# Visual Sonar: Fast Obstacle Avoidance Using Monocular Vision*

Scott Lenser and Manuela Veloso
{slenser,mmv}@cs.cmu.edu

Carnegie Mellon University, Pittsburgh, USA

**Abstract**

*We contribute a fast system for avoiding unknown obstacles on a mobile robot using a simple camera as the only sensor. The vision module detects objects, both known and unknown, around the robot. Unknown objects are detected by paying attention to occlusions of a floor of known colors. Range and angle to the objects is calculated and used to create a radial model of the robot's vicinity. This modeling component keeps tracks of objects that are currently outside the field of view of the camera allowing the robot to avoid obstacles it is not currently looking at. We show the effectiveness of the vision and modeling algorithms by creating a simple behavior which wanders around while avoiding obstacles.*

## 1 Introduction

General obstacle avoidance can be effectively accomplished with several robot sensors including infrared, sonar, or laser. However this requires that a robot be equipped with such sensors. Instead, we are focused on robots that have vision as their main sensor. For such robots, modeling all possible obstacles for detection and avoidance is not practical. In this paper, we contribute an algorithm that allows for a vision algorithm to behave similarly to a sonar sensor. Hence we call our approach "visual sonar". With visual sonar, the robot is capable of effectively detecting modeled objects in the world while also avoiding any other unmodeled obstacle. The key technical contributions of the visual sonar algorithm consist of vision algorithms to detect obstacles, partial identification of obstacles, and fast algorithms to update a "radial map" of the environment. The robot can then use the radial map for its behaviors, in particular for general obstacle avoidance. The robot computes the open areas without obstacles and then can move towards these open areas. Visual sonar is robust to different backgrounds, the robot being capable of being trained on different backgrounds. Our work is inspired by other vision processing algorithms that reason about the background of an image [5, 7, 10, 14].

In comparison to related work, our approach offers several advantages. Our vision algorithms run extremely quickly, running in time $O(r(w+h)/f)$ where $r$ is the angular resolution desired, $w$ and $h$ are the width and height of the image respectively, and $f$ is the field of view of the camera. We can process a 176 by 144 pixel image in $\approx$ .4ms on a 400MHz MIPS R4000 excluding the segmentation step (which is done by another module). Note that the complete vision update (including segmentation) scales with $\sqrt{s}$ where $s$ is the size of the image with a small constant factor, making it extremely fast. Motion updates take time $O(nr)$ where $r$ is the angular resolution and $n$ is the number of object categories. One object category is required for each set of objects that must be reported at the same angle at the same time. Note that this scale linearly with the resolution. Motion updates take approximately 6ms peak on our processor. Visual sonar takes $\approx 10\%$ of our processor updating for vision at 25Hz and motions at 31.25Hz (these are full rate updates for our hardware).

Our vision routines integrate domain specific identification of detected obstacles when possible. This allows us to selectively ignore objects that we do not wish to avoid (such as stripes on the floor or balls that we wish to manipulate). General obstacle avoidance is still achieved as all non-floor colored objects are added to a general unidentified obstacle category. Selective object identification allows the model to be used for more than just obstacle avoidance as it carries extra information about the types of objects around the robot. It also allows objects to be ignored which cannot be separated using only color using a little domain knowledge to disambiguate between the two classes. This allows us to ignore white stripes on the field while still avoiding the white walls. Our vision segmentation method allows for the robot to correctly avoid obstacles even on multi-colored floors as we can represent multi-model distributions for the floor (we require only that the colors of obstacles be mostly separate from the colors of the floor). Our technique requires only a simple monocular color camera to operate making it applicable to a wide range of inexpensive robots.

Our model representation is an alternative to the popular occupancy grid approach. Our model scales as the square root of the area covered by the model as opposed to linearly for an occupancy grid. The price we pay for this speed up is the inability to represent objects behind another object of the same type. Since these objects are not of interest for large classes

*This research was sponsored by Grant No. DABT63-99-1-0013 and by generous funding by Sony, Inc. The content of this publication does not necessarily reflect the position of the funding agencies and no official endorsement should be inferred.

of tasks (almost any reactive task), this is a small price to pay for increased speed. Our representation allows us to selectively remove outdated data and ignore classes of objects as required by the task.

Others have also used vision to calculate free regions for purposes of robot navigation. None of their vision techniques allow for the selectively ignoring obstacles, allow use of the radial model for other purposes (requires object identification), or process images as quickly. Many techniques require that the floor be a uniform color, unlike our approach. Some require additional vision hardware and this is noted as appropriate. Many of these techniques are based upon the height of object in the world. They are unable to avoid flat obstacles such as grassy areas (when following a pathway) or caution tape on the floor. Some of these groups incorporate models into their systems. All of these models are based on grid representations which have the shortcomings noted above. Each group is compared in more detail below.

The OMNI RoboCup small-size team [10] segments an image based on background color to find free space around the robot using an omni-directional camera. Their algorithm scales with the size of the image and is unable to handle floors consisting of multiple colors or overlap of floor colors. Ulrich and Nourbakhsh [14] use color histograms to segment camera images into clear floor and obstacle regions. They do their filtering in HSI space and therefore require a color transform per pixel. Their algorithm scales with the size of the image. Their algorithm requires 200ms to process a 320 by 260 image on a PII 333MHz. Lorigo et.al. [5] use a reference area at the bottom of the image to create histograms which are compared with other areas of the image using a distance threshold to find obstacles. Their algorithm scales linearly(performing color space conversion) with image size, window size, and histogram bins. Their method doesn't calculate the actual position of objects around the robot and therefore precludes some uses of the model and makes behaviors harder to develop. Ohno et.al. [7] have created a system for following paths using color vision. Their segmentation method is based off using minimum Mahalonobis distance from clusters in YIQ color space. There representation requires bimodel (two color) distributions for colors of free space and obstacles. They have to calculate 4 Mahalonobis distances per pixel which requires several matrix multiplications at each pixel and thus scales with the size of the image.

José the robot waiter [3] uses a vision based approach to obstacle avoidance based upon a trinocular camera system for calculating depth information. Obviously, this technique requires more cameras and processing which adds to cost. Burschkal, Lee, and Hager [2] use stereo disparity for creating a distance map from stereo images. This requires stereo cameras and textured images for the disparity calculation. Their technique runs at 11.2Hz for vision alone on a PIII 850MHz using megapixel cameras.

Pears and Liang [8] use corner tracking and image homographies to find coplanar corners to find the ground plane. They use the ground plane regions found to seed a color based region growing method which leaves small false obstacle regions which must be post processed. Their technique scales with the size of the image (corner finding) and number of corner calculations performed. Rasmussen [9] created a algorithm for road detection. This computes many image features and uses a neural network to combine features. Speed should be slow given the amount of processing being done.

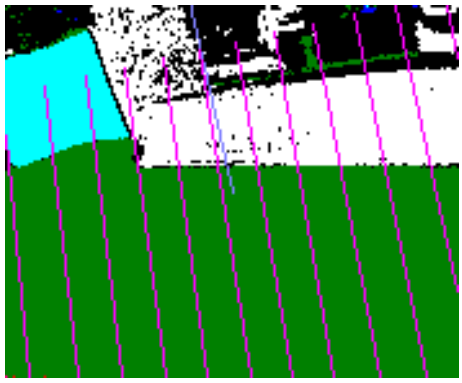

***Figure 1:*** *Radial vision image with scan lines.*

Many vision researchers have developed systems to separate moving objects from a stationary background in video sequences (e.g. [4, 6, 11, 13]). These systems do not address the full obstacle detection problem since they are incapable of detecting stationary obstacles. Obstacle avoidance using visual sonar results in a navigation pattern of the robot to the open spaces of the environment. Visual sonar has then the potential to be applied to path planning that aims at largely covering a space (as in [12]) by guiding the robot to avoid previously visited regions.

The paper is organized as follows. Section 2 presents the vision algorithms. Section 3 describes the representation of the world model. Section 4 shows how the radial map is used for general obstacle avoidance, while being able to not avoid any modeled objects if needed, such as the lines on the field and the ball. Our visual sonar approach is fully implemented in our Sony ERS-210 (AIBO) robots. Although the best proof of the results is through actual life demonstrations, we present an illustrative sequence of images of the performance of obstacle avoidance. Section 5 concludes the paper.

## 2 Vision

The vision component is responsible for detecting objects present along different angles around the robot. The vision processing consists of several discrete stages. The first stage takes the raw camera image and classifies each pixel into one of several color classes. Pixels corresponding to the floor are classified into the "floor" class. Pixels of other colors are classified into either one of the color classes for various objects or into the "unknown" class for general obstacles. Scan lines are then drawn in the image that correspond to lines on the ground plane emanating from the reference point for the

robot. Figure 1 shows an example image with the scan lines drawn over the image. We used scan lines spaced every $5^\circ$ degrees around the robot. The classified pixels on each scan line are then run length encoded. Objects are located along each scan line and identified if possible. Most of the run time is spent classifying the image which does one table lookup per pixel. Other parts of the processing contribute negligible additional time. The algorithm does an excellent job of locating objects and identifying them when possible. Figure 2 gives pseudo-code for the complete algorithm.

**Segmentation:** We segment the image into color classes using CMVision 2, a real time color vision library [1]. CMVision uses a lookup table(threshold map) to perform the mapping from YUV pixel values to symbolic color class. We use 4 bits of Y and bits of U and V to make a 16 bit index into the lookup table. Each entry in the table has the symbolic color class that pixels of that image pixel color should be assigned. Note that each symbolic color class may correspond to several physical colors, e.g. a "floor" color class which includes all the colors found on the floor. The threshold map is trained by taking images with the camera and hand labelling the symbolic color classes to be used for that image. The color classes we use include a color class which corresponds to unknown colors. We take advantage of this class to identify obstacles of unknown colors which are known to not be of the same color as the floor (the "floor" color class). The camera and hand labelled images define a supervised classification problem. We take an example driven approach to this problem.

Our solution is to take each example pixel in YUV space and spread it using geometric decay in all directions of the full YUV space. The geometric decay is done in Manhattan distance which allows for an efficient implementation using a fill stage followed by a constant number of sweeps across the threshold map. We use a geometric decay rate of .5 which decays rapidly with distance. For each box in the threshold map, the total example weight due to each color class is calculated. Each color class has an associated confidence threshold. If the proportion of weight due to a particular color class compared to the total weight in this threshold box is greater than the confidence threshold of the color class, the threshold box is labelled with this symbolic color. If no color class satisfies these conditions, the threshold box is labelled with the "unknown" color class.

**Object detection:** The vision identifies the following object types: *wall*, *stripe*, *unknown_obstacle*, *red_robot*, *blue_robot*, *ball*, *cyan_goal*, and *yellow_goal*. All of the objects are identified using the following heuristic: the object is the first continuous set of pixels in the scan line composed completely of color classes that are "ok" for the object and which has $k$ more pixels of the "best" color class for this object than all other color classes. The value of $k$ reflects the amount of noise present in the image; we use a value of 5. Since *wall* and *stripe* are both composed of the same color (white), an additional test is needed to disambiguate when a *wall*/*stripe*

---

**Procedure** Vision($v$:vision)
  Project corners of image into egocentric coordinates.
  Calculate angle of each projected corner.
  **let** *min_angle* $\leftarrow$ Minimum angle of corners.
  **let** *max_angle* $\leftarrow$ Maximum angle of corners.
  Snap *min_angle* and *max_angle* to one of the
    fixed scan angles.
  **for** $\alpha \leftarrow$ *min_angle* **to** *max_angle*
    Find the line on the image $l$ which corresponds to
      a line on the ground at angle $\alpha$.
    Scan through the segmented image along $l$ creating
      a run length encoded version of the line $r$.
      Store the start/end screen coordinates of each run.
    IdentifyObjects($v$,AngleIdx($\alpha$),$r$)

---

***Figure 2:*** *Radial vision*

is detected. The object is considered a *wall* only if it is at least 50mm wide and the number of white pixels on the *wall* is at least as great as the number of green pixels after the *wall*. The first filter makes sure the wall could not just be a *stripe*. The second filter is due to our use of a green carpet as our floor which does not extend much beyond the white walls. These filters reliably distinguish between *wall* which must be avoided and *stripe* which can be transversed safely. An *unknown_obstacle* is detected whenever enough pixels belonging to the "unknown" class occur together. Since one of the color classes is "floor", this corresponds to occlusions of the floor by other objects. These objects get labels as *unknown_obstacles* in this manner. Because the bodies of robots are also of unknown color, *unknown_obstacles* often occur for us due to robots. We peek ahead along this scan line to see if this obstacle can actually be identified and if so move this obstacle into the *red_robot* or *blue_robot* class as appropriate.

Distance to objects is calculated by intersecting rays through the closest pixels of the object on the image onto the ground plane the robot is standing on. This generates very accurate distances for objects that are close and noisy estimates for more distant objects. Because only the closer objects are useful for obstacle avoidance, distances to the objects of interest are all quite accurate. Some distances can be slightly high due to parts of the object being off of the ground. This also is not a problem because the estimates improve somewhat as the robot approaches. Since objects in the air must be supported by something, the supports are usually also visible and since they are resting on the ground, they will have accurate distance estimates. So for all objects of interest for obstacle avoidance, accurate distance estimates are available.

## 3 Model

Since the camera has a limited field of view ($55^\circ$ in the case of our robot), it is necessary to keep a model of the local environment to ensure good obstacle avoidance performance. The purpose of the model is to remember objects that the robot has seen recently that must be avoided and are currently outside

```
Procedure ModelVisionUpdate(m:model,v:vision,time)
  for α:angle ← 0 to NumAngles − 1
    if ¬ SeeAngle(v,α)
      continue
    let visible_start ← ClosestVisiblePt(v,α)
    let visible_end ← FarthestVisiblePt(v,α)
    for i:object ← 0 to NumObjectTypes − 1
      if m^i_α.valid ∧ (|m^i_α.x⃗| < visible_start)
        continue
      if visible_start <= |m^i_α.x⃗| <= visible_end
        m^i_α.valid ← false
      if v^i_α.valid ∧ ((¬m^i_α.valid) ∨ (|v^i_α.x⃗| <= |m^i_α.x⃗|)
        m^i_α.x⃗ ← v^i_α.x⃗
        m^i_α.valid ← true
        m^i_α.last_seen ← time
```

***Figure 3:*** *Model vision update. $m^i_\alpha$ refers to the $i^{th}$ model object at the $\alpha^{th}$ angle index. $m^i_\alpha.\vec{x}$ refers to the point associated with this angle and object. Visual information is shown using the same syntax.*

the field of view of the camera. The model is represented radially to be similar to the visual information. Each object type in the model is represented separately using its own radial map. The model for each object type is divided into a set of pie shaped slices that each cover a range of contiguous angles. Only the closest object of each object type is kept in order to conserve memory. Objects of the same type which are farther away are not of interest for obstacle avoidance and usually not interesting in general. Each pie slice for an object type keeps track of the closest object of that type seen recently. The model is updated for both new visual information and the movement of the robot.

Vision updates are handled by simply replacing the visible part of the model with the new visual information. We are careful to keep objects which are too close to be seen with the current position of the camera as these objects are critical for obstacle avoidance. Objects which should have been seen but weren't are removed from the model as the camera rescans the area. The algorithm is detailed in Figure 3.

The motion update updates the model for the motion of the robot. Each angle of each object has an associated point in Cartesian coordinates which is the egocentric coordinate of the location at which the object was seen. The motion update proceeds by moving each of these points counter to the motion of the robot. The points are then copied over into a new radial model. The connectivity of the points is considered while performing this update to ensure that different parts of objects don't fragment into disconnected components. Two points are considered connected during the update if they are of the same type and in neighboring angles. Connectivity must be considered because as the robot moves closer to an object, more resolution in angle is needed to represent an object of the same size. We maintain connectivity by creating

```
Procedure ModelMotionUpdate(m:model,u:update,time)
  let m_new ← m
  for i:object ← 0 to NumObjectTypes − 1
    ModelMotionUpdate(m,m_new,u,i,time)
  m ← m_new
Procedure ModelMotionUpdate(m_old,m_new:model,
            u:update,i:object,time)
  for θ:angle ← 0 to NumAngles − 1
    m_new^i_θ.valid ← false
    m_old^i_α.valid ← m_old^i_α.valid ∧
      (time − m_old^i_α.last_seen <= TooOldTime)
    m_old^i_α.x⃗ ← Apply(u,m_old^i_α.x⃗)
  for α:angle ← 0 to NumAngles − 1
    if ¬ m_old^i_α.valid continue
    let p⃗ ← m_old^i_α.x⃗,   θ ← p⃗.Angle()
    if ¬m_new^i_θ.valid ∨ (|p⃗| <= |m_new^i_θ.x⃗|)
      m_new^i_θ ← m_old^i_α
    let β ← (α + 1) mod NumAngles
    let next_valid ← m_old^i_β.valid
    if next_valid
      let q⃗ ← m_old^i_β.x⃗,   φ ← q⃗.Angle()
      if θ=φ continue
      FillInMissing(m_old,m_new,i,θ,φ,Line(p⃗, q⃗),
        Min(m_old^i_α.last_seen,m_old^i_β.last_seen))
Procedure FillInMissing(m_old,m_new,i,θ,φ,line,last_seen)
  for ν ← (θ + 1) to φ mod NumAngles
    if Intersects(line,ν)
      let x⃗ ← Intersect(line,ν)
      if ¬m_new^i_ν.valid ∨ (|x⃗| <= |m_new^i_ν.x⃗|)
        m_new^i_ν.valid ← true
        m_new^i_ν.x⃗ ← x⃗
        m_new^i_ν.last_seen ← last_seen
```

***Figure 4:*** *Model motion update. $m^i_\alpha$ refers to the $i^{th}$ model object at the $\alpha^{th}$ angle index. $m^i_\alpha.\vec{x}$ refers to the point associated with this angle and object.*

new points on the line between the two old points if necessary to fill in missing angles. Old observations must eventually be removed to avoid the unbounded accumulation of motion error. We take the simple approach of simply throwing out observations that are older than a timeout during the motion update. The motion update algorithm is detailed in Figure 4.

Figure 5 shows an example of a motion update that requires connectivity maintenance. The black(solid) points represent the location of points in the radial model before the motion update. The black(solid) lines represent the logical connectivity of points inferred by the motion update algorithm. The blue(dashed) points represents the points updated for the forward/leftward motion of the robot. The blue(dashed) lines represent the desired logical connectivity of the points. Notice that the angle in front of the robot has no blue(dashed) point. This is fixed by intersecting the blue(dashed) line with the bold line which bisects the angle and adding a new point at this location (shown in bold).

We have shown how to create a radial model which remem-

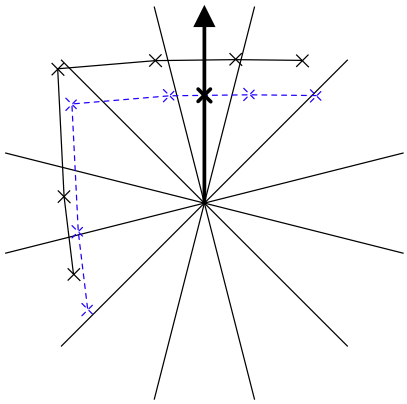

***Figure 5:*** *Motion update example. Divisions between angles are shown in black, old points in black(solid), points after motion update in blue(dashed), and points due to connectivity maintenance in bold. See text for details.*

bers objects which cannot currently be seen. This model can be used in the same way that sonar sensors are used. It provides local range estimates around the robot of all pertinent objects. It also provides some ability to identify and track different types of objects. This allows the robot to be selective about what it avoids. By using this feature, we are able to walk over stripes and through balls while still avoiding other obstacles including ones that look quite similar visually such as walls. See Figure 6 for an example of the output of the radial model. See Figure 7 for pictures of the domain from which this model was generated.

## 4 Behaviors

We tested the visual sonar algorithm by implementing a basic behavior that wanders around while avoiding obstacles. The robot walks fast ($\approx$ .2m/sec) most of the time. As the space around the robot gets more crowded the robot slows down to give itself more time to react. The behavior ignores obstacles farther than **MaxAvoidDist**. If the robot gets too close to obstacles (**StopAvoidDist**), it will back up some. The robot tries to walk straight. If obstacles are in the way, the robot turns towards free space and away from obstacles. This is done using a simple angular potential function in which objects repel the robot angularly.

The behavior bases all of its calculations on the minimum distance of an object in the set of avoid object types at each angle. This distance is then bounded between 0 and **MaxAvoidDist**. A weighted distance $avg\_dist$ to obstacles in front of the robot is calculated where each angle is weighted using a Gaussian centered on straight ahead. The fraction of full forward speed to use is calculated as $(avg\_dist - \textbf{StopAvoidDist})/(\textbf{MaxAvoidDist} - \textbf{StopAvoidDist})$. Any remaining fraction is available for turning. The direction and amount to turn is calculated based upon a penalty term for left versus right. A turn direction is penalized by the total amount by which obstacles to that side of the robot are closer

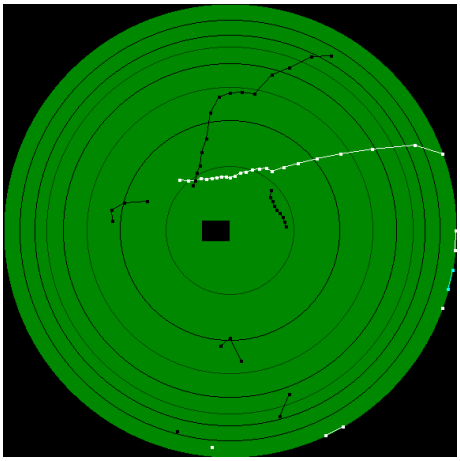

***Figure 6:*** *Radial model while avoiding obstacles. The black box represents the robot which is facing to the right of the image. The black circles near the robot are .5m meter marks. The white line to the left of the robot is a wall of the field. The black line in front of the robot represents a hammer that the robot is in the process of avoiding. The clusters of black dots to the right of the robot represents a lid and a satchel seen earlier.*

than **MaxAvoidDist**. The turn direction is then calculated as $(right\_penalty - left\_penalty)/(right\_penalty + left\_penalty)$. This quantity will be between -1 and 1 and represents the direction of turn and the amount of the available movement fraction available for turning to use. Despite the simplicity of the algorithm, it successfully avoids obstacles while wandering around an area.

## 5 Results

As with most complete systems designed to perform a task, it is difficult to produce quantitative results. We tested the system extensively using a variety of different colored and shaped objects. The system was able to wander around without hitting the obstacles as long as two conditions are met: the objects have a reasonable fraction of colors that are not present in the flooring and the robot actually looked at the obstacle. The system was also able to avoid obstacles that appeared suddenly within close range if the obstacle was seen before the collision occurred. The robot spent most of its time during tests walking quickly forward or veering slightly, only occasionally slowing down to turn away from a particularly close obstacle (usually one that suddenly appeared). A typical path taken by the robot is shown in Figure 7. As you can see in the figure, the robot tends to randomly cover large sections of the environment and can successfully avoid obstacles even in cluttered environments. We have shown how to avoid obstacles using only a simple monocular camera and a fraction of the available CPU, freeing CPU to be used on other tasks and algorithms while still avoiding obstacles.

**Acknowledgements:** We would like to thank James Bruce for many useful suggestions related to this work.

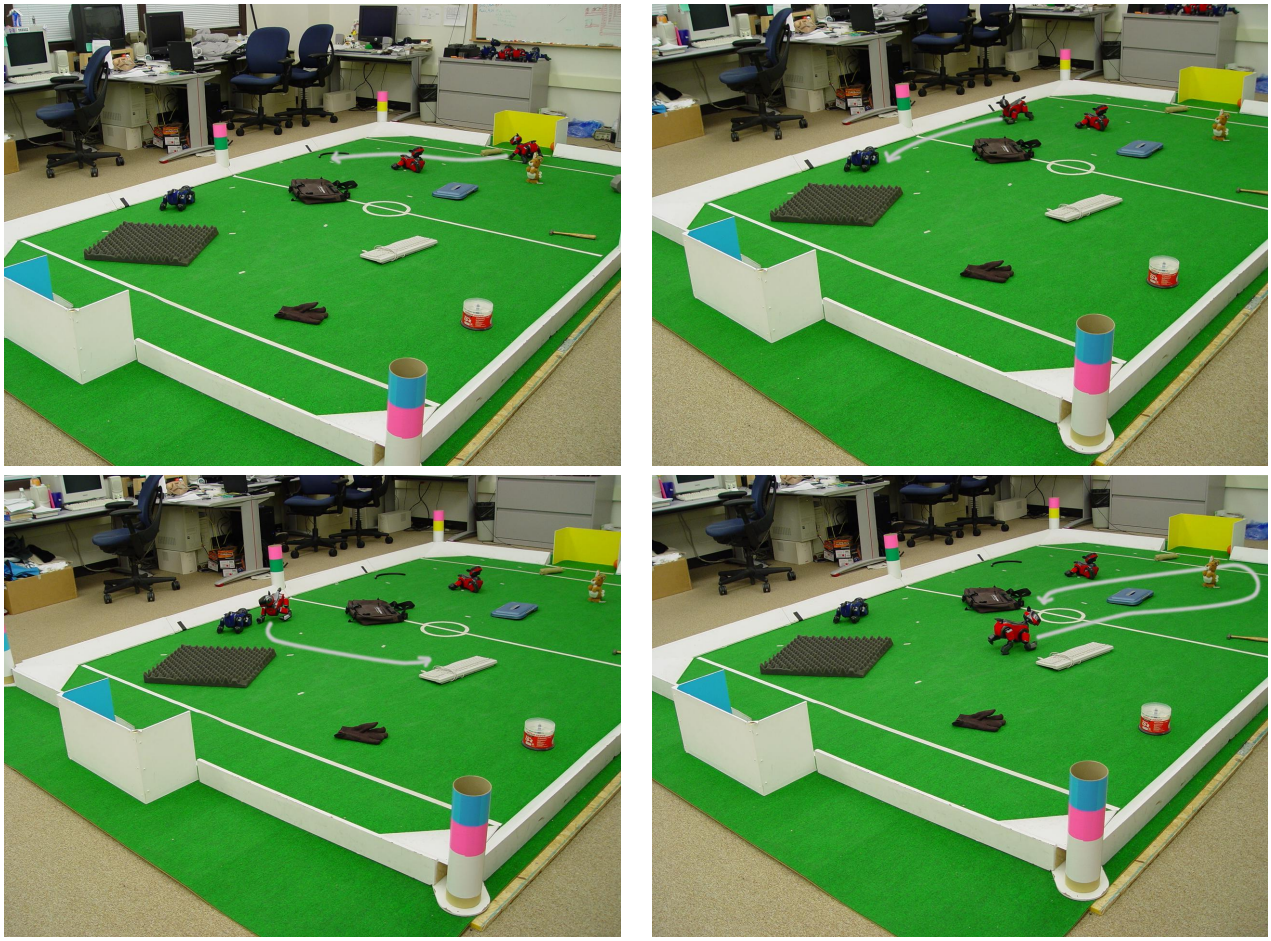

***Figure 7:*** *Example path taken by robot. The path of the robot is shown with the white line with the arrow at the end. This path took approximately 50secs to execute. The arena is 4.2m by 2.7m and the robot is 20cm long. The robot successfully avoided an umbrella, a rubber hose, several robots, packing foam, a keyboard, an IROS bag plus a few miscellaneous objects in the vicinity of the robots path.*

## References


[1] J. Bruce, T. Balch, and M. Veloso. Fast and inexpensive color image segmentation for interactive robots. In *Proceedings of IROS-2000*, 2000.

[2] D. Burschkal, S. Lee, and G. Hager. Stereo-based obstacle avoidance in indoor environments with active sensor re-calibration. In *ICRA 2002*, volume 2, pages 2066–2072, 2002.

[3] P. Elinas, J. Hoey, D. Lahey, et al. Waiting with José, a vision-based mobile robot. In *ICRA 2002*, volume 4, pages 3698–3705, 2002.

[4] S. Jehan-Besson, M. Barlaud, and G. Aubert. Video object segmentation using Eulerian region-based active contours. In *Proceedings of IEEE International Conference on Computer Vision (ICCV)*, volume 1, pages 353–360, 2001.

[5] L. Lorigo, R. Brooks, and W. Grimsou. Visually-guided obstacle avoidance in unstructured environments. In *In IROS 1997*, volume 1, pages 373–379, 1997.

[6] J. M. F. Moura, R. S. Jasinschi, H. Shiojiri, and J.-C. Lin. Video over wireless. *IEEE Personal Communications*, 6(1):44–54, February 1996.

[7] K. Ohno, T. Tsubouchi, S. Maeyama, and S. Yuta. A mobile robot campus walkway following with daylight-change-proof walkway color image segmentation. In *IROS 2001*, volume 1, pages 77–83, 2001.

[8] N. Pears and B. Liang. Ground plane segmentation for mobile robot visual navigation. In *IROS 2001*, volume 3, pages 1513–1518, 2001.

[9] C. Rasmussen. Combining laser range, color, and texture cues for autonomous road following. In *ICRA 2002*, volume 4, pages 4320–4325, 2002.

[10] D. Sekimori, T. Usui, Y. Masutani, and F. Miyazaki. High-speed obstacle avoidance and self-localization for mobile robots based on omnidirectional imaging of the floor region. In *Proceedings of RoboCup 2001 International Symposium*, Seattle, USA, August 2001.

[11] B. Stenger, V. Ramesh, N. Paragios, F. Coetzee, and J. Buhmann. Topology free hidden markov models: application to background modeling. In *Proceedings of IEEE International Conference on Computer Vision (ICCV)*, volume 1, pages 294–301, 2001.

[12] J. Svennebring and S. Koenig. Building terrain-covering ant robots. Technical Report GIT-COGSCI-2002/10, College of Computing, Georgia Institute of Technology, Atlanta, Georgia, USA, 2002.

[13] Y. Tsaig and A. Averbuch. A region-based MRF model for unsupervised segmentation of moving objects in image sequences. In *Proceedings of IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 1, pages I889–I896, 2001.

[14] I. Ulrich and I. Nourbakhsh. Appearance-based obstacle detection with monocular color vision. In *Proceedings of AAAI National Conference on Artificial Intelligence*, Austin, TX, USA, 2000.