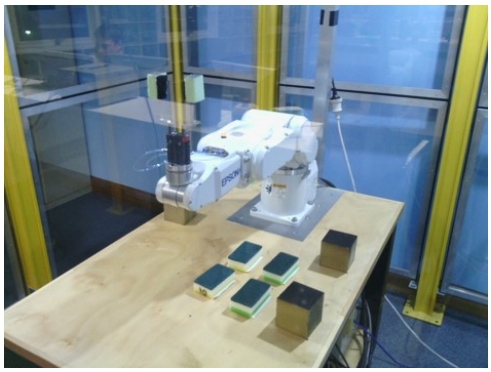
CONCENTRATION
Epson Manipulator Card Sorting System

Kausthub Krishnamurthy
312086040
Email: kkri3182@uni.sydney.edu.au

James Ferris
311220045
Email: jfer6425@uni.ssydney.edu.au

Sachith Gunawardhana
440623630
Email: sgun5213@uni.sydney.edu.au

***Abstract*—An implementation of an Epson robotic arm as a card sorting mechanism aimed towards mimicing human style situational problem solving regarding the game Concentration (A.K.A. Memory) with the intent on addressing methods to simplify decision making processes for the game Set. By first addressing a simpler problem we can assess the suitability of this system overall in performing complex sorting operations in a way that directly mimics the way a human would perform the task.**



I. Introduction

The card game Concentration begins with a set of cards placed face down on a table. The player's objective is to pick up one card at a time and match it with another they have already seen before based on a specified criteria. This form of concentration is a building block to the game Set in that the similarity criteria can be modified and be made more complex. Something as simple as having the same card in every way to matching cards that share only some features.

The driving interest in this project was trying to exlore the similarities and differences between the series of processes and actions a human would take to complete a job and compare them with those that a robotic manipulator would need to undertake in order to match those actions. Since there is an inherently different way that machines work to humans we wanted to look into those differences and consider how best to make a system come as close to the original human playing style as possible.

Previous efforts from other research students around the world, on card sorting systems, seem to involve a robotic system that sorts cards it in a way that is specific to robots but lacks the capacity to PLAY the game.

II. Background

This section outlines the background knowledge that was important to the setup of this project. It has been split into the paradigms and concerns that are pertinent to each of the modules.

A. Program Control Paradigms

There are a few key factors to designing an Artificial Intelligence software to deal with problem solving games, many of which can be seen in good programming style across many disciplines of Mechatronics and Computer Science. These practices are used to simplify the way the computer "perceives" the world around it and affords us certain layers of abstraction that makes it both easier to program and less susceptible to semantic and logic errors.

1) Object Oriented Programming: OO Programming paradigms are a key feature in the way we handle the idea of a "card" in this system. A human will be able to look at a card and immediately associate with it specific traits that make it unique like the colour, shade, shape and number of shapes. In an Object Oriented programming style it's important for the program itself to be able to deal with the concept of a card (and its traits) in the same way. This incorporates the use of data structures that logically organise the information (pertinent to each card) in a way that it makes it easy to access (unlike storing values in arrays and having arbitrary correlations between objects and array indices) irrespective of the number of instances of the cards that are present.

2) Modular Programming: Maintaining code that can be modular and split into individual functions that can be reused on demand by multiple parts of the same program. The advantage of this is that it allows for less convoluted code leading to fewer logical failures and leaves potential for each individual function (already proven to work) to be used in a multithreaded version of the system. It also creates a clear function dependency heirarchy which makes the system easier to test both piecewise and in a cascaded series of running tests.

3) Error Handling: As with any game system there will be situations in which the AI reaches fail cases in which it cannot succeed in its objective. As such it is crucial for the system design to take this into account and to allow it to either recover or gracefully exit from these issues.

B. Vision Detection Principles

The task was to identify the model of the card from the variations on shape, shape count and the filler. Figure1 shows the different variables in the used card set.

Fig. 1. *Different types of cards*

As a part of the system the robot arm was appended with a camera mounted on top of the table which can be used to acquire images of the objects place on the table. These images can be used to identify the object properties using image processing techniques.

Shape	Shape Count	Filler
Triangle	One	Shaded
Rectangle	two	Non-Shaded
Ellipse	three	Block

Fig. 2. *Variables*

There are plenty of approaches to find features in image processing such as:

1) Edge detection
2) Binary image processing[2]
3) Corner detection
4) Hough transformation
5) Swift operator
6) Histogram comparisons

In this project Edge detection and Binary image processing methods were heavily used to identify the cards from the table and shapes in the cards.

1) Edge Detection: An edge in an image is a significant local change in the image intensity, usually associated with a discontinuity in either the image intensity or the first derivative of the image intensity[2]. The inbuilt Matlab toolbox for edge detection (the Canny[1] operator) was our primary port of call for this project.

2) Binary image processing: Using a histogram of the image a threshold can be identified to convert the image to a binary image with areas of interests. This method was feasible to this project as the back of the card gives pixel values that can be approximated to black whereas the front of the card gives pixel values nearer to white, allowing this to be a core factor in differentiating cards that were face up or face down.

After creating the binary image with interested regions Matlab's inbuilt functions can be used to identify centroids and orientations of the interested regions. Local positions can be transferred to world coordinates using a simple coordinate geometric calculation based on the 3 fiducial boxes placed in position. Fiducials can effectively be anything (including the manipulator arm itself) but it was simpler to use a fiducial that we knew was as close as possible to the table in order to minimise the need to calculcate object depth correlations between the arm in the air and the cards on the table.

C. Motion

When designing the motions undertaken by the robotic arm, the most important factors to consider are simplicity, efficiency and the possibility of degeneracy. Additionaly, of key importance was the design of the end effector - the gripper used to pick up the cards. The particular design of this would limit the motions that the arm was capable of, it must be carefully considered. Coding design is also an important factor, as the epson arm needed to be easily controllable via the MATLAB interface, with any errors beign able to be handled easily by the user (without having to spend exorbitant amounts of time positioning the robot arm manually, joint by joint).

1) Degeneracy: For any position of a robotic arm, movement to another position can run into problems. It may be that it is simply impossible for the arm to move there, as the endpoint of the robotic arm is outside the workspace of the arm itself. Alternatively, it may be that there are simply too many possible movements that could result in the end effector being placed at the desired point - there are multiple solutions to the inverse kinematics. In the case of the EPSON arm, the controller would be unable to choose a suitable set of movements and so return an error. Close consideration of the possible movemements taken by the arm are vital.

2) Gripper Design: Design of the grippers could effectively restrict the set of possible movements of the robotic arm. As such, it was important for is grippers to be as low profile as possible while still achieving their purpose. It would also be key to the effective running of the overall program, as being able to successfully pick up a card would determine whether or not the program could proceed.

3) Coding Design: Function coding for the EPSON arm must be modular and reversible. For every function, a single set of arm movements would occur. There would also be a reversing function, to return from that position. Any function had be callable from the MATLAB interface. This made the code extremely modular, and in some cases larger than necessary. However, it had the advantage of being extremely simple to follow, both in terms of readability and logic. It makes error handling far simpler, removing the need to worry about the overall program functioning perfectly.

III. Experimental Design

The following section details the design rationale and the process followed for each part of the system.

A. AI Setup

Program Flow: The program is set up in a way that adheres to the following flow structure:

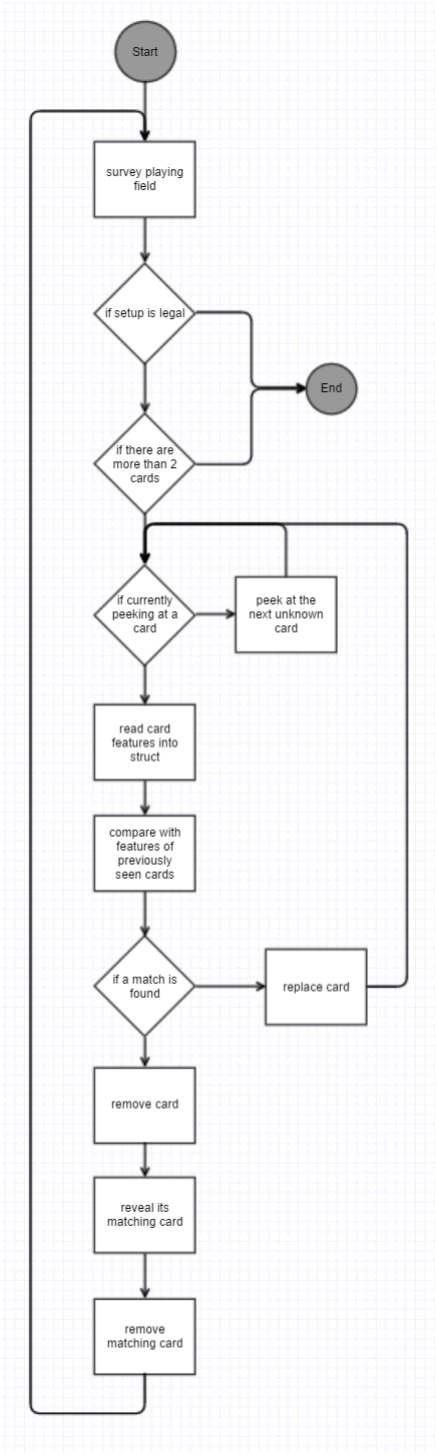


Fig. 3. *Program Flow Diagram*

This structure in Fig.3 focuses on claiming a matching pair as soon as it is found.

Game Modes: Consider the logic table in Fig.4 which shows us 16 different numbers that can be expressed as a single character where the the 4 right hand bits represent a flag value for what should be compared in each run of the game based on the game mode that is selected. The reason for this comparison structure is that if individual flags are set high we would have to poll all the flags but by using bits in a character as a flag system we can simply compare with a final value that we expect to see which reduces the number of flag checking to a single poll irrespective of the number of factors that need to be compared. This maximises the scalability of this program to series of cards with many more differences.

It's important to note that this compare function is designed to be modified for the game Set. In the game Concentration we use a Game Mode that's specified in code that masks the bits in a way that we only get identical pairs. Whereas to develop this for set we would have to see assign a "score" for each set that was found and at the end of examining EVERY card on the field figure out the combinations that will maximize the score that can be made from the series of cards at hand. In this setup we had Game Mode 11 as we couldn't acquire colour images from the camera. This game mode will accept sets of cards that have the same Shape, Filler, and Count.

Game Mode (dec)	Game Mode (binary)	Shape	Colour	Filler	Count
0	0000 0000	0	0	0	0
1	0000 0001	0	0	0	1
2	0000 0010	0	0	1	0
3	0000 0011	0	0	1	1
4	0000 0100	0	1	0	0
5	0000 0101	0	1	0	1
6	0000 0110	0	1	1	0
7	0000 0111	0	1	1	1
8	0000 1000	1	0	0	0
9	0000 1001	1	0	0	1
10	0000 1010	1	0	1	0
11	0000 1011	1	0	1	1
12	0000 1100	1	1	0	0
13	0000 1101	1	1	0	1
14	0000 1110	1	1	1	0
15	0000 1111	1	1	1	1

Fig. 4. *Game Modes*

Card Data Structure One of the more important parts of the setup is that the program is able to see what it considers a set of cards where each card has:

- X Coordinates
- Y Coordinates
- Orientation
- Shape
- Colour
- Filler
- Count

This makes it very easy to address individual traits on any set of cards. This needs to also have a trait that links it to other cards that it could form a set with and the score potential

B. Vision Detection Principles

In order to achieve the vision detection task we had broken down the task into 4 sub tasks as follows:

1) Identify the Cards on the table.
2) Coordinates and pose of the cards.
3) Match the local coordinates to the world coordinates.
4) Identify the features of the cards.

1. Identify the cards on the table.

- In the initial sweep after hiding the robot arm the camera will capture an image of the table. Histogram equalisation has been used to equalise the intensity over the image to make the image processing task easier.
- The back of the card gives pixel values that are very close to black as represented in a a grayscale image. Using this data a logical image has been created by using a threshold of pixel values less than 0.09. After removing continuous pixel groupings with areas less than 1000, *bwareopen* a structured element has been used to morphologically close the founded blobs.
- The same logic has been used to find face up cards as well but with a threshold of 0.91 to choose cards that are white in colour (and therefore are face up).

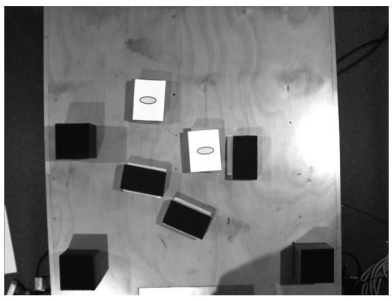


Fig. 5. *Initial captured image*

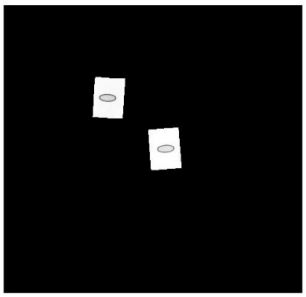


Fig. 6. *Isolated Cards*

2. Coordinates and Pose of the cards

- A logical image was created using the aforementioned methodand this image was then used to find the local centroids and the orientations of the cards.

The function *regionprops* has been used to find the centroids and orientation. After finding both, centroids and orientations of the fiducials have been removed from our collection in order to present the collection of cards back to the Card data structure. In orientations a redundancy has been added to make sure that the robot arm will always work in the safety limits by checking the value of the angle and deciding the shortest path to the desired gripper angle (in order to stop the arm from over-revolving around joint 4 and damaging the pneumatic lines that control the end effector action).

3. Match the local coordinates to the world coordinates

- Three fiducials have been used to get the relative position of the cards by using two fiducials at a time to create two theoretical axes (the lines formed between them) and then using a perpendicular distance calculation to find the real world position of each card as seen in Figure7. This is a simplification of the trilateration issue that we have encountered in the past with this Epson arm setup.

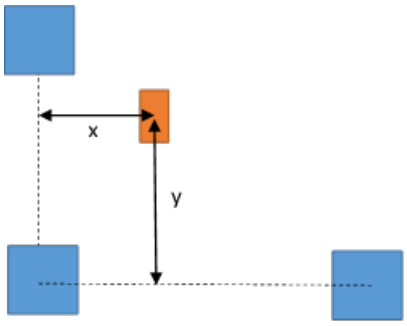


Fig. 7. *Method used to match with world coordinates*

4. Identify the features of the cards

- The main target was to identify the card itself before trying to extract features which used the same method as above to identify the card using pixel values as thresholds. We could then use the discorvered logical image as a mask to isolate only the card face that was relevant to us. This is one of the major advantages of using the peek-unpeek method as it is easier to control mechanically than to flip in place and the vision detection is simplified by being told exactly where to apply the masks.

- From this image a bounding box was used to crop out the card to process it for the features of shape, shape count and filler.

- Edge detection has been used to find the shape. After detecting the edges of the shape unnecessary edges has been removed using clearing edges connected to border and removing connected regions less than a threshold pixel area.

- Perimeter of the discovered blob was used to identify the shape as triangle, rectangle and ellipse had different perimeter values. Number of connected components in this stage has been used to identify the

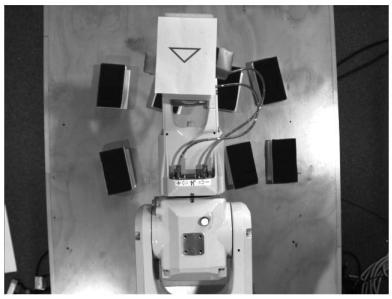


Fig. 8. *Initial image of the faceup card*

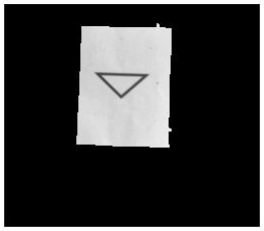


Fig. 9. *Isolated Card*

Fig. 10. *Identified Edges*

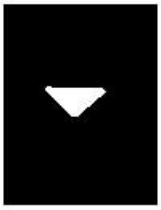


Fig. 11. *Identified Shape*

shape count. If the shape count is greater than 1, the mean perimeter will be checked for identify the shape. The final logical image with the shape has been used with the original image to identify the filler status. Mean intensity of the blob area of the logical image in original image will have a value greater than 0.72 if the shape is not shaded as pixel value for white is 1. For block status the mean intensity will be lowest and for shaded status the intensity value will be between 0.72 and 0.5.

C. Motion

The primary program functions for the movement of the Epson arm were dclearly defined, with each function performing a single command. Commands were chained together within Matlab. Every function was designed with a reversal function. Any error that breaks the program flow can only by undone by the reversal function, so this was absoulutely essential (especially during testing of the early stages of the overall program). The alternatives was to manually move the Epson arm, joint by joint, to a position from which a movement could occur. This was incredibly time consuming, so pre-built functions which would do this for us were essential.

- Go Home
  - This function moves the arm to a predefined home position. Care need to be taken when using this function, as it is not always possible for the arm to return to the home position, mainly due to degeneracy or self-obstruction. As such, this function is only ever called after calling the reversal of the previous function to run, if it had one.
  - This is the default reversal function
- Hide Epson Arm
  - This moves the epson arm out from the view of the camera mounted above the system, ensuring that the camera has a clear, unobstructed view of the table.
  - The reversal function is Unhide Epson Arm
- Unhide Epson arm
  - This is the reversal of Hide Epson Arm. It returns the Epson arm to the home position. Simply calling Go Home would result in a controller failure due to degeneracy, so a custom function was necessary
- Position Tool
  - Given a a centroid location of a card, this function moves the end effector to the *x-y* coordinates of the centroid.
  - The reversal of this function was Go Home, as it does not make any unsual movements which could cause degeneracy of obstruction.
- Set Tool Height
  - This function controls the *z* coordinates of the end effector. It uses predefined measurements of the height of the cards, so that the function can take an integer number defining how many cards high we want to be. For exampl, to pick up a card we go to height 0, which would actually be equal to half the height of the card, enabling the grippers to effectively pick it up. Height 1 would place the grippers in such a position where it could be a card stacked on top of another, and so on.
  - This function is its own reversal, simply returning to a default operating height that avoids collisions with the cards and feducials.
- Set Tool Angle
  - Given the angle to the horizontal that the card is on (as determined by the vision processing), this turns the grippers to that angle. Great care was taken to ensure that any twisting motion was limited to be between 0 and 180 degrees. Turning further than 180 degrees, or into the negative range, ran the risk of damaging pneumatic tubing that powered the grippers. This range was chosen as there is still a siginificant amount of motion left before the risk of damage occured, and the range also ensured that the a card could always be picked up.
  - This function is its own reversal
- Display Card
  - Shows the card to the audience - raises the grippers by 90 degrees.
  - Is reversed by UnDisplay
- UnDisplay Card
  - Returns the grippers to their default vertical position.
- Peek
  - Turns a card up so that it can be seen by the camera. This places the card in a fixed, known position, allowing identification by the camera simpler.
  - The reversal is UnPeek.
- Unpeek
  - The reversal for Peek
- Open Grippers
- Close Grippers

IV. Results

The image processing algorithm was able to match cards considering shape, shape count and filler. Results of the image processing can be summarised as follows:

Task	Completed
Identify & Differentiate Face Ups & Face Downs	100%
Identify the pose of cards & match with world coordinates	100%
Identify different Shapes	100%
Identify no of Shapes	100%
Identify the filler of the shape	100%
Identify the colour of the shape	0%

Fig. 12. *Results Table*

Note: We were only able to get gray images from the camera as there was a problem with white balancing with the camera driver which we found unable to solve manually. If RGB images were consistently procurable this could have been used to identify the colour of the card simply by checking the dominant colour channels (as RGB is a three-channel format) and would also then be converted into an HSV format image where Hue and Saturation adjustments could be made to resolve shadow problems.

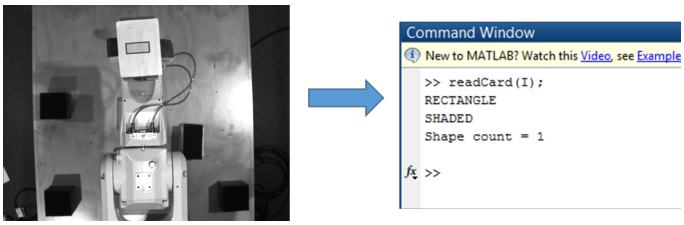


Fig. 13. *Results for a shaded rectangle*

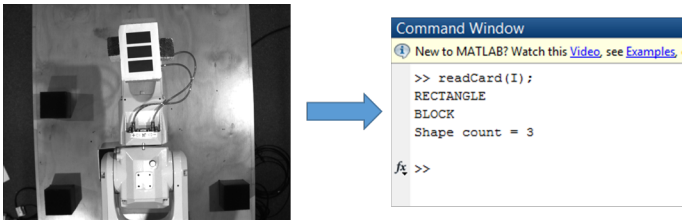


Fig. 14. *Results for three block coloured rectangles*

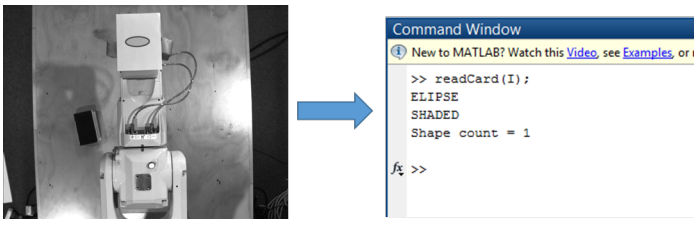


Fig. 15. *Results for Shaded Elipse*

V. Discussion

A. AI

One of the major unforseen issues that the program flow faced was that it would consistently compare a new card with itself and consider itself a match, leading to every card being put on the tower in the order that they were picked up. The issue was resolved by mainly that instead of having two states for cards (seen and unseen) the system needed to consider the newly read card differently so that it would only compare it with OTHER cards. Other than this the comparison functionality seemed to work consistently after that fix was introduced. It is also interesting to note that the split of motion functions into ”peek” and ”unpeek” came with its own share of problems while testing because when the program crashed from a failure during testing and development it would need to be manually pulled back into its working zone (similar degeneracy issues to hiding and unhiding the arm). The change that managed this was that at every error handling situation we would have to check if it was hiding or peeking and reverse that action before exiting the program with the error message. Once this change was made (prior to the presentation and demonstration) it performed without having such issues because by that point the errors in other areas of the programming (such as not being able to detect some cards) had been dealt with. The Hardware input dependent functions (such as survey field and read card and peek card) had to have these checks in place as majority of the error would be found in the physical end of the project. One of these issues that couldn't be solved was that the entire manipulator arm is a an open loop without any feedback for whether it was in the right position or whether it had picked up a card properly. In order to fix this the best way would be to apply touch sensors or flex sensors to the end effector that would allow us to deal with error situations when there is a malfunction such as not picking up a card. This could then be incorporated into the software testing procedures undertaken to ensure that the system was robust.

B. Vision

As shown in the results section the image processing algorithm developed was able to find matching cards irrespective of changes in card orientations. There were a number of pitfalls to the vision detection algorithm. One such unforseen pitfall was that we hadn't considered (initially) the possibility of trying to read a card when the arm had failed to pick it up in the first place which results in an image similar to Figure 16 or in Figure 17 which meant that some of our bounding box factors couldn't be fored, causing certain variables to be null which created an uncharacteristically abrupt failure to the system. By being able to adjust this (setting conditional default values) we managed to alter the system to be able to recognise that it had not identified any card, allowing us to handle that error and exit gracefully from the program.

Shadows caused by the sponges themselves (and the manipulator arm) were found to affect our ability to properly estimate the position of the cards. This is, in part, because we could not automatically manipulate the thresholds to remove shadows. This is particularly observable under dull lighting conditions when the shadow's pixel representation is similar

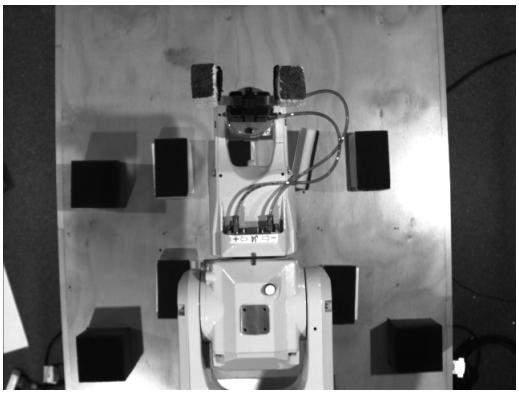


Fig. 16. *Failure to pick up a card*

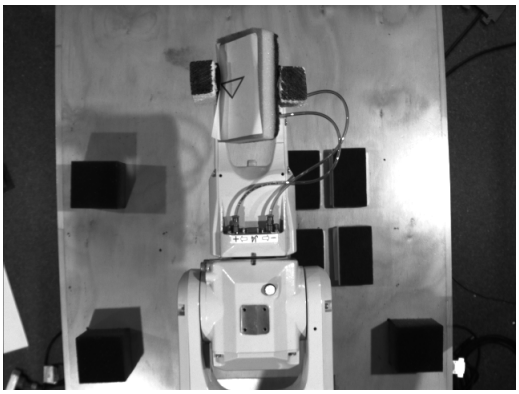


Fig. 17. *Struggling to hold the card properly*

to the back of the card and can be solved by obtaining an RGB image (which was subject to the aforementioned issues with the camera driver) and conderting it into HSV format and nullifying shadow effects. It could also be solved by removing the structures taht created the shadows (such as the sponges) by changing the end effector to work without the sponges because the cards then would be smaller and thinner, thus minimising shadow that they create. However the shadow of the manipulator arm itself is a constant issue and an alternate lighting arrangement could balanced light source by providing light in multiple directions to minimise the shadows (similar to the effect of multiple floodlights on a sports field).

C. Motion

1) Error Handling: Initial code styles had full functions written within the Epson arm program code, and Matlab simply calling that function. However, it was quickly determined that any error that occurred during this process caused the Epson arm to cease function wherever it may have been at the time (usually at the peek position, or arm-hidden position), with no way to return to the home position. Attempts to return to home resulted in errors due to degeneracy or self-collisions. Thus, it was realised that epson arm code should be made as modular as possible, and that every function had to have a counterpart that would reverse it. This made the the program code morre fragmented, but enabled the easy handling of any errors that occurred during operation. A system halt at the peek position could be easily solved by calling the reverse peek function. Any complexity to the arm motion happened in the Matlab script, enabling much greater control of the movements the arm made.

2) Arm care: Great care had to be taken to ensure that all motions the arm made were within the limits of the hardware. Usually, this posed no problem - the arm would not move to a position it that would result in self-collision. However, the grippers were powered by pneumatic tubing that ran down the length of the Epson arm. Any twisting motion, especially that of the joint that controlled the angle of the end effector, also twisted the tubing. If The arm twisted too far it could cause the tubing to become wrapped around the arm, stretching or breaking it and rendering the grippers useless - a very serious problem we wanted to avoid. Thus, limiters were coded into the Epson arm program code to ensure that any change of angle could not exceed a safe amount. As an additional factor of safety, any angle determined by the vision processing was further analysed in Matlab, and would be modulated to remain within the safe operating range, while still being able to pick up the card.

VI. Conclusions and Future Work

Considering the successes and failures discussed above the logical future steps would include acquiring a white-balanced colour input in order to be able to deal with colour differences and shadow removal. Generalising this code to work with any set of cards would be a positive step for the game "Concentration" but for the game of Set it would be counter productive. Instead, making the changes to the comparison logic and the card data structures to allow score analysis would be the next step in completing the game such that the robot would play both of these games as a human would, bringing to it the natural advantage that is its robust use of memory (which is the biggest downfall of the human capacity to play these games). Future development may also consider redesigning the gripper arm with something like a suction tip or similar grasping method that allows for the setup to eliminate the need for sponges as the robot will then be able to pick up the cards from the table by itself.

References

[1] Canny, John: *Pattern Analysis and Machine Intelligence* ,6th ed IEEE, 1986/11.

[2] R.Jain,R Kasturi,B.G.Schucnk: *Machine Vision*, 1st ed McGraw-Hill,Inc, 1995.

[3] Niku, Saeed B.: *An Introduction to Robotics: Analysis, Control, Applications*, 2nd ed. John Wiley and Sons, 2011.

[4] Register, Andy H.: *A Guide to MATLAB Object-Oriented Programming*, SciTech Publishing Inc., 2007, Georgia Tech Research Institute

[5] Shell, Michael: *IEEEtran*, IEEE formatted LaTeX template http://www.michaelshell.org/tex/ieeetran/