AIR HOCKEY

Mark Woodward (mwoodward@cs.stanford.edu)
John Rogers (jgrogers@stanford.edu)
Archan Padmanabhan (archanp@stanford.edu)

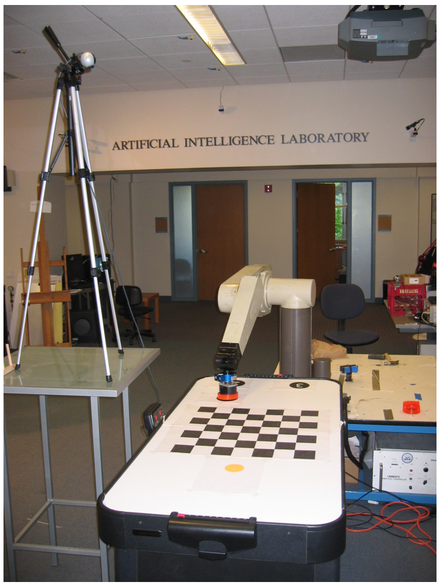


Figure 1: the air hockey setup

SUMMARY

The goal of our project was to create a system that could play competitive air hockey using a Pumma arm.

SETUP

1) A webcam was used for video capturing.
2) A Puma 500 for actuation
3) OpenCV was used to process images
4) The cs225a environment was used to control the puma

CALIBRATION

At the start of each session, the system needs to be calibrated. The result of the calibration process is two transformation matrices. The first is the transformation from table coordinates to camera coordinates, and the second is the transformation from table coordinates to robot coordinates.

-TableToCamera-

In finding the table to camera transformation, the system first finds the "checker board to camera" Transformation using openCV. But, the origin returned by OpenCV may not be the checker board origin that we expect. So the system then performs 4 transformations, searching for a specific orange spot. When the orange spot has been found, the system knows the transformation from the desired checker board corner to the corner returned by OpenCV.

From these two transformations, and the transformation from the table to the desired checker board corner, we know the transformation from the table origin to the camera

$${}^{cam}_{tbl}T = {}^{cam}_{cv-org}T \cdot {}^{cv-org}_{des-org}T \cdot {}^{des-org}_{tbl}T$$

-TableToRobot-

In order to create the transformation from the table coordinate frame to the robot coordinate frame, the system asks the user to place the end effector on three dots. The location of the three dots is known with respect to the table and the location with respect to the robot is captured as the user places the end effector on them.

The vector from dot two to dot one is aligned with the table X axis, so the norm of this vector in robot coordinates is the first column of the rotation matrix for converting from table to robot coordinates. The vector from dot two to dot three is aligned with the table Y axis, so the norm of this vector in robot coordinates is the second column of the rotation matrix. The cross product of the X vector and the Y vector gives the third column of the rotation matrix. The three vectors are then slightly modified to inforce orthogonality.

The 4th column of the Transformation matrix is then set to the robot location of the second dot. This then gives the transformation from dot two to the robot coordinates. The transformation from table coordinates to Robot coordinates is then given by

$$^{robot}_{tbl}T = {}^{robot}_{two}T \cdot {}^{two}_{tbl}T$$

EXECUTION

Once the system is calibrated, the playing begins. The system performs 4 steps while playing the game. First, it locates the puck in the image, then it projects this location onto the table coordinate system, then it integrates this information with the current state of the puck and computes the intersection time and location with the robot line, and finally it commands the robot to move to the intersection point and to strike the puck at the appropriate time.

FIND PUCK

The puck is found using an iterative HSV color space filter. The HSV filter parameters can be dynamically adjusted at runtime to handle various lighting conditions. Cross hairs are displayed over the puck, so that the user can easily see whether the puck is being tracked properly.

PUCK IN WORLD COORDINATES

In order to find the puck in world coordinates, the image coordinate are transformed to camera coordinates using the intrinsic camera matrix. The location of the camera and the tip of the puck vector in camera coordinates are then transformed into table coordinates using the $^{table}_{cam}T$ matrix computed in the calibration phase.

The difference of these vectors gives the direction of the puck with respect to the camera, expressed in robot coordinates $^{world}P_{puck-wrt-cam}$. The following equation is then solved to get X and Y in table coordinates. (3 equations, 3 unknowns). “t” scales the vector from the camera until it intersect with the table.

$$\begin{bmatrix} X \\ Y \\ 0 \end{bmatrix} = {}^{world}P_{cam-origin} + t \cdot {}^{world}P_{puck-wrt-cam}$$

PUCK INTERSECTION WITH LINE

Given a series of estimates for the location of the puck in table coordinates, an object exists that looks ahead, reflecting the puck off of walls, and computes the location where and time when the puck will intersect a line in front of the robot.

ROBOT CONTROL

The robot is commanded using a cubic spline in operational space with variable initial velocity. Every time a new puck intersect is computed, a new spline is generated. When the puck intersection time falls below some threshold, the robot is commanded to strike through the puck, rather than move to the location just behind the point on the computed intersection with the line.

GUI

Finally, a lot of effort was put into creating a user interface that is easy to use. The user is walked through each step of the calibration process, and can go back at any time. In addition, all large activities require confirmation, such as exiting the program, and re-entering the calibration steps once playing. Lastly, safety was taken into consideration. While in play mode, any button on the key board will float the robot and put the application in stopped mode. Enter would then start the robot again.

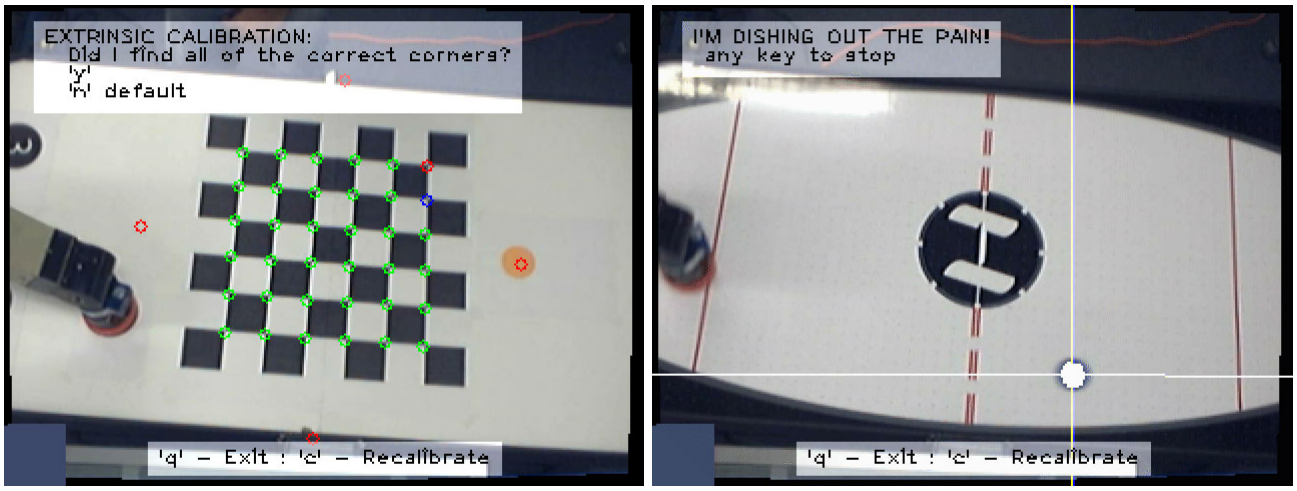
EXTENSIONS AND FUTURE WORK

1) Kalman filter for better puck state estimation (position, velocity, direction)

2) Use a 30Hz camera w/ capture card for faster frame rate and lower latency.

3) Active hiting (always try and hit in a direction that will go in the goal, either by reflection or a direct shot)

4) Detect goals, do a little dance of joy, or shame if scored on.



```
EXTRINSIC CALIBRATION:
  Did I find all of the correct corners?
  'y'
  'n' default

'q' - Exit : 'c' - Recalibrate
```

```
I'M DISHING OUT THE PAIN!
 any key to stop

'q' - Exit : 'c' - Recalibrate
```

Figure 2: calibration and tracking