A Robot Golfing System Using Binocular Stereo Vision.*

Roger W. Webster, Ph.D.
Robot Vision and Artificial Intelligence Laboratory
Department of Computer Science
Millersville University
Millersville, PA 17551
(717) 872-3539

Yong Wei
Department of Mechanical Engineering
Northeast University of Technology
Shenyang, Liaoning Province
P.R. China

Abstract

This paper describes the work-in-progress of a robot vision golfing system. The Automated Robotic Navigational unit with Intelligent Eye and Putter (ARNIE P)τ project was initiated to investigate the problems and develop software solutions for robotic tasks that require good hand-eye coordination and an intelligent sensor feedback mechanism. This system has only one frame buffer and no specialized hardware, so quasi-real time 3D tracking is accomplished in software using the Unix Spline facility. The single frame buffer and digitizer, stores and retains the location of the ball from two separate cameras during the time interval between the golf ball initially crossing a trigger scan line and the ball coming to a complete stop. The most novel aspect of this study is that by attempting to build or model a difficult perceptory task such as golf, which requires integrating many complicated computational pieces (binocular stereo vision, robot arm motion, heuristic feedback, learning), it appears to be a good platform to experiment with artificial intelligence techniques and robotics.

1. Introduction.

The objective of this project is to investigate the use of AI techniques for robotic tasks that require good hand-eye coordination and intelligent feedback techniques. The reason for choosing robot golf is that:

- Golf is difficult for humans. It requires many hours, sometimes years of practice to be proficient at it, therefore, it is perceived by lay people as non-trivial for a robot system to accomplish.
- It requires a clever algorithm or methodology for hand-eye coordination.
- It requires an intelligent feedback mechanism (using binocular stereo vision) or *learning* in order to improve its performance.
- It is a constrained environment in which to experiment with the methodologies and techniques of artificial intelligence with robotic applications.

2. How a Robot Putts a Golf Ball.

When a human putts a golf ball there are three basic parameters which determine the outcome of the putt: Speed, Pitch, and Angle. The *speed* of the club, how far back the club is *pitched* before starting the descent towards the ball, and the *angle* of the swing (the angle at which contact is made with the ball) all determine where the ball is going to go. If the speed is too fast, the ball can overshoot its destination, the cup. If the speed is correct but the pitch is too far back, the ball can again overshoot the cup. If the angle that the club makes contact with the ball is not orthonormal to the cup (if the putt is a straight shot), the ball will miss the cup by an angle θ, which can be computed from the contact angle. A robot can be programmed to manipulate and coordinate all of these parameters in order to successfully putt a golf ball into the cup.

In order to compute the effect of these parameters the following data was collected to perform a *calibration* of the robot putting software. What was needed was to compute the distance the ball would be putted, as a function of the *speed* and *pitch* of the club. The contact angle, for now, is set orthonormal to the cup. From this data an equation was derived which determined the proper speed and pitch of the club in order to putt the ball a given distance.

The robot can be programmed at six different speeds [247...252]. Ten putts were recorded at each of the 6 speeds, using 14 different club pitches measured in motor steps [20..280] (see Figure 1). A pitch of over 280 brought the club too far back and into the mobile base of the robot. The ball used for the data in Figure 2 was a *Titleist-4*tm, the putter was semi-permanently affixed to the robot gripper. It is very interesting to note that different balls and different putters do, in fact, perform differently. For each trial, the distance from the robot base to the ball's final destination was measured. The average distance as a function of speed and pitch is shown in Figure 2.

After all the data had been collected, regression tests were done with MATH-CADtm software using a modified template supplied by MATH-CAD. The regressions were done to selected portions of the curves, the slope reversals were eliminated. Based on the range of expected distances required, one of the six equations of motion can be selected. One equation for each speed. For example, the equation most often selected based on the range and repeatability for our purposes is:

$$\text{Speed} = 252 \text{ (the highest speed):}$$
$$\text{Pitch} = 21.865 + (0.116 * D) + (0.001 * D^2)$$

where D is the distance to the cup. The distance to the cup is computed by binocular stereo vision. This is usable over a range of 0 to 339 inches, which is the length of our putting green in the lab. Thus, given the distance we wish to putt the golf ball (computed by binocular stereo vision), the Pitch is computed and using Speed = 252, the robot putts the ball towards the cup.

* This work is based upon a project supported by the National Science Foundation (NSF) under Grant No. USE-9050371 and the Faculty Grants Committee of Millersville University.

τ Nicknamed after Arnold Palmer, one of golf's best ever, on the putting green.

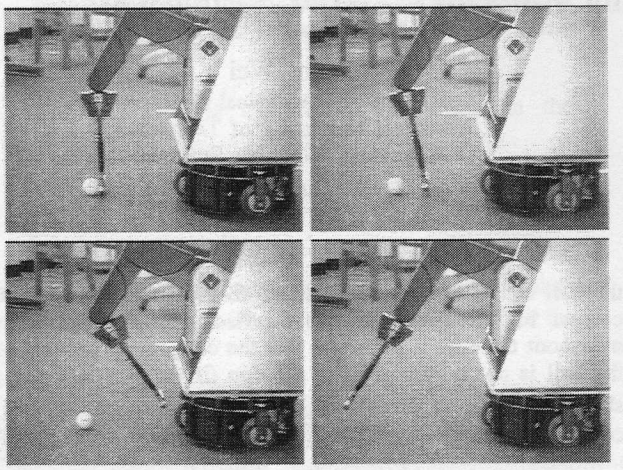

Figure 1. (upper left) is normal approach, (upper right) Pitch=60 motor steps, (lower left) Pitch=200 motor steps, (lower right) Putter hitting through the ball.

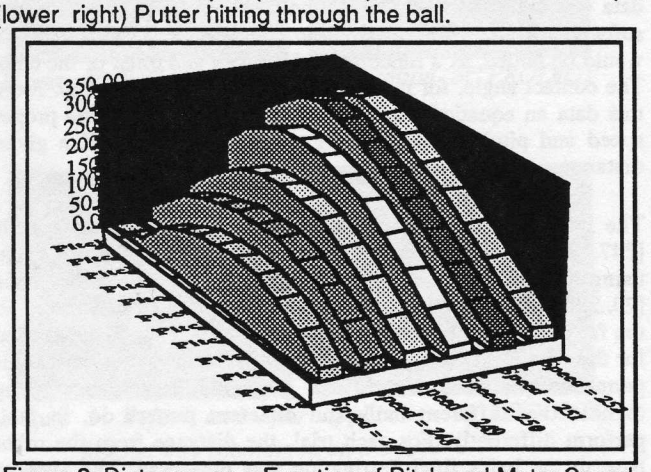

Figure 2. Distance as a Function of Pitch and Motor Speed.

3. Hardware and Software.

The Real World Industries (RWI) B12 mobile base is used as the foundation unit. This 3-wheel synchro drive base can move at speeds up to 2 meters per second, carry up to 20 kg., and can be programmed to move via a standard RS232 serial port. The Microbot Alpha-II six axis, stepper motor driven robot is the actual golfing unit. A conventional golf putter was cut and bolted to the existing two-fingered gripper. The SUN SparcStation^{tm} 330 with 16 MB is the main controller (see Figure 3). Two DAGE-MITI CCD high resolution cameras each with a 12.5mm lens, (six inches apart), are used as the inputs to the binocular stereo vision algorithm. The Imaging Technology Inc. FG100 board is used as the image processor which provides 640 X 480 X (12 bits deep) resolution. A Texas Instruments speech synthesizer is used to speak the results and heuristic adjustments.

Two software drivers were written, one for the Alpha-II robot and the other for the RWI B12 mobile base. The ITEX100^{tm} software calls to the FG100 imaging board was used by the computer vision algorithms. All drivers are written in C and the application code is written in both C and Quintus Prolog.

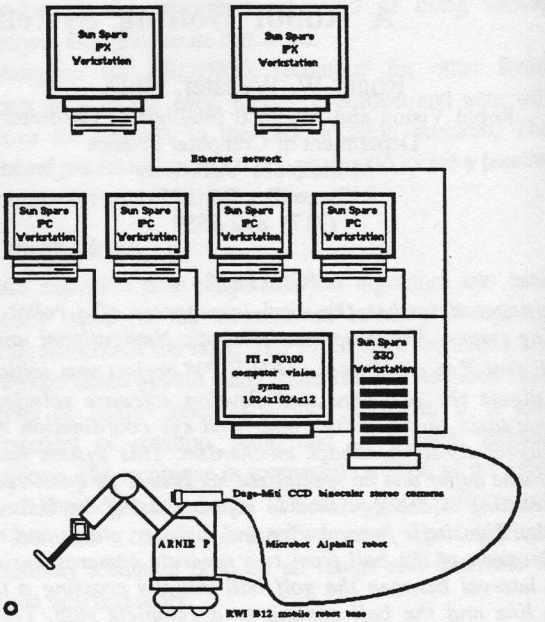

Figure 3. Hardware Configuration.

4. Camera Model for Binocular Stereo Vision.

The following perspective transform equations are essentially those originally described by Yakimovsky and Cunningham [1] and further explained by Kak in [2] and Thompson in [3]. We use Kak's notation [see 2]. The three dimensional location of the camera relative to the robot base is given by the coordinates of its len's focal center as shown in Figure 4. The coordinates of the focal center are the Cartesian components of vector C. The vector $\mathbf{C} = [X,Y,Z]$ emanates from the robot base which is the origin (0,0,0) of our coordinate system. The focal center is at a distance f, the focal length of the lens, (in our case 12.5mm) from the image plane along the optic axis.

The orientation of the camera is denoted by the unit vector **a**, which is the normal vector of the image plane. Two additional unit vectors **h** and **v**, orthogonal to each other and to the vector **a** are used. All the points in the image plane, in terms of **h**, **v** and **a**, are completely described by:

$$\mathbf{C} - f\mathbf{a} + s\mathbf{h} + t\mathbf{v} \quad (1)$$

for different values of s and t. The ordered pair (s,t) are the coordinates of a point in the image plane (measure in millimeters), also $\mathbf{C} - f\mathbf{a}$ is the position vector for the center of the image plane at image coordinates (Io, Jo).

The perspective transformation equations can compute the image coordinates (s,t) for any given object point **P**. To derive these equations, we borrow a technique from Perspective Geometry namely comparing the appropriate similar triangles. The results are

$$s/f = \mathbf{D}\cdot\mathbf{h} / \mathbf{D}\cdot\mathbf{a} \qquad t/f = \mathbf{D}\cdot\mathbf{v} / \mathbf{D}\cdot\mathbf{a} \qquad (2)$$

where $\mathbf{D}\cdot\mathbf{h}$, $\mathbf{D}\cdot\mathbf{a}$ and $\mathbf{D}\cdot\mathbf{v}$ are the vector dot products, as shown in the Figure 4. The vector from the camera's focal center to the object point is **D**. This can be computed as $\mathbf{D} = \mathbf{P} - \mathbf{C}$, given the object point **P** and the location of the camera's focal center **C**.

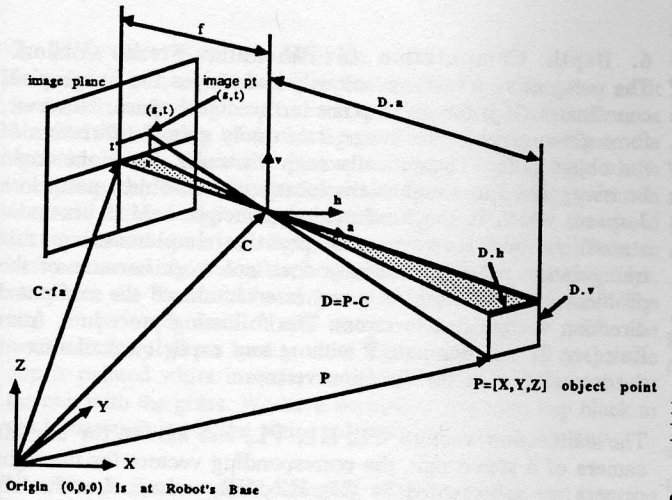

Figure 4. Vectors C, P, and D with respect to the robot base real world coordinate system. (Figure recreated from Kak [2]).

The two equations (1) and (2) are sufficient to determine the image point for any given object point. These equations must be modified when computing points in a digitized image. Instead of using the analog coordinates s and t, which are measured in mm, the following relationship for the digitized versions I and J is used:

$$s = (I - I_o)\,\Delta s \quad \text{and} \quad t = (J - J_o)\,\Delta t \qquad (3)$$

where Δs and Δt are measured in mm. In this system the imaging contact area in the Dage/Miti CCD camera is 8.8mm x 6.6mm, therefore $\Delta s = 640 / 8.8$ and $\Delta t = 480 / 6.6$. Note that in this case $\Delta s = \Delta t$.

In our system, $I_o = 320$, $J_o = 240$ because the frame buffer's video resolution is 640 X 480. The row or horizontal index I and the column or vertical index J are measured from the upper left hand corner of the image frame. The assumption is made that the pixel with indexes (I_o, J_o) coincides with the center of the (s,t) coordinate plane. Substituting Equation 3 into Equation 2, we get:

$$I - I_o = (f \cdot \mathbf{D} \cdot \mathbf{h}) / (\Delta s \cdot \mathbf{D} \cdot \mathbf{a}) \qquad (4)$$

$$J - J_o = (f \cdot \mathbf{D} \cdot \mathbf{v}) / (\Delta t \cdot \mathbf{D} \cdot \mathbf{a}) \qquad (5)$$

These equations can be expressed as:

$$I = (\mathbf{D} \cdot \mathbf{H}) / (\mathbf{D} \cdot \mathbf{a}) \quad \text{and} \quad J = (\mathbf{D} \cdot \mathbf{V}) / (\mathbf{D} \cdot \mathbf{a}) \qquad (6)$$

where:

$$\mathbf{H} = (f / \Delta s) \cdot \mathbf{h} + I_o \cdot \mathbf{a} \quad \text{and} \quad \mathbf{V} = (f / \Delta t) \cdot \mathbf{v} + J_o \cdot \mathbf{a} \qquad (7)$$

5. Determining the Focal Center of Camera Lens.

The objective of the camera calibration procedure is to compute the vectors **C**, **a**, **H** and **V**, thereby determining where the camera's focal center is located relative to the robot base (0,0,0). These represent 11 unknowns because the magnitude of a is unity. In theory, the calibration can be accomplished by showing to the camera at least 6 points whose three dimensional coordinates [X,Y,Z] are known and then setting up at least 12 equations for the 11 unknowns. In practice, however, it is prudent to show the camera more than 10 non-coplanar points that then lead to more than 20 equations for the unknowns. The procedure, from Kak [see 2], is as follows.

Let the known object points be at locations denoted by position vectors $\mathbf{P}_1, \mathbf{P}_2, \ldots, \mathbf{P}_n$. Each $\mathbf{P} = [X,Y,Z]^T$ point was measured (in mm) from the robot's base (0,0,0), where T denotes the transpose. The corresponding vectors from the camera focal center will be represented by $\mathbf{D}_m$, where

$$\mathbf{D}_m = \mathbf{P}_m - \mathbf{C}$$

for m = 1, 2, ... N. Let (I_m, J_m) represent the image (s,t) coordinates of the *m*th object point. Using the perspective transform equations in Equation 6, we have the following:

$$I_m = (\mathbf{D}_m \cdot \mathbf{H}) / (\mathbf{D}_m \cdot \mathbf{a}) \quad \text{and} \quad J_m = (\mathbf{D}_m \cdot \mathbf{V}) / (\mathbf{D}_m \cdot \mathbf{a}) \qquad (8)$$

for m=1,2,...N. The equations generated by the image indexes I_m can also be formulated as:

$$(\mathbf{D}_m \cdot \mathbf{H} - I_m \mathbf{D}_m \cdot \mathbf{a}) = 0 \qquad \text{for m = 1,2 ,3,...,N}$$

or as:

$$\mathbf{P}_m \cdot \mathbf{H} - \mathbf{C} \cdot \mathbf{H} - I_m \mathbf{P}_m \cdot \mathbf{a} + I_m \mathbf{C} \cdot \mathbf{a} = 0 \qquad (9)$$

where $\mathbf{D}_m = \mathbf{P}_m - \mathbf{C}$. Note that these equations are nonlinear as a function of the unknowns since we have the products of the unknowns appearing on the left hand side. Yakimovsky and Cunningham in [1] propose the following scheme which linearizes the equations. First, declare two more unknowns C_h and C_a as follows:

$$C_h = \mathbf{C} \cdot \mathbf{H} \quad \text{and} \quad C_a = \mathbf{C} \cdot \mathbf{a}$$

Equation (9) then becomes:

$$\mathbf{P}_m \cdot \mathbf{H} - C_h - I_m \mathbf{P}_m \cdot \mathbf{a} + I_m C_a = 0. \qquad (10)$$

Second, express Equation 10 in the following linear form:

$$X_m H_x + Y_m H_y + Z_m H_z - I_m X_m a_x - I_m Y_m a_y - I_m Z_m a_z - C_h + I_m C_a = 0 \qquad (11)$$

where X_m, Y_m, Z_m is the location of the object point: $\mathbf{P}_m = [X_m, Y_m, Z_m]^T$ for m = 1,2,...N. This set of the equations constitutes N linear equations for the 8 unknowns H_x, H_y, H_z, a_x, a_y, a_z, C_h and C_a. An assumption is made that one of the unknowns a_z is equal to 1, in order to generate a solution out of these equations. Of course this now violates the requirement that **a** be a unit vector. However, we can satisfy this requirement by first computing a_y and a_z, and all the other unknowns. All the other unknowns will be linearly dependent on a_x. If we then alter the value of the computed a_x such that |a| is equal to unity, we can then appropriately scale the unknowns later. By setting a_x equal to 1, the linear equations given can be reformulated into the following form:

$$X_m H_x + Y_m H_y + Z_m H_z - I_m Y_m a_y - I_m Z_m a_z - C_h + I_m C_a = I_m X_m \qquad (12)$$

for m = 1,2,...N. Defining a vector **U** of the unknowns expresses these equations in vector-matrix form as follows:

$$\{\mathbf{U}\} = \{H_x, H_y, H_z, a_y, a_z, C_h, C_a\}^T \qquad (13)$$

where T denotes the transpose. The matrix of coefficients in Equation 12, denoted by **K**, are expressible in the following vector-matrix form:

$$[\mathbf{K}]\{\mathbf{U}\} = \{\mathbf{R}\} \qquad (14)$$

where the vector **R** denotes the right hand side in Equation 12:

$$\{\mathbf{R}\} = \{I_1 x_1, I_2 x_2, \ldots, I_n x_n\}^T$$

and x_i is the actual X value (measured in mm) of the object point P_i. We have set N equal to 9 to mitigate digitization and measurement errors.Note that N is greater than the number of unknowns. When N is greater than 7, there is no solution to the set of equations because of measurement errors in the P_m's. Therefore, instead of finding an exact solution to this system of equations, the problem becomes one of finding a "best possible" or "*least squares*" solution. A solution to Equation 13 is provided by generating a set of values for the unknowns parameters which minimize the expression:

$$\{[K]\{U\}-\{R\}\}^T\{[K]\{U\}-\{R\}\} \quad (16)$$

If an exact solution existed, this expression would be zero. It can be shown that the minimizing solution must be a solution of the following normal equations:

$$[K]^T[K]\{U\}=[K]^T\{R\} \quad (17)$$

The system of equations in equation (17) now possesses a unique solution because the matrix $[K]^T[K]$ is 7X7. These 7 equations in 7 unknowns can be solved by Gaussian elimination or the Gauss-Jordan algorithm. We used Gaussian elimination with Pivoting found in [4] and [5] and the Fortran code, which we converted to C, can be found in [6].

After computing the unknowns in Equation 17, the magnitude of the resulting vector **a** is computed. Dividing all the unknowns by the ratio of a_x and |a| normalizes the results and ensures that |a| is equal to unity. This computation gives us the best possible values of the unknowns listed in the column vector U in Equation 13. The rest of the unknowns can be solved by considering the unused set of equations in Equation 8 and solving for the J coordinates of the image points. These equations are then written in the form:

$$\mathbf{P_m \cdot V} - \mathbf{C \cdot V} - J_m\mathbf{P_m \cdot a} + J_m\mathbf{C \cdot a} = 0 \quad (18)$$

Introducing the new variable C_v, eliminates the nonlinearity:

$$C_v = \mathbf{C \cdot V}$$

A linearized version Equation 18, is as follows:

$$X_mV_x + Y_mV_y + Z_mV_z - J_mX_ma_x - J_mY_ma_y - J_mZ_ma_z - C_v + J_mC_a = 0 \quad (19)$$

for m = 1, 2, ..., N. We then solve this set of equations with Gaussian Elimination with Pivoting (with nine equations and four unknowns) as before to yield an optimum solution for C_v and the three components of **V**.

After we have calculated C_h, C_v, and C_a, we can determine the location of the focal center of the camera (given by the components of the vector **C**) by solving

$$\begin{aligned} \mathbf{C \cdot a} &= C_a \\ \mathbf{C \cdot H} &= C_h \\ \mathbf{C \cdot V} &= C_v. \end{aligned} \quad (20)$$

Gaussian elimination is performed on these three equations with three unknowns, yielding the vector **C** which determines the 3D location of the camera's focal center in the robot coordinate frame. Thus, the calibration procedure is completed. The entire process is repeated for the second camera in a binocular stereo vision configuration ([7] and [8]), and in the case of a trinocular stereo vision system, repeated for the third camera [9].

6. Depth Computation in Binocular Stereo Vision.

The perspective transform uniquely determines the image pixel coordinates (i, j) for given point in the object space. However, for a given pixel in the image it can only give the direction of the object point. Theoretically such binocular directions could be triangulated to compute the location of an object point in a 3-space, which is the fundamental principle behind binocular stereo vision. However, in practice implementing this triangulation procedure usually does not work because of the problem of computing an exact intersection of the computed direction vectors due to errors. The following procedure, from Kak [see 2], can compute **P** without any explicit calculation of the intersection of the direction vectors.

The calibration vectors **CL**, **HL**, **VL**, and **aL** are for the left camera of a stereo pair, the corresponding vectors for the right camera are represented by **CR**, **HR**, **VR**, and **aR**. Let the left image and right image coordinates of an object point located at **P** be represented by (il, jl) and (ir,jr), respectively. Both (il, jl) and (ir,jr), are computed in mm and derived as per equations (3) through (6). From Equation 6:

$$il = (\mathbf{DL \cdot HL}) / (\mathbf{DL \cdot aL}) \quad \text{and} \quad jl = (\mathbf{DL \cdot VL}) / (\mathbf{DL \cdot aL}) \quad (21)$$

where **DL** is the vector from the focal center of the left camera to the object point at **P**. Vector **DL** is related to **CL** by **DL** = **P** - **CL**.

From these equations for the coordinates of the image pixel in the left image we compute only the direction of **DL**. Assuming the following:

$$\mathbf{DL \cdot aL} = 1 \quad (22)$$

the expressions for il and jl are now:

$$\mathbf{DL \cdot HL} = il \text{ and } \mathbf{DL \cdot VL} = jl \quad (23)$$

The three components of **DL**, denoted by DL.x, DL.y, and DL.z, as per Equation 22 and Equation 23 can be recast into the following form:

$$\begin{aligned} (DL.x \cdot aL.x) + (DL.y \cdot aL.y) + (DL.z \cdot aL.z) &= 1 \\ (DL.x \cdot VL.x) + (DL.y \cdot VL.y) + (DL.z \cdot aL.z) &= jl \\ (DL.x \cdot HL.x) + (DL.y \cdot HL.y) + (DL.z \cdot HL.z) &= il \end{aligned} \quad (24)$$

These three equations can be solved using Gaussian elimination with pivoting for the three unknown components of **DL**. Presently the computed **DL** only represents the direction information, because of the assumption represented by Equation 22. The assumption is made that the actual vector from the focal center of the left camera to the object point is k**DL**, where k is an unknown scalar constant. The right image contains information to determine k. By use of the perspective transform for the pixel coordinates ir and jr:

$$ir = (\mathbf{DR} \cdot \mathbf{HR}) / (\mathbf{DR} \cdot \mathbf{aR}) \quad \text{and} \quad jr = (\mathbf{DR} \cdot \mathbf{VR}) / (\mathbf{DR} \cdot \mathbf{aR}) \quad (25)$$

The vector **DR** may be expressed as

$$\mathbf{DR} = \mathbf{P}\text{-}\mathbf{CR} = k\mathbf{DL} + \mathbf{CL}\text{-}\mathbf{CR} \quad (26)$$

Computing k is as follows: k =

$$\{ir\,(\mathbf{CR}\text{-}\mathbf{CL}) \cdot \mathbf{aR} - (\mathbf{CR}\text{-}\mathbf{CL}) \cdot \mathbf{HR}\} / (\mathbf{DL \cdot HR} - ir\mathbf{DL} \cdot \mathbf{aR}) \quad (27)$$

Plugging k back into the equation completes the depth perception computations, therefore, given two image points (il, jl) (ir,jr) from the left camera image and the right camera image respectively, the procedure will compute a 3D [X,Y,Z] real world coordinate point, plus or minus some epsilon of error.

7. How the System Works.

In order to putt a golf ball into the cup the first thing that must be done is to find the cup in each camera's field of view. From a bimodal distribution of the histogram of the area we perform the optimal binary threshold procedure, (see Horn [10]), which computes the best binary threshold for the putting green in contrast to the cup. Computing the optimal binary threshold compensates for various lighting conditions e.g., the difference between morning and afternoon sun, cloudy or overcast days, etc. Once the threshold is computed the image is binarized. The cup is painted matt black to provide contrast with the predominantly green background. On a real golfing green the cup is painted white in order to help the human golfer contrast the cup with the grass. We have decided to paint the cup black to aid in determining if the white golf ball has gone in the cup.

Next the program scans bottom up (from the robot out into the putting green) to accurately find the cup using a modified scan and border following algorithm. The border-follow procedure, originally described by T. Pavlidis in [11], determines the perimeter of the cup and then we can compute the centroid of the cup in the left camera image. The same procedure is performed on the right camera's image to produce a centroid of the cup in the right eye (see Figure 5).

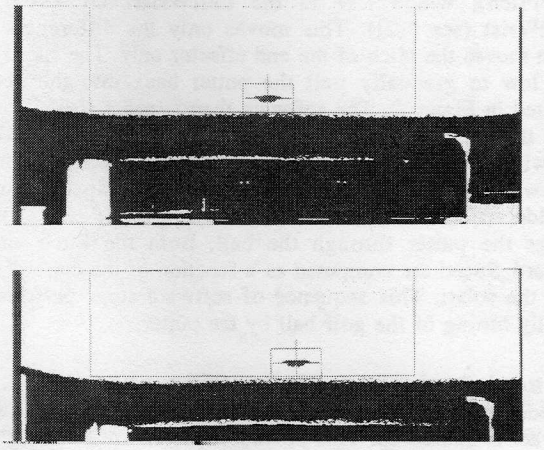

Figure 5. Right camera image (top) and left camera image (below) with centroid of the cup marked with a cross-hair.

The two centroids (il, jl) (ir,jr) can now be given to the Binocular Stereo Vision (BSV) procedure to compute the three dimensional point [X,Y,Z] of the centroid of the cup. Now that we have the 3D point of the cup we rotate the mobile base an angle θ1 to align the robot arm orthogonal to the cup. Due to the fact that the putter is rigidly attached to the arm 2.5 inches to the right of the center of the forearm, an additional rotating angle θ2 must be computed in order to align the putter orthogonal to the cup. Given the [X,Y,Z] real world position of the cup, the rotation of the mobile base- θ3 (shown in Figure 6) is computed as such:

$$h = SQRT(X^2 + Y^2)$$

$$\theta 1 = \text{asin}(X / h) \quad \text{and} \quad \theta 2 = \text{asin}(2.5 / h)$$

$$\theta 3 = \theta 1 + \theta 2$$

where h is the hypotenuse. The binocular stereo vision 3-space coordinate system uses the center of the robot mobile base as the origin (0,0,0).

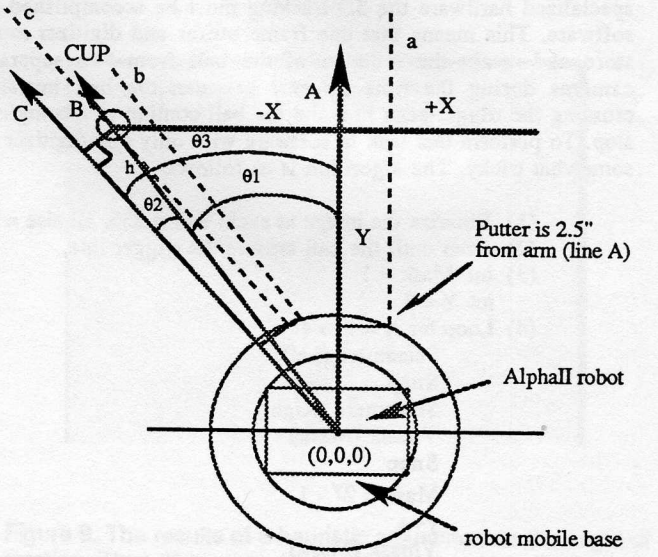

Figure 6. Computing the angle of rotation to align the putter orthogonal to the cup.

The RWI B12 three wheeled mobile base can rotate in position so the distance to the cup has not significantly changed after rotation. However, we recompute the distance to the cup after rotation to help mitigate the errors caused by lens distortion. After the first rotation the cup is relatively centered in the field of view. We then perform another *fine tuning* rotation to obtain more accurate results. As depicted in Figure 6, line A is the arm at its fixed home position coincident with the real world Y axis. Line *a* (dashed line) is orthogonal to the putter which is rigidly attached to the robot arm. Line B is the arm rotated by θ1 to align the arm with the cup. Line *b* (dashed line) is parallel to line B and orthogonal to the putter (after the arm is rotated by θ1). Line C is the robot arm rotated by θ3 and now its corresponding putting line *c* (dashed line) is orthogonal to the cup.

To start the alignment process the arm is first homed and then programmed to place its putter orthogonal to line *a* (and parallel to line A, the real world Y axis) by calling the procedure ROBOT$MOVE(X=0, Y=5, Z=-1.5, Pitch=90°, Roll=0°) in the robot device driver. This procedure moves the robot end effector to the Cartesian coordinate location. The device driver contains all the kinematic equations and computations for the AlphaII robot. The mobile base is then programmed to rotate in position by θ3 which now aligns the putter (line *c*) orthogonal to the cup.

The problem now is that programming the robot in a Cartesian coordinate motion to putt the ball effectively (swing the club) is

an arduous and difficult task. We have opted to program the stepper motors directly so as not to move the base, shoulder or elbow joints. This is accomplished by issuing the command: @STEP(speed=96, Base=0,Shoulder=0,Elbow=0, LW=Pitch, RW=Pitch), where LW is the LeftWrist and RW is the RightWrist (see [12]). This moves only the differential joint which moves the pitch of the end effector only. The speed is set very low to gradually pull the putter back into the position depicted in Figure 2. The software then issues a *sleep(1)* which holds the club back in its *Pitched* position for one second. The following stepper motor command is then issued to complete the swing of the club: @STEP (Forward_Speed, Base=0, Shoulder=0, Elbow=0, LW= -(Pitch*2), RW= -(Pitch*2)). This swings the putter through the ball. Both the Pitch and the Forward_Speed are computed as a function of the cup's distance from the robot. This sequence of software steps performs the actually hitting of the golf ball by the putter.

8. *Quasi*-Real-Time 3D Tracking of the Ball.

Immediately after the ball is hit towards the cup, the 3D tracking software is started up. This procedure begins by scanning along the bottom row of pixels in the left camera until the white ball (grey shade>250) crosses the scan line. This triggers a series of eight *bit-masked* snap shots of the ball in motion. Due to the fact that we only have one frame buffer and do not have any specialized hardware the 3D tracking must be accomplished in software. This means that one frame buffer and digitizer must store and retain the location of the ball from two separate cameras during the time interval between the ball initially crossing the trigger scan line and the ball coming to a complete stop. To perform this task in software with only one digitizer is somewhat tricky. The algorithm is as follows:

```
(1) Binarize the image as such: ball = 255, all else = 0
(2) Scan until the ball crosses the trigger line.
(3) int Mask = 1
    int Y = 2
(4) Loop for X = 1 to 4
        Setcamera (Left)
        Snap
        Setcamera (Right)
        Vmask (Mask)
        Snap
        Mask = 2^y - 1
        y++
        Vmask (Mask)
        Mask = 2^y - 1
        y++
        If (X == 3)
            sleep(4)   /* sleep for 4 seconds then
        wakes up to find where ball stopped*/
    End Loop
(5) End Algorithm.
```

where Snap is the digitizer freeze frame command, Vmask(Mask) video masks the bits set by Mask, Setcamera(X) switches to the X camera. Thus at the end of this loop the frame buffer shows two separate series of snap shots of the ball in motion. The *left* four balls (see marked cross hairs in Figure 7) being the snap shots from the Right camera and the *right* four balls being the snap shots from the Left camera.

In the frame buffer, pixels of grey shade GS=1 represent the ball at time t, pixels of GS=2 at time t+1, GS=4 time t+2, GS=8 time t+3, GS=16 time t+4, GS=32 time t+5, GS=64 time t+6, GS=128 time t+7. This exhausts the 8 bit video input masking capability, hence only 8 snap shots. The *left* camera's four balls are grey shades 1,4, 16, 64. Respectively, the *right* camera's four balls are grey shades 2, 8, 32, 128. One would expect that balls at GS=1 and GS=2 would have imaging locations along the same Y axis, i.e., their centroids would be along the same Y axis. Note, however, that due to the fact that the software must switch cameras (set a register) and set the video mask (set another register), there is a slight time lag (time Δ) between the left and right camera snap shots. The cross-hairs in Figure 7 mark the centroids. Note that the last pair of cross-hairs are aligned in the Y axis, this is because the ball has stopped.

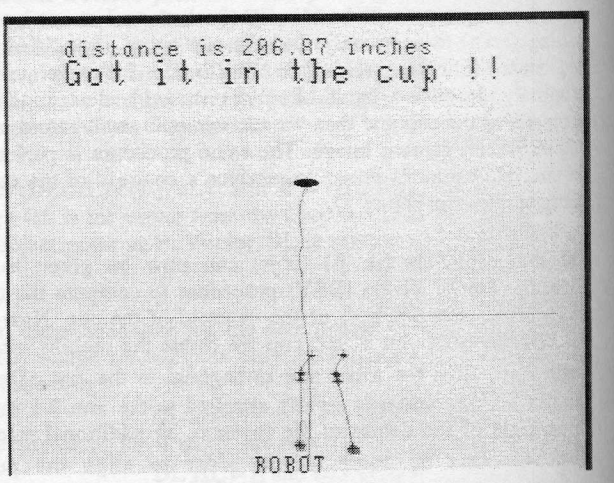

Figure 7. The Unix™ Spline function is used to curve fit the *left* camera's four snap shots and the *right* camera's four snap shots. Also shown is the quasi-real-time tracking of the motion of the ball as it is putted into the cup.

The two image centroids (il, jl) (ir,jr) of the balls GS=1 and GS=2 can not simply passed into the binocular stereo vision (BSV) algorithm. Doing so produces erroneous 3D points for the motion of the ball. Thus, given (il, jl) we need to determine where, in the image, the ball would have been in the right eye (ir,jr). This can not be produced by computing the centroid on GS=2 due to the time lag. We, therefore, used the Unix Spline function to curve fit the *left* camera's four snap shots (centroids) and the *right* camera's four snap shots (centroids). We can now take an arbitrary number of image pairs (il, jl), (ir,jr) to feed into the binocular stereo vision algorithm to produce an *approximation* of the 3D tracking of the motion of the ball. The results of piping the N splined image pairs (il, jl) (ir,jr) to the BSV procedure produces N real world coordinates.

The 3D real world points are passed into another call to the Spline facility to produce a smooth curve representation of the path of the ball as shown in Figure 7 (the line from the robot to the cup, shown in 2D). This 3D tracking software is used to provide a feedback mechanism to the robot to make adjustments

in its original computations for Pitch, Speed, and Angle of rotation.

9. Experimental Results and Heuristic Feedback Mechanism.

Due to many external factors: breaks in the green, curves, small hills and valleys, cumulative error over the green, distance, position of the ball relative to the putter, etc. some putts do not go in the cup on the first try. Therefore, it is necessary to feedback to the robot what went wrong so that it can make adjustments and eventually learn from its mistakes (missed shots).

Presently, the system can make adjustments to Pitch, Speed, Angle, and Strategy in order to correct itself and putt the ball into the cup. As shown in Figure 8 this putt has just missed the cup a little to the *left*. The Pitch and Speed are appropriate given this distance of 192.65 inches. Notice that the path of the ball was heading into the cup if it had continued on a straight line, but the curvature of the green caused a break of the ball. Thus, the putt curved off to the left and missed the cup. The robot was aligned properly, assuming a flat, straight shot to the cup. The 3D tracking software knows the final destination of the ball, the approximated path of the ball, and at what point the ball was closest to the cup (marked with a cross-hair). Thus, it can detect that the ball has missed the cup to the left and by how far. The cross-hair in Figure 8 shows the closest point that the ball ever got to the cup.

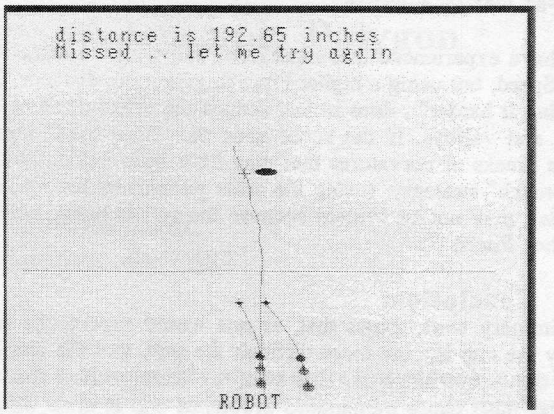

Figure 8. This putt has just missed the cup a little to the *left*. The Pitch and Speed are appropriate given this distance and curvature of the green (break). The Cross-hair marks the closest point the ball was to the cup.

As shown in Figure 9, the system has made an adjustment, but only to the angle of rotation because the final destination of the ball was not very far from the cup, i.e., the Pitch and Speed are appropriate given this distance. The angle adjustment is made by computing the difference between the actual angle and the computed angle to the cup and adjusting the present mobile base angle to 1/2 of this difference. The actual angle is computed using the closest point to the cup along the path of the ball. Then with the (Closest_X, Closest_Y) point, the angle is computed as:

$$\text{hypotenuse} = \text{SQRT}\ (\text{Closest\_X}^2 + \text{Closest\_Y}^2)$$

$$\theta\text{actual} = \text{asin}\ (\ \text{Closest\_X} / \text{hypotenuse})$$

$$\text{hypotenuse} = \text{SQRT}\ (\text{CUP\_X}^2 + \text{CUP\_Y}^2)$$

$$\theta\text{cup} = \text{asin}\ (\ \text{CUP\_X} / \text{hypotenuse})$$

$$\theta\text{rotate} = 0.5 * (\theta\text{actual} - \theta\text{cup}\ )$$

This heuristic appears to work fairly well in the preliminary practice runs so far. For example, when a putt that has missed the cup to the *left* by a relatively large amount and if the system adjusted the angle by the entire difference it may, in fact, putt the ball to the *right* of the cup by a significant amount as well. The reason for this is that many factors conspire to produce the errors, e.g., breaks in the green (small hills and valleys), curves, position of the ball relative to the putter, and cumulative error over the green. Our hill climbing method and the heuristic of 1/2 the difference tends to pull the path of the ball towards the cup each adjustment, rather than bouncing back and forth from the left of the cup to the right of the cup.

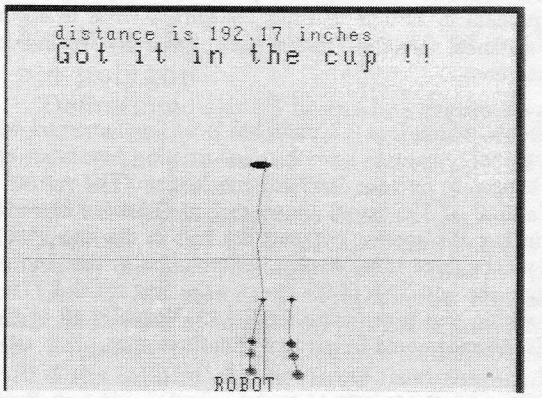

Figure 9. The results of a heuristic adjustment to the angle of rotation. Thus, the second try at this hole, with adjustments, succeeds.

Many professional golfers actually try to putt the ball approximately 24 inches beyond the cup. Of course, there are two distinct strategies when putting a golf ball: (1) to try to "hit it hard" (approximately 24 inches beyond the cup) so that it actually goes in the cup; or (2) to try to get the ball close to the cup (within 1 foot), meaning *approach* the cup gently so as to be able to easily tap the ball in. However, using the first strategy ("hit-it-hard") can produce a very difficult next shot if you miss because the ball may travel well beyond the cup. Our program uses a strategy that is heuristically determined from the distance of the cup. Longer putts begin with an "approach-the-cup" strategy and shorter putts use a "hit-it-hard" strategy. Of

course, based upon the results from the tracking software the strategy may be adjusted.

We have experienced that hitting the ball with the same Angle and Speed, but using a higher Pitch to gain more distance, hence "hitting it harder" , does in fact reduce the effect of some small hills and valleys. If the robot uses the "hit-it-hard" strategy, some breaks or curvatures that may have been evident with the "approach" strategy, (using the same parameters for Speed and Angle), may not be evident because the ball is being hit harder (higher Pitch).

10. Conclusion.

Preliminary work shows that, as one would expect, the further away the cup is, the more difficult the putt, and the less likely that the putt will go in. However, we have noticed that some shorter putts cause tremendous problems, and at other times the longer putts are easy. This is because the green is comprised of many small moguls and valleys, it is not just a flat, planar surface. If the surface was more like a billiard table then there would be many paths in which the ball could travel to go in the cup (all in a straight line towards the cup). However, with a complex three dimensional surface with many small moguls and valleys, there may be a limited number of possible paths for the ball to go in the cup. There may, in fact, be only one possible path or even worse, none! The difficulty of the putt is due to a number of factors which all conspire to build up error. However, it is prudent to assume that this conspiracy tends to accumulate over distance, therefore, longer putts, generally, are harder than shorter ones.

A process which logs all putts has been implemented which is used to help evaluate how to tailor existing heuristics and add new heuristics into the feedback mechanism. The results of this log (a total of 195 putts) shows that at distances between 120-150 inches the system can putt the ball in the cup 95% of the time within three tries. At this distance, the system can putt the ball into the cup 59% of the time on the first try and 77% of the time within two tries (using feedback). Virtually all of the putts at this distance could be made within three tries.

At distances of 151-170 inches the system can putt the ball in the cup 56% of the time the first time, and 80% within two tries and 92% within three tries. At distances of 171-220 inches - 45% the first time and 70% within two tries and 87% within three tries. Averaging over all distances (120-220 inches) our robot putt the ball in the cup 54% of the time the first time and 76% within two tries and 92% within three tries. This is noticeably better than the average person can golf, but not quite the performance of a professional golfer.

This paper has described the work-in-progress of the "ARNIE P." (Automated Robotic Navigational unit with Intelligent Eye and Putter) project. It was initiated to investigate the problems and develop software solutions for robotic tasks that require good hand-eye coordination and a sensor feedback mechanism to provide a closed loop robot vision system. The most novel aspect of this study is that by attempting to build or model a difficult perceptory task such as golf, which requires integrating many computational pieces (binocular stereo vision, robot arm motion, heuristic feedback, learning), it appears to be a good platform to experiment with artificial intelligence techniques and robotics.

A learning mechanism is currently being developed using a neural network to evaluate the adjustment parameters of Pitch, Speed, Angle, and Strategy based on experience. A navigational component is planned. A video tape of the entire process of actually putting a golf ball into the cup from a variety of angles and distances is available from the author.

Acknowledgements.

This work was partially funded by the National Science Foundation under Grant No. USE-9050371 and the Faculty Grants program of Millersville University. The authors would like to thank the following undergraduate research students: D. Helfrick, M. Beitler, S. Lovelidge, M. DeBeradino, G. Shay, and H. Kuo. Many thanks go to Dr. Paul Ross for comments on earlier drafts of this paper, and Dr. Albert Hoffman, Dean, and Professor Ronald Davis, Chairman, for their continued support of the activities of the Robot Vision and Artificial Intelligence lab. *Titleist-4* is a trade mark of the Acushnet Corporation, MATH-CAD is a trade mark of MathSoft, Inc., SparcStation is a trade mark of SUN Microsystems, Inc., ITEX100 is a trade mark of the Imaging Technology Co., Unix is a trade mark of AT&T Bell Laboratories.

REFERENCES

[1] Yakimovsky,Y. and Cunningham, R., "A System for Extracting Three-Dimensional Measurements from a Stereo Pair TV Cameras", *Computer Graphics and Image Processing*, Vol. 7, 1978, pp. 195-210.

[2] Avinash C. Kak, "Depth Perception For Robots", Ch. 16. in *Handbook of Industrial Robotics*, (New York: John Wiley & Sons, 1985), pp. 272-319.

[3] Alan M. Thompson, "Camera Geometry for Robot Vision", *Robotics Age*, Mar/Apr 1981, pp. 20-27.

[4] Ben Nobel and J. W. Daniel, *Applied Linear Algebra*, (Englewood Cliffs, N.J. : Prentice-Hall, Inc., 1977), pp. 179-195.

[5] Webb Miller and C. Wrathall, *Software for Roundoff Analysis of Matrix Algorithms* (New York: Academic Press, 1980), pp. 116-125. {code in Fortran in here}

[6] Andrew R. Magid, *Applied Matrix Models*, (New York: John Wiley & Sons, 1985), pp. 35-45.

[7] K.S.Fu, R.C.Gonzalez, and C.S.G. Lee, *Robotics-Control,Sensing,Vision, and Intelligence* (New York: McGraw- Hill Book Company, 1987) pp.307-328.

[8] Dana H. Ballard, C. M. Brown, *Computer Vision*, (Englewood Cliffs, N.J. : Prentice-Hall, Inc., 1982) pp. 20-27.

[9] Nicholas Ayache, *Artificial Vision for Mobile Robots-Stereo Vision and Multisensory Perception* (Cambridge, Massachusetts: MIT Press, 1991) pp.1-42.

[10] B.K.P. Horn, *Robot Vision* , (Cambridge, Massachusetts: MIT Press, 1986) pp.92-94.

[11] T. Pavlidis, *Algorithms for Graphics and Image Processing*, (Rockville, M.D.: Computer Science Press, 1982), pp.142-148.

[12] Microbot, Inc., Robot Manual Programming Guide Vol. II, (Sunnyvale, CA: 1986), pp. 44-56.