Omnidirectional Vision Simulation and Robot Localisation

Christopher Burbridge **Libor Spacek**
Department of Computer Science
University of Essex
Wivenhoe Park
Colchester, CO4 3SQ
{cjcbur, spacl}@essex.ac.uk

Abstract

Most work that is carried out for omnidirectional robot vision is done on real hardware, with expensive mirrors. In this paper a robot simulator with accurate omnidirectional vision capability is presented. Vertically aligned co-axial omnistereo vision with two conical mirrors (Spacek, 2005) is tested using the simulator. Experiments are performed to calculate the distance to six objects and the results are compared to the known distances. The calculated distances are then used in a simple natural landmark based localisation algorithm, and the estimated position of the robot is compared to the actual position.

1. Introduction

Computer vision is a rapidly progressing discipline and its relevance to robotics is obvious – of all robot sensors available, vision provides the most information. However, with this additional information come new problems such as increased computation time and complexity. Despite these difficulties, vision is rapidly becoming the sensor of choice in robotics.

Navigation and environment sensing are essential for mobile robots. The ability to estimate position in an environment is often crucial. Omnidirectional vision offers detailed information about the environment in all directions, and as such is ideally suited for use in robot localisation.

Omnidirectional vision sensors have been constructed in many different ways. (Tan et al., 2004) use a pyramid of mirrors, and point multiple cameras at the pyramid. Different directions are reflected into each camera, and these pictures are then combined to form a panoramic picture. This configuration offers high resolution, and the possibility of a single view point. However, the camera arrangement can be complicated. Rotating cameras and mirrors were used by (Kang and Szeliski, 1997) and (Ishiguro et al., 1992). However, difficulties were encountered with registration and the motion of the camera. Wide angle and fisheye lenses have been used by (Swaminathan and Nayar, 2000) and (Shah and Aggarwal, 1997). Methods of rectifying some of the distortion of the images caused by these lenses have been developed but problems are still encountered with some distortion and the lack of a single effective viewpoint.

Catadioptric omnidirectional sensors have a single mirror and camera. The mirror is rotationally symmetrical and the camera points along its rotational axis. See figure 1 for an image of a conical catadioptric sensor. One major advantage that this design has over other omnidirectional sensors is the faster image taking, as there is no need to wait for a moving camera or synchronise multiple cameras. This leads to better suitability to dynamic environments, and mobile robotics.

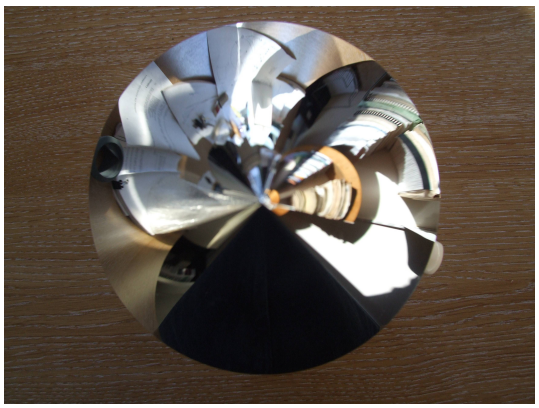

Figure 1: An image of a conical mirror.

There are several types of mirror that can be used in a catadioptric setup, and in this work we use the conical mirror. This has the benefit of producing no radial distortion or loss of radial resolution, as is produced by hyperbolic mirrors. However, conical mirrors produce multiple effective viewpoints, which until recently was problematic. (Spacek, 2005) shows that a single effective

viewpoint is not actually required for a correct perspective projection.

Most experimental work that is carried out for omnidirectional robot vision is done on real hardware, using expensive mirrors, cameras and robots. If a simulator was to be used then the cost could be minimised and the level of productivity may increase. Results such as distance could be compared to the *ground truth*, which will be known precisely. In this paper we develop software that simulates the computer vision of a robot, in particular catadioptric omnidirectional vision.

We then create a simulated robot containing two catadioptric omnidirectional sensors, vertically aligned, and present a simple algorithm for calculating the disparity of basic shapes between the two images. Finally, we use the distance to these basic shapes in order to calculate the predicted position of the robot, and we compare this to the actual position.

2. Omnidirectional Vision Simulation

Many robot simulators exist, but few simulate vision. Amongst the few that do is Webots^{TM} (Michel, 2004), a commercial 3D robot simulator capable of accurately simulating the physics of mobile robots. It also allows the simulation of robot sensors, including sonars, lasers and cameras. Computer vision on robots can be programmed and tested using just this simulator. However, there are some concerning limitations that prevent its use in this work. Firstly, mirrors cannot easily be created, meaning that catadioptric omnidirectional vision cannot be done. Secondly, the simulation of vision is performed using openGL. This is a 3D graphics library which doesn't accurately simulate the physics of vision, and might lead to problems when performing complex computer vision tasks such as measuring the disparity between images for stereo vision.

Other robot simulators that are capable of simulating cameras also make use of openGL. These include Gazebo (part of the Player/Stage project) and EyeSim (Bräunl, 2003). An alternative to using graphics libraries such as openGL for simulating vision is to use ray tracing. This takes a single scene, and traces rays of light through it, taking into account the surface types and colours, refraction, reflection etc. The resulting image is physically accurate, containing realistic shadows and reflections. For our simulator we use POV-Ray (http://www.povray.org/), a free open source ray tracer.

2.1 POV-Ray Raytracing

POV-Ray can be used as a command line tool, with a scene file being passed to it as an argument. The scene files are plain text files that define all the shapes, objects, cameras and lights in the scene in a simple structured way. POV-Ray will analyse the scene and then output the resulting view to an image file. This approach lends itself well to the development of a robot simulator, as the simulator can rely entirely on POV-Ray for all image generation tasks and the structure of scene files is simple to read. POV-Ray is also capable of producing animations but only along a fixed path which cannot be changed by an outside program at runtime, as is needed in a dynamic robot simulator.

Complex shapes, mirrors and multiple light sources and cameras are all supported by POV-Ray. Different camera types can be used, including perspective, fisheye and panoramic, and their resolution can be defined. Scene files can be created graphically, if desired, by using a graphical modelling front end. In our work we use Kpovmodeler, an open source interface for the Linux platform. This allows easy visualisation of the environment, producing the POV-Ray script mostly automatically.

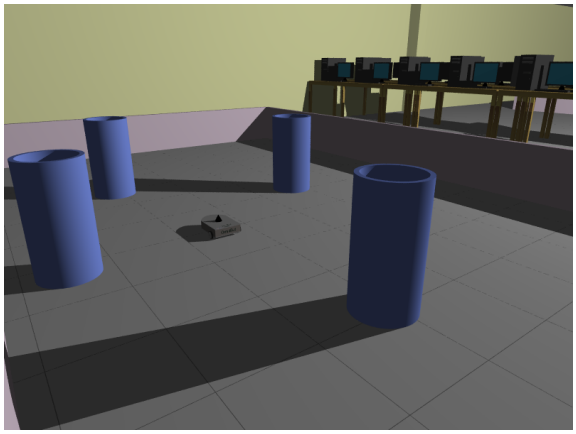

Figure 2: An example lab scene generated with POV-Ray. This is loosely modeled on the Brooker lab at the University of Essex.

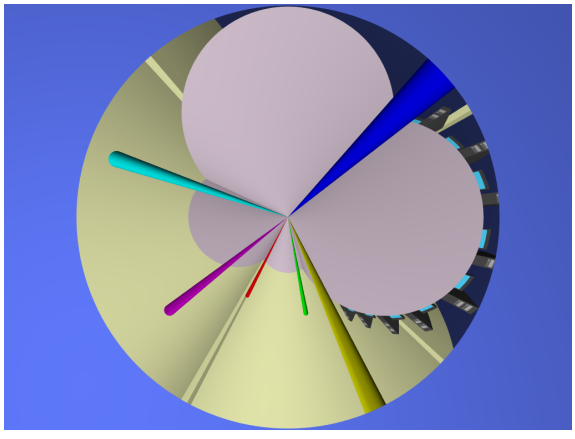

Figure 3: An image of a simulated catadioptric conical mirror.

The main disadvantage of using raytracing rather than

openGL and the like is speed. Rendering a single frame using POV-Ray takes approximately 1 second, meaning that the simulator does not work in realtime. To run the robot for five minutes at 15 frames per second, with two cameras for stereo vision, currently requires approximately 2.5 hours to generate the images. However, a video sequence can be generated and stored for analysing at a later point if desired.

2.2 Robot Motion and API

Our simulator is split into two parts, a server, and a client controller program. Communication between the two takes place transparently through shared memory. This is achieved by using a Robot class that client applications use to manipulate the robot, and receive images from cameras.

When starting up, the server loads a specified scene file and scans it to locate cameras and the robot. Once the locations are known, it waits for commands through the shared memory space. A simple API has been implemented in the simulator to allow control programs to set the wheel velocities of the robot, find the exact location of the robot and cycle to the next video frame. When asked to cycle, the server moves the robot and camera objects in the scene file, and passes it to the POV-Ray engine to produce the next video frames. Basic kinematic equations for two-wheel differential drive robots are used when calculating the position of the robot from frame to frame.

See figure 4 for a block diagram showing how the various aspects of the simulation fit together.

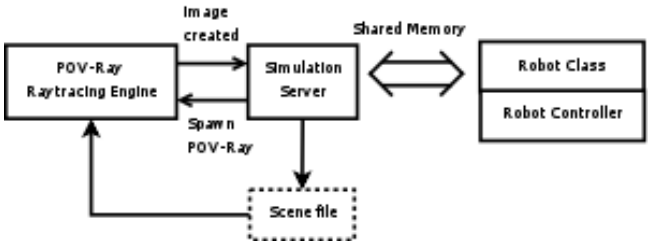

Figure 4: A block diagram showing how the various aspects of the simulation fit together.

2.3 Omnistereo

Since any number of cameras can be created in the simulator, simulation of stereo vision is instantly possible. Any imaginable stereo setup can be constructed easily and tested without the need for the expensive and time consuming building process that usually accompanies omnidirectional vision research.

For this work we have modelled vertically aligned omnistereo, and experiments have been performed using the simulator to produce a video sequence. The video sequence is then analysed, and the range to known objects is calculated by using the disparity between images. See figure 5 for an image of the modelled robot. Full details of the experiments and results are presented in section 3.1.

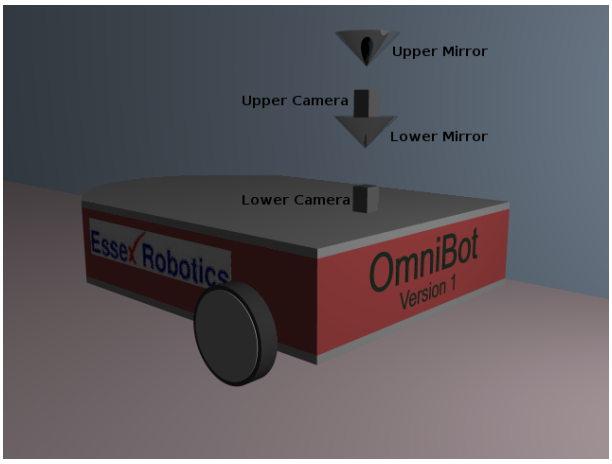

Figure 5: The simulated robot equipped with two vertically aligned conical mirrors and two cameras. Each mirror has an angle of 90° at the tip.

2.4 Future Developments

Although a very useful tool already, the simulator developed is very much a work-in-progress. Collision detection is left up to the controller program and the robot will currently quite happily drive through obstacles. Although the basic kinematic equations are used for calculating the robot position, friction and momentum are not taken into account. In future developments the Open Dynamics Engine (ODE) could be used to provide more realistic physics for the simulation.

Another desirable development would be to implement dynamic objects, and multiple concurrently running robots.

3. Robot Localisation using Omnistereo

3.1 Experimental Method

3.1.1 Virtual Arena Setup

For our experiments we set up the simulated arena as shown in figure 6, and configured the robot for vertically aligned omnidirectional stereo as shown in figure 5. Six distinctly coloured pillars were placed at known locations around the edge of the arena.

3.1.2 Random Wandering Behaviour

The robot was programmed with a random wandering, obstacle avoiding behaviour. This was run for 450 frames

(30 seconds at 15 frames per second), and all images were saved alongside the actual robot trajectory. These frames were then processed to calculate the distance to the pillars, and the estimated position of the robot in the arena. The results are shown in section 4.

In order to achieve obstacle avoidance a map of the arena was programmed into the control program. The controller drives the robot forwards, checking the map until it is at a pillar or wall. It then randomly turns either left or right for a random duration and continues. The trajectory of the robot is shown in figure 7.

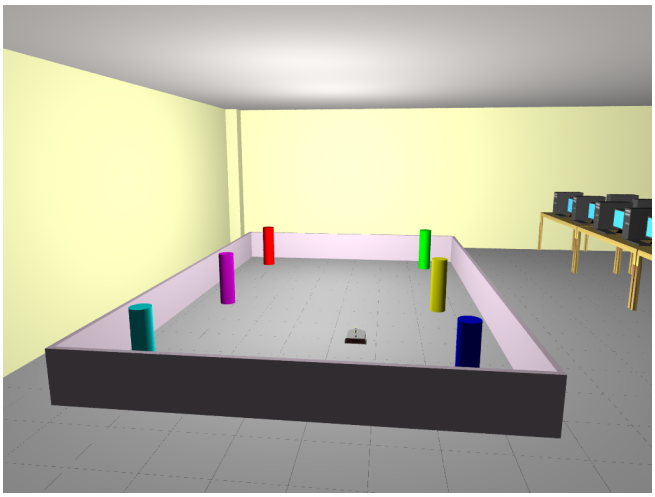

Figure 6: The lab setup. Six pillars are positioned around the arena at known locations for landmark based localisation.

3.2 *Omnistereo Range Finding*

The range to objects can be calculated using a pair of omnidirectional images in much the same way as in classical forward looking stereo vision. The formula for calculating the range (Spacek, 2005) is:

$$range = \frac{vs}{h_{i1} - h_{i2}} - d \tag{1}$$

where d is the distance from the camera lens to the tip of the mirror, s is the distance between the mirrors, v is the focal length of the camera and $h_{i1} - h_{i2}$ is the radial disparity between images. This is shown if figure 8.

To calibrate v and convert between pixels and mm, v is calculated as (Spacek, 2005):

$$v = (\frac{d}{R} + 1)r_m \tag{2}$$

where R is the radius of the mirror, and r_m is the radius of the mirror in the image.

These formulae are only true for 90° mirrors, as any other angled mirror will not result in a cylindrical coordinate system.

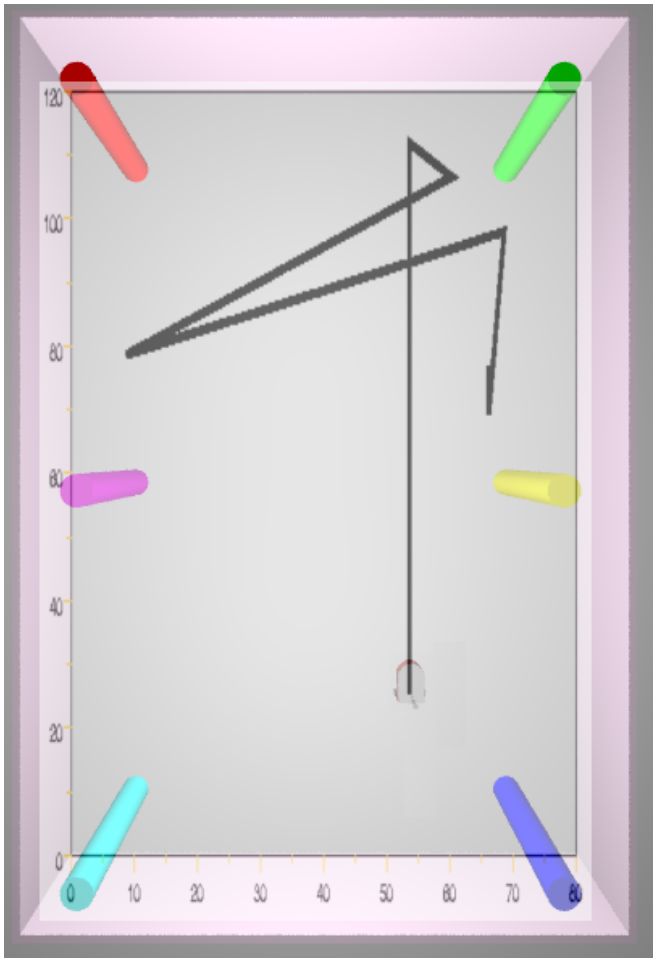

Figure 7: The trajectory of the robot overlayed onto an overhead image of the arena. Measurements are in POV-Ray units. 1 unit is 5 cm.

The distance to each of the six coloured pillars in each video frame was calculated by measuring the disparity of the top of the pillar between upper and lower images, and applying equations 1 and 2 above. All the distances were recorded and the results are shown in section 4.

3.2.1 *Calculating Disparity*

The radial disparity between upper and lower mirror can be seen quite clearly in figure 9. The algorithm used for locating the pillars is as follows:

1. Look at the lower mirror, and test all the pixels around a circle centred on the tip of the cone, with the same radius as the cone to see if they are above the colour threshold. If so, stop testing circles and store the location, if not then repeat with a smaller circle.

2. Look at the upper mirror image and test all the pixels along a line from the the location stored for the lower

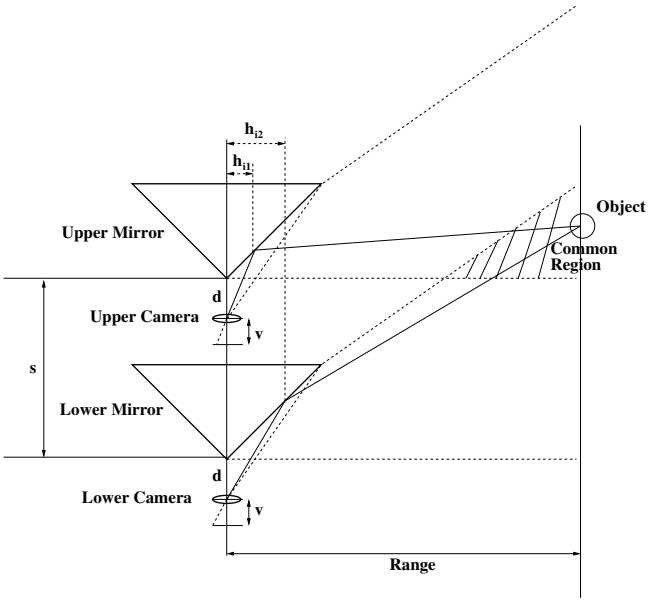

Figure 8: Diagram showing the vertical aligned stereo configuration. The distance to an object can only be computed if it falls in the common region.

mirror to the centre of the circle. Store the location of the first pixel above the colour threshold.

3. Calculate the distance between the two stored points, and store as the disparity.

This is repeated for each of the six colours.

3.3 Position Calculation

Having calculated the distances to objects, the known location of the objects was used for simple landmark based localisation. This was done by calculating the point of intersection between circles with radius equal to the estimated distance between the robot and object, centred on the objects. The circles will intersect at two places, so three distances are required. Since the distance measured to the object is never going to be precise, a triangle of intersections will form which the robot will be within. The centre of this triangle is recorded as the robot position. See figure 10.

4. Results

4.1 Omnistereo vision

450 frames were generated using the simulator, and for each frame the distance to the six colour pillars was calculated. The following table shows a summary of the results:

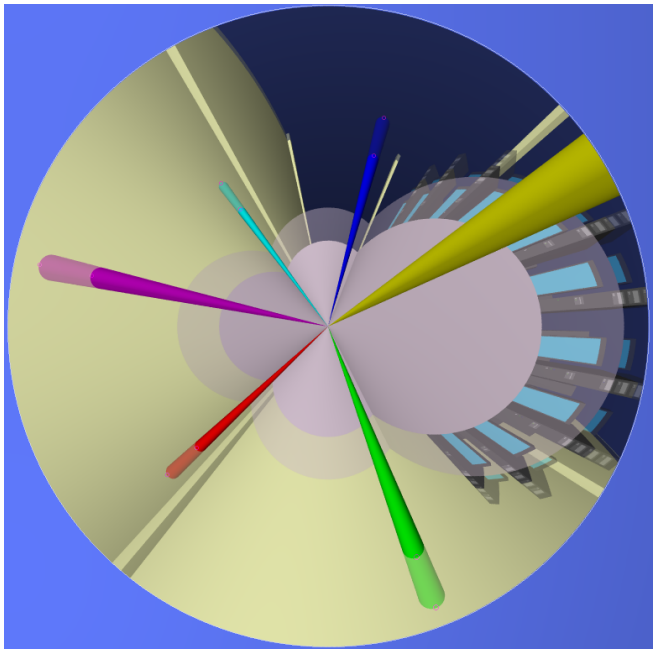

Figure 9: The disparity between upper and lower images can be seen quite clearly, with the image of the upper mirror superimposed on the lower. The circles show where the algorithm has matched. The top of the yellow pillar is not visible in the mirrors, so its range cannot be calculated.

Pillar	Max. error/mm	Average error/mm
Red	136.6	21.95
Green	124.8	25.42
Blue	256.9	51.97
Cyan	250.1	62.93
Magenta	110.5	23.17
Yellow	75.28	20.14

Graphs of distance against frame number for each object are presented in figures 11, 12, 13, 14, 15 and 16. When it is not possible to calculate the distance to an object, for example if it is too close or occluded, it is recorded as 0. It is clear from the graphs that as the distance of the object increases, the computed distance becomes more volatile. This is also shown by the average error for both the blue and cyan pillars being far greater than the others, and these two pillars being the furthest from the robot (see figure 7). This is because as the top of the pillar gets closer to the centre of the mirror, the horizontal resolution decreases. This makes being sure that the nearest point on the cone is being matched impossible, which means that two points that are not on the same radial line could be being selected for the disparity measure.

4.2 Robot localisation

The accuracy of the robot's localisation depends entirely upon the accuracy of the distance measured. By us-

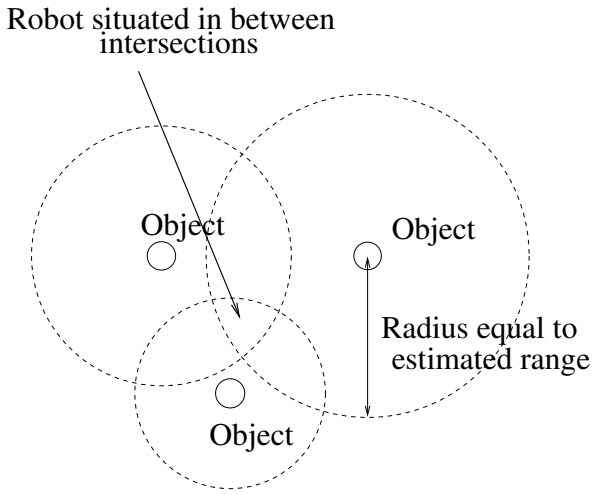

Figure 10: Circles are drawn around three objects, and the robot lies in the inner triangle of intersections. The centre of the triangle is recorded as the position of the robot.

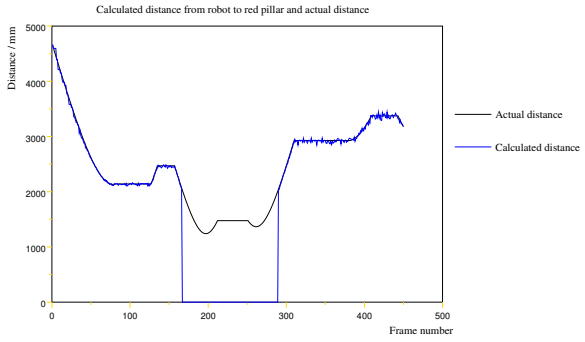

Figure 11: Graph of actual distance to red pillar, and computed distance.

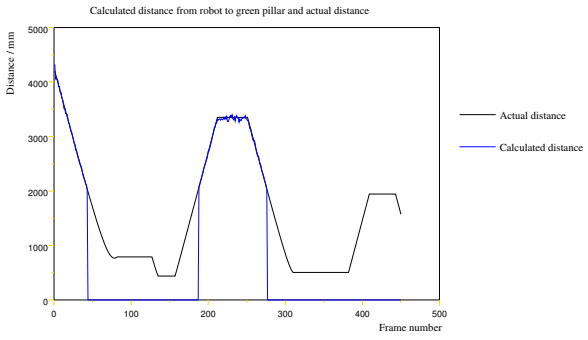

Figure 12: Graph of actual distance to green pillar, and computed distance.

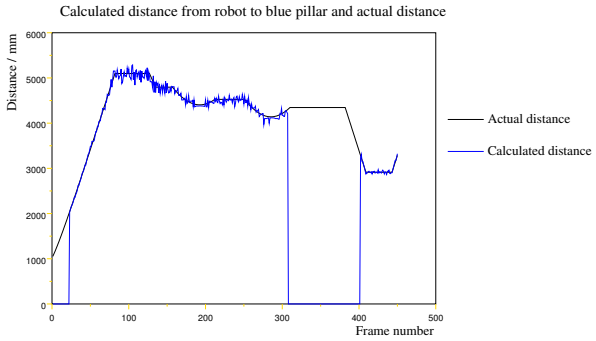

Figure 13: Graph of actual distance to blue pillar, and computed distance.

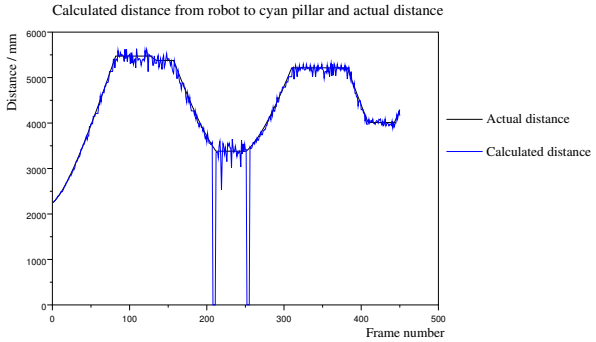

Figure 14: Graph of actual distance to cyan pillar, and computed distance.

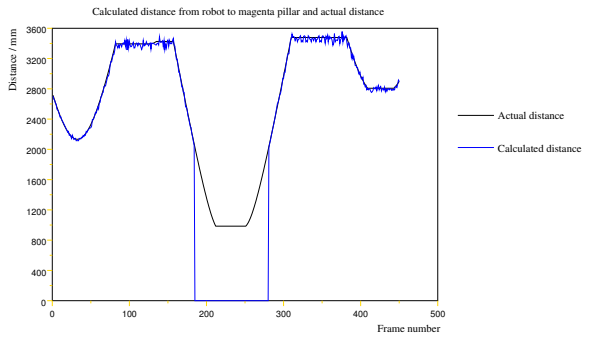

Figure 15: Graph of actual distance to magenta pillar, and computed distance.

ing the nearest objects, and therefore the most accurate measurements, the robot was able to localise to within 130mm of its actual position, with an average error of 29.97mm.

The graph in figure 17 shows the error in the robot's localisation for the first 180 frames. The actual trajectory and the calculated trajectory of the robot are shown in figure 18.

5. Conclusions

The simulator created is a very useful tool for omnidirectional research. It is no longer necessary to be able to afford expensive mirrors, or carefully mount them on a robot in order to create useful algorithms. New ideas and designs can be tested without the time consuming process of getting the hardware first. The experiments performed above have shown the usefulness of simulating computer vision, and in particular, that the omnidirectional images created by the POV-Ray raytracing backend are physicaly accurate. The restriction of processing power has been removed, allowing more emphasis to be placed on what is possible, not what is possible with *today's* hardware.

With some minor extensions to the simulator, robots that are already equiped with omnidirectional vision, such as many RoboCup (www.robocup.org) robots (Lima et al., 2001), can be fully simulated and algorithms tested and developed in software.

The use of omnidirectional stereo vision for range finding is quite clearly a definite possibility. The results show that for close objects the error in distance averages about 2cm. Different angled mirrors will give different ranges. A 120° mirror, for example, will show the ground around the robot rather than objects in the distance. The accuracy achieved is better than that of sonar sensors, and in many situations vision ranging is more favourable than laser range finders as more information is provided – here the object is identified as well as its distance measured, avoiding problems such as perceptual aliasing.

Omnidirectional vision is ideal for robot localisation as objects do not go out of sight when turning. With accurate omnistereo vision, localisation is a simple task of trilateration. The results achieved here show that even with a very basic and limited approach, the use of omnistereo results in reasonable accuracy. Over the 450 frames of video, the localisation of the robot was wrong on average by 29.97mm, which compares favourably to localisation by dead reckoning where the error will accummulate with time. Figure 17 shows that the error in position stays roughly constant.

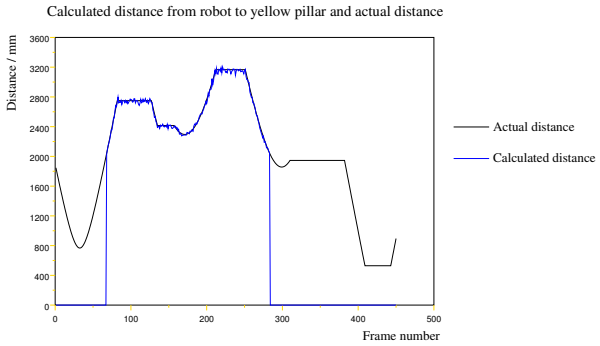

Figure 16: Graph of actual distance to yellow pillar, and computed distance.

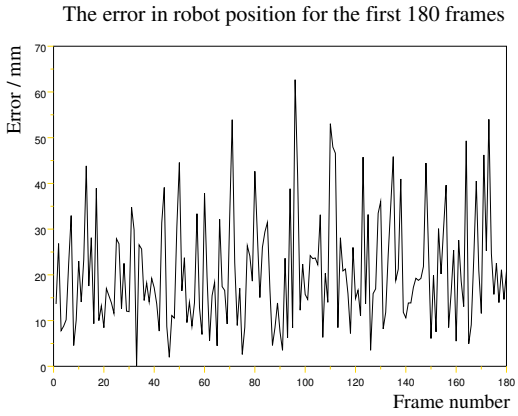

Figure 17: Graph of the error in the predicted position of the robot for the first 180 frames.

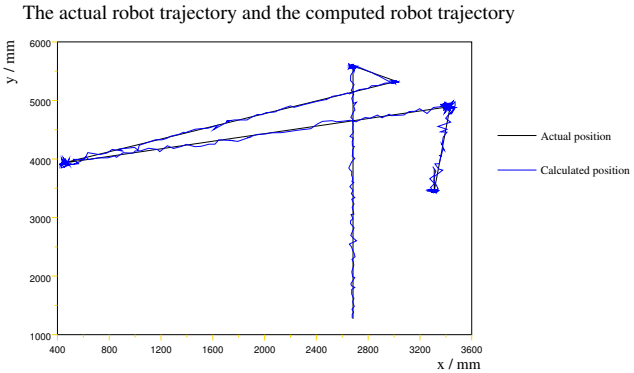

Figure 18: Graph of the error in the predicted position of the robot.

References

Bräunl, T. (2003). *Embedded Robotics: Mobile robot design and applications with embedded systems*. Springer.

Ishiguro, H., Yamamoto, M., and Tsuji, S. (1992). Omni-directional stereo. *IEEE Trans. Pattern Anal. Mach. Intell.*, 14(2):257–262.

Kang, S. B. and Szeliski, R. (1997). 3-d scene data recovery using omnidirectional multibaseline stereo. *Int. J. Comput. Vision*, 25(2):167–183.

Lima, P., Bonarini, A., Machado, C., Marchese, F., Ribeiro, F., and Sorrenti, D. (2001). Omni-directional catadioptric vision for soccer robots. *Robotics and Autonomous Systems*, 36(2-3):87–102.

Michel, O. (2004). Webots: Professional mobile robot simulation. *Journal of Advanced Robotics Systems*, 1(1):39–42.

Shah, S. and Aggarwal, J. K. (1997). Mobile robot navigation and scene modeling using stereo fish-eye lens system. *Mach. Vision Appl.*, 10(4):159–173.

Spacek, L. (2005). A catadioptric sensor with multiple viewpoints. *Robotics and Autonomous Systems*, 51(1):3–15.

Swaminathan, R. and Nayar, S. K. (2000). Non-metric calibration of wide-angle lenses and poly-cameras. *IEEE Trans. Pattern Anal. Mach. Intell.*, 22(10):1172–1178.

Tan, K.-H., Hua, H., and Ahuja, N. (2004). Multiview panoramic cameras using mirror pyramids. *IEEE Trans. Pattern Anal. Mach. Intell.*, 26(7):941–946.