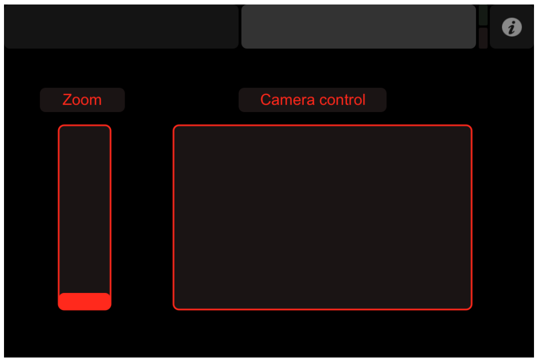
```
battery = robot.getBatteryVoltage();
        sprintf(btry_msg,"Battery : %.1fV",battery);
        if(strcmp(btry_msg,btry_msg_old)!=0)// Test if the battery level is different
        {
            strcpy(btry_msg_old,btry_msg);
            if (!sock2.isOk())
            {
                cerr << "Error connection to port " << PORT_NUM_OUT << ": " << sock.errorMessage()
     << "\n";
            }
            else
            {
                sprintf(btry_msg,"Battery : %.1fV",battery);// Send it to the iDevice
                cout << btry_msg << std::endl;
                Message msg("/1/label2");
                msg.pushStr(btry_msg);
                PacketWriter pw;
                pw.startBundle().startBundle().addMessage(msg).endBundle().endBundle();
                bool ok = sock2.sendPacket(pw.packetData(), pw.packetSize());
            }

        }
```

Here you can see that we send the voltage of the battery, for that we need to check if the voltage has change (with strcmp) and the we send the string chain to the iPhone at the address ”/1/label2”.

The iPhone’s interface is visible below:



first tab with direction commands and battery voltage



second tab with camera control

Chapter 6

Color tracking using Matlab

This part is not directly related with the mobile robots, because it simply uses a webcam and a computer. It allows you to track an object in the camera's field of view according to its main color. In our example we used a red sheet that we hold in front of the webcam.

In the Matlab program, we first set the camera settings such as its name and output resolution. Then we specify what color we want to track, according to its red, green and blue values (from 0 to 255). Now the program can detect the area on the video which contains pixels of the given color range. Then each frame of the video is processed to obtain a binary picture. In fact, all pixels with the right color appear in white, and everything else in black. With this new picture, the program can now draw a rectangle around the white area, and the coordinates of its center can then be used to determine in which direction the object is moving. If there is more than one area of interest the program will consider only the largest one.

```
a = imaqhwinfo;
[camera_name, camera_id, format] = getCameraInfo(a);


%Capture the video frames using the videoinput function
%You have to replace the resolution & your installed adaptor name.

%vid = videoinput(camera_name, camera_id, format);
%sprintf('%s, %d, %s',camera_name,camera_id,format)

vid = videoinput('winvideo', 1, 'YUY2_160x120');
height = 120;
width = 160;

%Set the properties of the video object
set(vid, 'FramesPerTrigger', Inf);
set(vid, 'ReturnedColorspace', 'rgb');
vid.FrameGrabInterval = 5;

%start the video aquisition here
start(vid);

x = [];
y = [];
target = 0;
prevX=0;
prevY=0;

%Set a loop that stop after 100 frames of aquisition
while(vid.FramesAcquired<=200)

    %Get the snapshot of the current frame
    data = getsnapshot(vid);

    %Now to track red objects in real time
    %we have to subtract the red component
    %from the grayscale image to extract the red components in the image.

    %diff_im = imsubtract(data(:,:,1), rgb2gray(data));

    %Split the original image into color bands.
    redBand = data(:,:,1);
    greenBand = data(:,:,2);
    blueBand = data(:,:,3);

    %Threshold each color band.

    %Lower Threshold
    redThreshold = 170;
    greenThreshold = 0;
    blueThreshold = 0;

    %Upper Threshold
    redThreshold2 = 255;
    greenThreshold2 = 120;
    blueThreshold2 = 120;

    redMask = (redBand > redThreshold);
    greenMask = (greenBand > greenThreshold);
    blueMask = (blueBand > blueThreshold);

    redMask2 = (redBand < redThreshold2);
    greenMask2 = (greenBand < greenThreshold2);
    blueMask2 = (blueBand < blueThreshold2);

    %Combine the masks to find where all 3 are "true."
    diff_im = redMask & greenMask & blueMask & redMask2 & greenMask2 & blueMask2;

    %Use a median filter to filter out noise
    diff_im = medfilt2(diff_im, [4 4]);

    %Remove all those pixels less than 300px
    diff_im = bwareaopen(diff_im,300);

    %Label all the connected components in the image.
    bw = bwlabel(diff_im, 8);

    %Here we do the image blob analysis.
    %We get a set of properties for each labeled region.
    stats = regionprops(bw, 'BoundingBox', 'Centroid');

    % Display the image

    subplot(2, 2, 2);
    imshow(diff_im);

    set(text(10,10,strcat(' Number of obect(s): ' ...
    ,num2str(length(stats)))), 'FontName', 'Arial', ...
    'FontSize', 8, 'Color', [0.8 0.8 0.8]);

    subplot(2, 2, 3);
    if target~= 0
        plot(x,y,x(1),y(1), '-m+')
        a=text(x(1)+20,y(1)+20, strcat('X: ', num2str(round(bc(1))), ' Y: ', num2str(round(bc(2)))
    ));
        set(a, 'FontName', 'Arial','FontSize', 8, ...
        'Color', 'Black');
        axis([0 width 0 height]);
        set(gca,'xticklabel',{[]});
        set(gca,'yticklabel',{[]});
        set(gca,'XTick',[]);
        set(gca,'YTick',[]);
        set(gca,'Color',[0.9 0.9 0.9]);


    else
        plot(x,y)
        axis([0 width 0 height]);
        set(gca,'xticklabel',{[]});
        set(gca,'yticklabel',{[]});
        set(gca,'XTick',[]);
        set(gca,'YTick',[]);
        set(gca,'Color',[0.9 0.9 0.9]);
    end

    subplot(2, 2, 4);

    if target~= 0
        plot(0);
        axis([0 1 0 1]);
        axis off

        a=text(0,0.9, strcat('X: ', num2str(round(bc(1))), ' Y: ', num2str(round(bc(2)))));
        set(a, 'FontName', 'Arial','FontSize', 20, ...
        'Color', 'Black');

        angle = atan((bc(2)-prevY)/(prevX-bc(1)));

        if (bc(1)-prevX) < 0
            if (bc(2)-prevY) < 0
                angle = -(pi/2 - angle) - pi/2;
            else
                angle = pi/2 - angle + pi/2;
            end
        end

        if prevX-bc(1) == 0
            if (bc(2)-prevY) < 0
                angle = pi;
            else
                angle = 0;
            end
        end

        angle = angle * 180 / pi

        speed = 7 %Minimum speed to detect

        if sqrt(abs((bc(1)-prevX)*abs((bc(1)-prevX)) + abs((bc(2)-prevY)*abs((bc(2)-prevY))))) >
    speed
            text(0.5,0.5,'----->','Rotation',angle,'FontSize',20);
        end

        prevX=bc(1);
        prevY=bc(2);
    end

    subplot(2, 2, 1);
    imshow(data);

    hold on

    %Find the biggest object
    target = 0;
    value = 0;

    for object = 1:length(stats)
        bb = stats(object).BoundingBox;
        if bb(3)*bb(4)>value
            value = bb(3)*bb(4);
            target = object;
        end
    end

    %This is a loop to bound the red objects in a rectangular box.
    for object = 1:length(stats)
        bb = stats(object).BoundingBox;
        bc = stats(object).Centroid;
        %Color in blue the biggest object
        if object == target
            rectangle('Position',bb,'EdgeColor','b','LineWidth',1.5)
            x = [bc(1) x];
            y = [height-bc(2) y];
        else
            rectangle('Position',bb,'EdgeColor','r','LineWidth',1.5)
        end
        plot(bc(1),bc(2), '-m+')
        %a=text(bc(1)+15,bc(2), strcat('X: ', num2str(round(bc(1))), '     Y: ', num2str(round(bc(2)))));
        %set(a, 'FontName', 'Arial', 'FontWeight', 'bold', 'FontSize', 12, 'Color', 'yellow');
    end
    hold off
end
%Both the loops end here.

%Stop the video aquisition.
stop(vid);

%Flush all the image data stored in the memory buffer.
flushdata(vid);

%Clear all variables
clear all
sprintf('%s','End')
```

The final result of what the program do in real time is visible in the screenshot below:

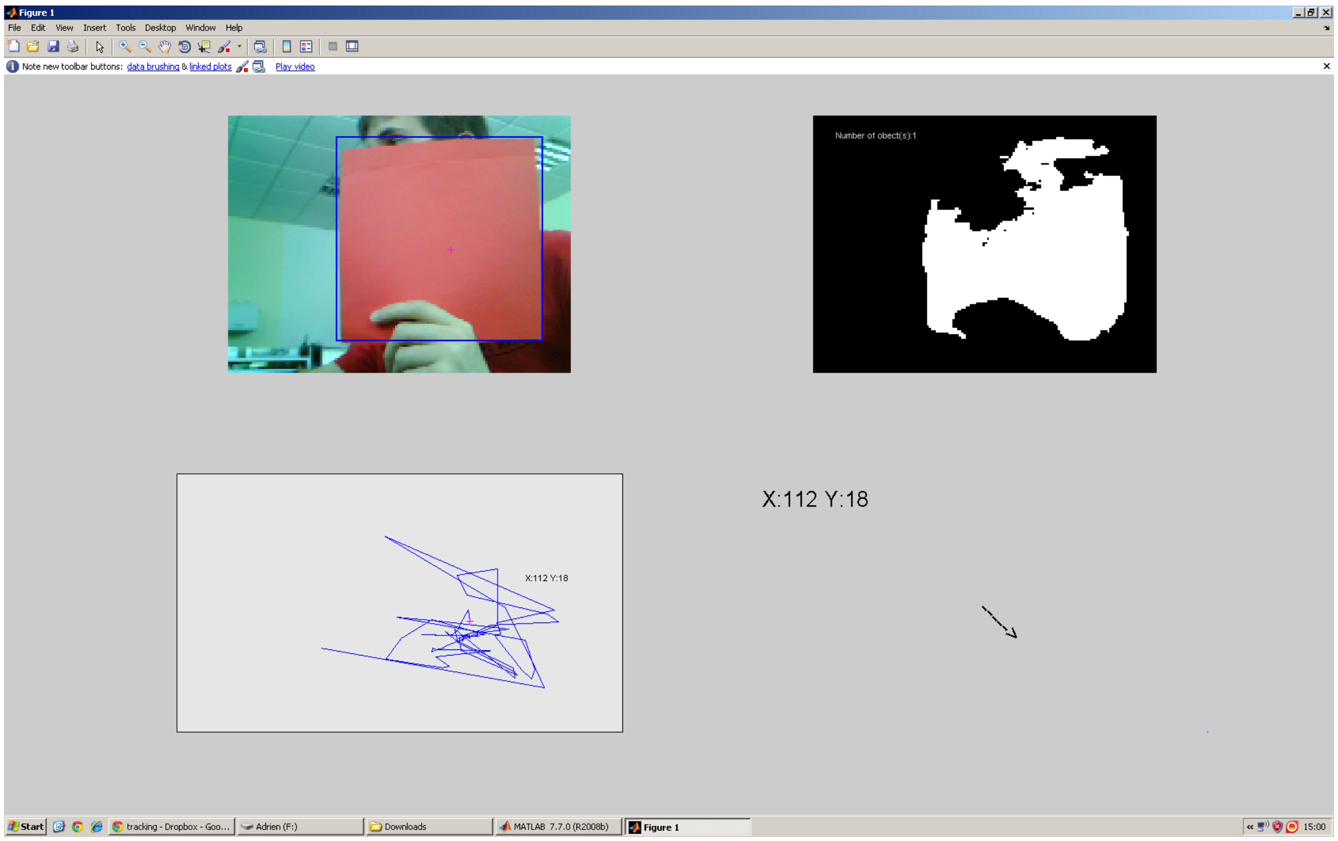


Figure 1
File Edit View Insert Tools Desktop Window Help
Note new toolbar buttons: data brushing & linked plots Play video
Number of obect(s):1
X:112 Y:18
X:112 Y:18
Start
tracking - Dropbox - Goo...
Adrien (F:)
Downloads
MATLAB 7.7.0 (R2008b)
Figure 1
15:00

top left: current frame with the red area surrounded by a blue rectangle

top right: binary picture

bottom left: trajectory of the center of the red shape

botom right: center's coordinates and current moving direction

Conclusion

Finally, we have seen that ARIA is a very powerful library and allow us to control the robot easily. Nevertheless, the ARIA library adapted to Matlab is not as powerful as those from C++, for example we can't do background task as easily as in C++, the compilation of this library is not so easy, it took a lot of time to run it perfectly. Maybe we will have to wait a little bit to have a better solution.

In the future it will be good to perform all tasks already made with the sonar but using the laser scanner. This captor is very accurate and can operate in a better way.

The iPhone application is very interesting but we need to have a proprietary application to make it run, it will be possible to create our own application (for Android).

The smooth trajectory project is not completed. Some issues must be corrected (especially about the coefficients, to work correctly in any situations). The best way to use it, must be with a function running in background, allowing us to do something else while the robot is moving.

The last advice will be to charge the P3-AT as soon as possible, batteries are not very efficient, and you will not be able to work on it more than 3 or 4 hours in the best conditions.