Stereo Vision Navigation of Autonomous Mobile Robot Based on Complex Positioning

Qiuhong Lu[*], Shaoyuan Li, Guozheng Yan
School of Electronics, Information and Electrical Engineering,
Shanghai Jiao Tong University,
Shanghai, China
[*] luqiuhong1@sina.com

Abstract—**A positioning and navigation method based on stereo vision, compass and encoders is presented. It has been used in an autonomous mobile robot which developed by the group of the authors. The compass and encoders are used complementarily each other to get correct position. 3D vision-based path-planning makes the robot walk along a better path each calculation cycle and avoid bumping other objects. This navigation system is cheap and convenient to create, and is effective in practice. Complex positioning, environment mapping, path-planning and behavior-based navigation algorithm are discussed is this paper.**

Keywords—Mobile Robot, Autonomous Navigation, Stereo Vision, Behavior Control

I. INTRODUCTION

When we refer to a robot's intelligence, a key problem is how to solve its navigation in real environments. There have been a lot of researches focusing on navigation algorithm, especially visual navigation, which is regarded as the highest level algorithm [1,2,3]. In the former studies, a stereo vision navigation algorithm has been used in our robot [3].

There are four basic problems relative with navigation: (1) apperceiving—the robot should interpret information from sensors and pick up useful data from them; (2) positioning—robot should know its own position and orientation in its environment; (3) cognizing—the robot should decide how to take action to achieve its goal; (4) motion controlling—the robot should adjust its movement to get expected track.

In the above four problems, positioning ability is the most basic problem for navigation. Assume that when the robot is operating, after it receives a command "move from present position to the goal", what action should it take? Obviously, it should know where itself is at first. As a person, because he/she have the sense of geography location, he/she know where himself /herself is in a room, or where the building are. He/she does not have to know the coordinates of own position clearly, but it is important for he/she to have the ability to remember the scenes and ability to distinguish own location. For a mobile robot, it is difficult to get the ability compared to a person. A robot's position is expressed as numerical format and is processed in this format. First, set the coordinate system in a certain point of its environment, then express its pose (position and orientation) concerning the coordinate system in numerical format.

In most mobile robot applications, two basic position-estimation methods are employed together: absolute positioning and relative positioning methods [1, 4-7].

Absolute positioning methods usually rely on (a) navigation beacons, (b) active or passive landmarks, (c) map matching, or (d) satellite-based navigation signals. Each of these absolute positioning approaches can be implemented by a variety of methods and sensors. Yet, none of the currently existing systems is particularly elegant. Navigation beacons and landmarks usually require costly installations and maintenance, while map-matching methods are either very slow or inaccurate [8], or even unreliable [9]. With any one of these measurements it is necessary that the work environment either be prepared or be known and mapped with great precision. Satellite-based navigation (GPS) can be used only outdoors and useless for robots walking indoors.

Relative positioning is usually based on dead-reckoning (i.e., monitoring the wheel revolutions to compute the offset from a known starting position). Dead-reckoning is simple, inexpensive, and easy to accomplish in real-time. The disadvantage of dead-reckoning is its unbounded accumulation of errors. Another approach to the position determination of mobile robots is based on inertial navigation with gyros and/or accelerometers. It can lessen accumulation of errors, but these sensors are exceedingly sensitive to drift, and any small drift will be enlarged by accumulating [10, 11]. Electronic compasses can determine the local vector toward the north magnetic pole, so it has no accumulated errors, and it can avoid the sensor's drift problem of inertial navigation. Moreover, compasses are easily and cheap to be installed in robots.

Thought over above positioning methods, we selected compass assisted by encodes to do positioning work in the navigation system of our autonomous robots. As shown in Figure.1, a compass was fixed inside the robot and a stereo camera was set in the front. We used a "Bumblebee" stereo vision in this research. The camera can rotate in two directions due to a Pan & Tilt mechanism under it. With a compass in its sensor board, the robot can measure the orientation of itself relative to the earth magnetic without accumulated errors. In order to reduce the influence of the compass suffered from outside magnetic field, encoders were used to correct angle values from the compass in the positioning process. During the walking behavior along the path preplanned, the robot calculates its position continuously and compares with the goal's coordinates until it arrived at the target. And it will give up the chosen way and plan paths again if an obstacle stands in front of it, because the obstacle-avoidance behavior has the highest priority in all behaviors.

The complex positioning, vision-based mapping and path-planning and behavior-based navigation algorithm are discussed in detail in this paper.

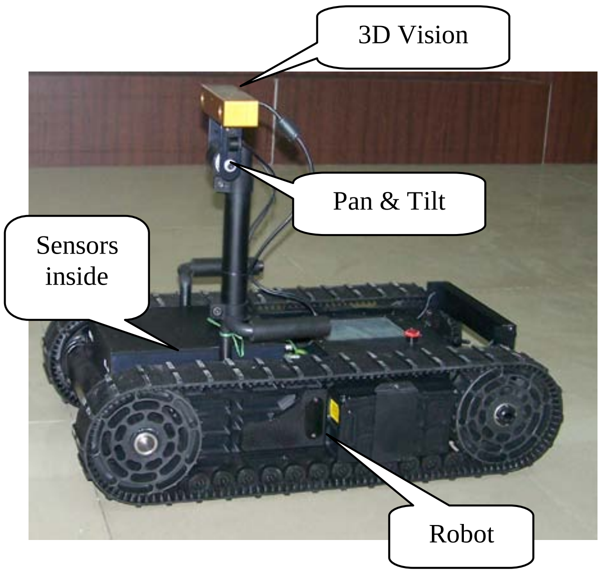


Figure 1. A mobile robot with navigation system

II. Positioning Algorithm

A. Dead-reckoning Positioning Technique

Encoders feed back revolution angles of the robot's wheels in real-time. By accumulating revolutions of left and right wheels, the robot can get its current position. This is a type of positioning algorithm named as dead-reckoning, which is used very popularly in mobile robotics.

The translational velocity and rotational angular velocity are as follows:

$$\begin{cases} v[t] = \dfrac{(v_L + v_R)}{2} \\ \omega[t] = \dfrac{(v_L - v_R)}{B} \end{cases} \tag{1}$$

where,

v_L and v_R is the right velocity and right velocity of the robot respectively;

B is the distance between the two wheels.

We can write the formula of dead-reckoning in discrete form as follows:

$$\begin{cases} x_{n+1} = x_n + \Delta u_{n+1} \cos\theta_n \\ y_{n+1} = y_n + \Delta u_{n+1} \sin\theta_n \\ \theta_{n+1} = \theta_n + \Delta\theta_{n+1} \end{cases} \tag{2}$$

where,

[x_n , y_n , θ_n] is the pose vector of the robot at a certain moment;

[x_{n+1} , y_{n+1} , θ_{n+1}] is the new pose vector of the robot.

Δu is the positional increment from old position to the new one. It is the average of left and right wheel's displacement increments.

$$\Delta u = \frac{\Delta L + \Delta R}{2} \tag{3}$$

$\Delta\theta$ is the orientation angle increment of the robot from old orientation to the new one. It is in direct propotion to the difference between two wheel's displacement increments:

$$\Delta\theta = \frac{\Delta L - \Delta R}{B} \tag{4}$$

If know the original pose in advance, measure data of the encoders fixed on left and right wheels, we can calculate the robot's new pose conveniently. Dead-reckoning is inexpensive, has higher precision in short term and higher sampling speed, so it is used widely. Also note that the new orientation is derived from the wheel travel distance, which is likely to be corrupted by wheel size errors, slippage, skidding, and related effects. Moreover, the error will be zoomed by integral. So it is not suit to long term situations.

B. Compass Positioning Technique

From the compass inside the robot, yaw, pitch (left-right) and roll (front-rear) angles can be gained. The pose of the robot can be calculated according to formula as follows.

$$\begin{cases} x_{n+1} = x_n + \Delta u_{n+1} \cos\theta_n \cos\phi_n \\ y_{n+1} = y_n + \Delta u_{n+1} \sin\theta_n \cos\phi_n \\ z_{n+1} = z_n + \Delta u_{n+1} \sin\phi_n \\ \theta_{n+1} = \theta_n + \Delta\theta_{n+1} \end{cases} \tag{5}$$

where,

[x_n , y_n , θ_n] is the pose vector of the robot at a certain moment;

[x_{n+1} , y_{n+1} , θ_{n+1}] is the new pose vector of the robot;

Δu is the positional increment from old position to the new one;

θ represents the yaw, and ϕ represents the pitch;

$\Delta\theta$ is the orientation angle increment of the robot.

Robot can know its new pose through a compass. However, the positioning results solely from compass is not enough or reliable. At most points within a large area, the compass will report an accurate heading. But there are always isolated spots near ferrous objects; the compass may report results errors, as much as 180° can easily occur.

C. *Complex Positioning Technique*

The two above methods have its own advantages and disadvantages. Dead-reckoning is simple but has accumulation of errors, where compass might be influenced greatly by ferrous or electromagnetic environment. We can combine both of them to get better positioning results.

Note that a compass may have unexpected errors, however, orientation based on information supplied by shaft encoders tends not exhibit sudden large errors. So we can use encoders to reduce the influence of the compass suffered from outside magnetic field [10].

Let θ_{rec} represent the best possible estimate yaw, θ_{comp} represent the magnetic heading measured by the compass; θ_{enc} represent the heading supplied by the encoders, which is the tracking of the robot's rotation and has no particular relationship to north. If we want θ_{rec} to represent the robot's true heading and if θ_{rec} is derived from θ_{enc}, we will have:

$$\theta_{rec} = \theta_{enc} + \theta_{corr} \tag{6}$$

where

θ_{corr} is a correction, which is the difference between the true heading of the robot and the heading given by the encoders alone. If we make a single measurement at a point far from any interfering ferrous objects, we can write:

$$\theta_{corr} = \theta_{comp} - \theta_{enc} \tag{7}$$

Because θ_{comp} is usually correct, we can take an average, that is:

$$\theta_{rec} = \theta_{enc} + \overline{\theta_{corr}} \tag{8}$$

Note that $\overline{\theta_{corr}}$ represents an average over distance rather than a time. Thus, the value θ_{enc} drifts slowly as the robot moves, but is corrected by regular readings of the electronic compass. The electronic compass gives an occasional inaccurate reading when the robot comes within a few feet of a ferrous object, but by averaging over a long distance, the effects are damped out. By this mean, the robot can know its own position clearly.

III. Environment Mapping

When the robot has known its real-time position, it should know how to get to the goal and protect itself against bumping into any object on its way. The stereo camera installed in the front of the robot takes charge of this task. While it is running, the robot turns the camera rotate left-right and up-down. By this way, the robot can get environment information around itself, create the map of environment, then judge where it can go.

A. *The Principle of Stereo Vision*

Stereo vision is an essential technology of getting 3D information from images. It is also called "binocular vision". Two CCD cameras catch two images of the same scene from different point of view, just as the human's eyes. Based on the principle of stereo parallax, by establishing correspondence between a same point in two images, the depth of the point is calculated. By this way, we can achieve the 3D measuring of all features in the scene.

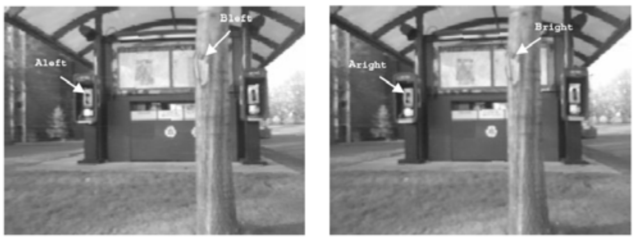
It needs three steps to perform stereo processing:

1. Establish correspondence between image features in different views of the scene.

2. Calculate the relative displacement between feature coordinates in each image.

3. Determine the 3D location of the feature relative to the cameras, using the knowledge of the camera geometry.

First step is image matching. Figure 2 is an example, where the point *Aright* corresponds to the point *Aleft*; similarly, point *Bright* corresponds to the point *Bleft*. In order to find the best pixel *Pr* from the right image matching the pixel *P1* in the left image, we need to search for all pixels along the same row in the right image as in the left image. Simply, set a rectangle window around the candidate pixel *Pr* and the same size window around the testing pixel *P1*; calculate the simular level between two windows; the pixel *Pr* has the greatest similarity will be picked out.



(a) left image (b) right image

Figure 2. Stereo matching

There are several estimation methods for matching two images: Sum of Absolute Difference (SAD), Sum of Squared Difference (SSD), Normalized Cross Correlation (NCC) and etc. The stereo image matching in this research uses SAD. Its extimation function is as follows:

$$\min_{d=d_{\min}}^{d_{\max}} \sum_{i=m/2}^{m/2} \sum_{j=m/2}^{m/2} I_{left}[x+i][y+i] - I_{left}[x+i+d][y+i] \tag{9}$$

where,

$d_{\min}$ and $d_{\max}$ are the minimum and maximum disparities;

m is the mask(window) size;

I_{left} and I_{right} are the left and right images.

After all pixel matching, we can calculate the disparity of every pixel, the depth of every point by formula as follows,

$$z = B \times \frac{f}{d} \tag{10}$$

where,

B is the baseline;

f is the foci of cameras.

and every point's 3D coordinates by formula as follows,

$$\begin{cases} x = z \times \dfrac{x_l}{f} \\ y = z \times \dfrac{y_l}{f} \end{cases} \tag{11}$$

where,

x_l and y_l are the coordinates of the point's projection in the left image respectively.

According to the above method, depth data can be calculated. Figure 3 shows the left image, right image and depth image. The robot can do environment mapping, path-planning and autonomous navigation with these depth information.

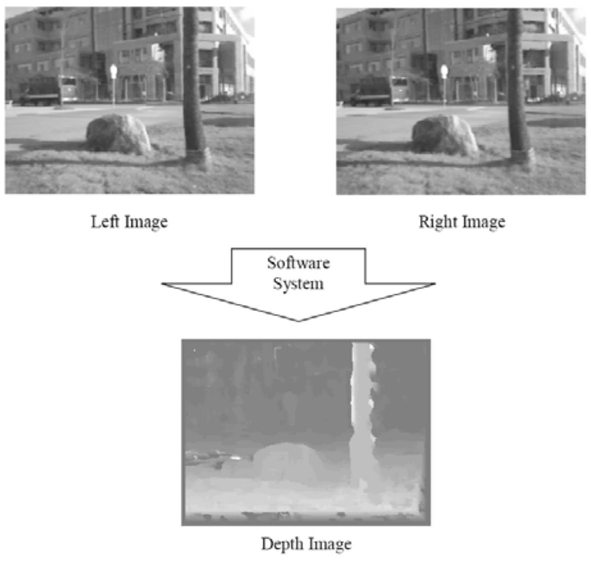


Figure 3. Left image, right images and depth image[12]

B. *Transformation of Coordinates*

The 3D coordinates information of the object derived from the stereo vision are the representations in the camera's world system. The robot has to do coordinates conversion to transform these information into its world system, before it can use them to do environment mapping and navigation.

In Figure 4 we create the robot's world coordinate system *XYZ* and the camera's world coordinate system *xyz*. The origin of *XYZ* is the centre of the two wheel's axis, *X* is to front, *Y* is to left, and *Z* is to up. The origin of *xyz* is the centre of the right camera, *x* is to right, *y* is to down and *z* is to font. There is a relationship between *XYZ* and *xyz*:

$$\begin{cases} Y = x + B/2 \\ X = z + A \\ Z = -y + H \end{cases} \tag{12}$$

where,

B is the baseline of the camera;

H is the vertical distance between the origin of the robot system and the camera;

A is the horizontal distance between the origin of the robot system and the camera.

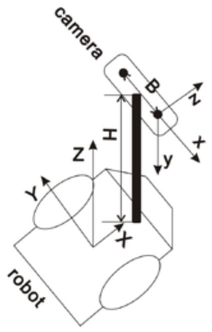


Figure 4. Robot's and camera's world coordinate systems

C. *Mapping*

After getting the depth information and transforming them to its world, the robot gets coordinates of each object which it encounters, then creates a map of around environment.

First, we cut out a window from the visual field of the robot (get rid of pixels include incomplete information or difficult to process, only process partial information to reduce calculating amount). Then transform the coordinate information to the values in the robot's world coordinate system. A grid map with depth information is built as in Figure 5.

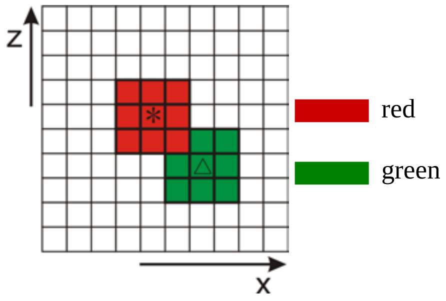


Figure 5. Mapping

Where, *x* is to right and *z* is to up; pixels in the upmost row are corresponding to object points farthest from the robot; pixels in the undermost row are corresponding to object points nearest to the robot. The window is divided into small grids uniformly. Each grid represents a certain practice size. Suppose that a grid is corresponding to an area of 10cm*10cm, the robot's width is 30cm, when a grid is taken by obstacles, the area of 3*3 around it is impassable for the robot.

There are five steps in the mapping algorithm:

1. Scan the map grid by grid along *x* direction and row by row from down to up. Each grid has two flags: *Obstacle_flag* and *Passable_flag*.

2. For each scanning point, mark its state according to obstacle evaluation algorithm: if there is an obstacle, set *Obstacle_flag*=1(*true*); otherwise, *Obstacle_flag*=0(*false*). Obstacle flags are figured as symbol * (is true)and △(is false) in Figure 5 respectively.

3. Use *m***m* mask to each scanning point, if *Obstacle_flag*=1(true), then set all points behind the mask *Passable_flag*＝0(false); otherwize, all points *Passable_flag*＝1(true). Passable flags are represented as red (is false) and green (is true) in Fig. 2. A red area is impassable where a green one is on the contrary.

4. *Passable_flag* is always false if only it is set to false once. Set its initial value true, an "and' operation can be used here: *Passable_flag*=(*last_Passable_flag*) and (*new_Passable_flag*). As shown in Figure 5, a red area may cover part grids of a green area.

5. The grids on the edges of the map difficult to be estimated will mark with unknown area, colored yellow.

IV. PATH-PLANNING ALGORITHM

According to the map of the environment, the robot can calculate a best path to the goal away from obstacles. There are two processes as obstacle evaluation and path choosing.

A. *Obstacle evaluation*

This process is to define what is an obstacle that the robot should avoid.

1. slope obstacles

A slope threshold is applied to detect obstacles too steep for the robot to climb. It is preset and can be adjusted in practice. For one evaluating grid (i, j), its following grid is (i, j+1) , the slope will be:

$$slope=arctan((y[i,j+1]-y[i,j])/w) \tag{13}$$

wherein, w is the width of the grid, y is the height of the object in the grid.

2. overhanging obstacles

This threshold is applied to detect "overhanging" obstacles that don't touch the ground. The robot can not pass a hanging object whose height is lower than the robot height. Two thresholds are preset here, the low limit and the high limit. The low limit can not be less than the height of the robot, while the high limit can not be greater than the height of sills which the robot can climb.

In a grid y coordinates are the heights of objects. If the minimum height is smaller than the low limit, and the maximum height is bigger than the high limit, there will be a overhanging object, so we mark the grid with *obstacle_flag*= true. If the minimum height is smaller than the low limit, but the maximum height is smaller than the high limit, the robot ignores the object because the robot can climb over it.

Note here the height value y should be subtracted the radius of the robot's wheel, because the origin of the robot's world coordinate system is the centre of the two wheels axes, but the thresholds are preset based on the ground.

B. *Path Choosing*

Starting from the radial upright forward from the center of the robot, draw a cluster of curves leftward and rightward. One curve is corresponding to a path of the robot. The paths have the same translational velocity, but different rotational angular velocity. One method of ours is to set the diameter of the robot is *D*, a path is corresponding to an arc in a circle whose diameter is several times of *D* respectively. According to diameters of arcs, all rotational angular velocities have been calculated.

A path is an optional path just when all grids in it are passable (green). If a path includes any impassable grid, it will be removed. If there are more than one optional paths, it is necessary to choose the best one. A lot of method can to use in path choosing. A simple is the nearest rule, i.e. choose the path in the end the robot is the nearest to the goal.

Figure.6 shows the terrain map and path-planning map. Each grid represents the practical space 0.1m*0.1m. There are 23 paths in all, and the robot is located at the start of the 23 paths. The robot deals with information in the area 3 meters before itself every pace, which is 20cm.

In the figure, red grids are impassable with obstacles, green grids are passable without obstacles, and yellow grids are unknown area. Red paths are impassable and green paths are passable. The white path is the optimum one which the robot is walking on currently.

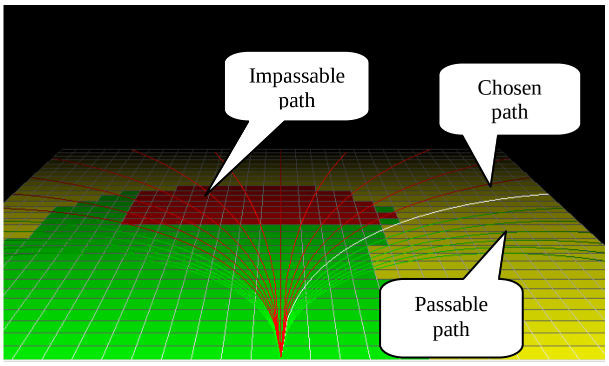


Figure 6. Path-planning based on map

V. BEHAVIOR-BASED NAVIGATION ALGORITHM

The navigation algorithm presented in this paper includes four steps: (1) goal set, (2) positioning, (3) obstacle avoidance, (4) path-planning. We use behavior-based control algorithm in programming.

Figure 7 is the behavior control block diagram of the navigation algorithm of the robot. The bottom behavior is life, also called as behavior surviving. The second layer is the avoid/path-planning behaviors implemented by stereo vision. These behaviors can be implemented all together. After mapping and path-planning, avoidance is also achieved, so other sensors are not necessary here for obstacle avoidance. The third layer is autonomous positioning behavior of the robot, which can be achieved by dead-reckoning with compass and encoders. The top layer is goal picking, which is achieved by picking points from the images caught by stereo vision.

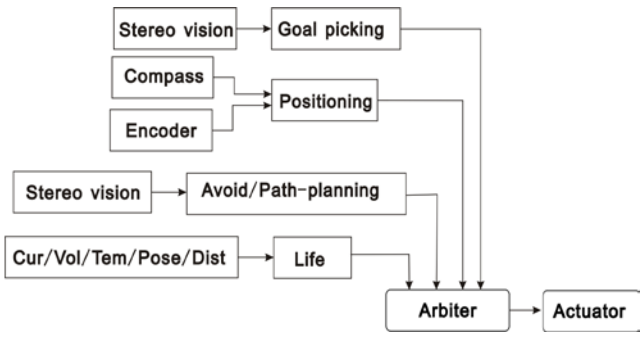


Figure 7. Behavior control block diagram of vision navigation

VI. CONCLUSIONS

In this paper, a complex of compass and encoders positioning technique and a navigation method based on stereo vision are presented. It has been used in an autonomous mobile robot which developed by the authors. The compass reduces possible accumulated errors of dead-reckoning and encoders correct possible great compass errors. By this complex positioning method, the robot can get accurate pose data.

Based on the stereo camera, depth information helps the robot take a better path each cycle and avoid bumping others. During running, the position of the robot is compared with the position of goal, when their difference is less than the preset threshold, robot will stop.

This navigation system is convenient and cheap to configure. It is able to assist the robot get the goal position appointed in the program at the beginning with less than 0.5 meter errors.

REFERENCES

[1] Chenavier, F., Crowley, J., “ Position Estimation for a Mobile Robot Using Vision and Odometry ” . Proceedings of IEEE International Conference on Robotics and Automation, Nice, France, May 12-14, pp. 2588-2593. 1992

[2] Eric Krotkov, Martial Hebert, and Reid Simmons. “Stereo Perception and Dead Reckoning for a Prototype Lunar Rover” . Robotics Institute Carnegie Mellon University

[3] ZHANG Guo-wei, Lu Qiu-hong, “A Path-planning algorithm of mobile robots Based on Stereo Vision” , 2008 sino-European Workshop on Intelligent Robots and System,2008.12

[4] Borenstein, J., Koren, Y., “ Motion Control Analysis of a Mobile Robot.Transactions of ASME ” , Journal of Dynamics, Measurement and Control, Vol. 109, No.2, pp. 73-79. 1987

[5] Hollingum, J., “Caterpillar make the earth move: automatically” . The Industrial Robot, vol. 18, no. 2, pp. 15-18. 1991

[6] Evans, J. M., “HelpMate: An Autonomous Mobile Robot Courier for Hospitals” , 1994 International Conference on Intelligent Robots and Systems (lROS '94). München, Germany, September 12-16, pp. 1695-1700. 1994

[7] Siegwart,R., Nourbakhsh,I.R., “Autonomous Mobile Robots” . Press of Xi’an Jiaotong University, 2006,9

[8] Cox, I. J., “ Blanche — An Experiment in Guidance and Navigation of an Autonomous Robot Vehicle” . IEEE Transactions on Robotics and Automation, vol. 7, no. 2, April, pp. 193-204. 1991

[9] Congdon, I. et al., CARMEL Versus FLAKEY — A Comparison of Two Winners”. AI Magazine Winter, pp. 49-56. 1992

[10] Barshan, B., Durrant-White, H.F., “An Inertial Navigation System for a Mobile Robot. Proceedings of the 1st IAV” , Southampton, England, April 18-21, pp. 54-59. 1993

[11] Barshan, B., Durrant-White, H.F., “Orientation Estimate for Mobile Robots Using Gyroscopic Information” . 1994 International Conference on Intelligent Robots and Systems (lROS '94). München, Germany, September 12-16, pp. 1867-1874. 1994

[12] Point Grey Research Corporation. Bumblebee Stereo Vision Camera Systems. 2007,9