# Robot Gaming based on Computer Vision

Vito Janko
Jozef Stefan Institute
Jamova cesta 39
Ljubljana, Slovenia
vito.janko@ijs.si

Nejc Mlakar
Faculty of Computer and Information Science
VeÇcna pot 113
Ljubljana, Slovenia
nejcmlakar37@gmail.com

Jani Bizjak
Jozef Stefan Institute
Jamova cesta 39
Ljubljana, Slovenia
jani.bizjak@ijs.si

***ABSTRACT:*** *This paper describes a robot system that can play the popular Mediterranean card game called Briscola. It elaborates on the three main components needed for an operational platform. First, it describes several artificial intelligence agents that can play the game using a combination of probabilities, heuristics and the min-max algorithm. Second, it describes the computer vision component for card detection using both classical and deep learning approaches and finally, it proposes a scheme for a robotic arm that can move the cards on the table.*

**Keywords:** Intelligent Society, Computer Vision, Game Theory, Robot Arm

**Received:** 3 May 2018, Revised 12 June 2018, Accepted 19 June 2018



## 1. Introduction

Briscula is one of Italy's most popular card games, played all around the Mediterranean (Italy, Spain, France, Greece, Slovenia, Croatia). Despite that, the possibility of making an artificial intelligence player for the game is poorly researched, with no papers

on the topic found. We propose a system that can not only play the game on a computer, but can actually play against human players in the real world using a robotic arm. To do so, we developed three separate modules. First, we used a combination of heuristics, card probabilities and min-max algorithm to plan agent's moves. Second, we used computer vision (CV) algorithms to determine which cards the AI agent has and which cards are being played by the (human) opponent. Finally, we proposed a robotic arm that is capable of picking up a card from a predetermined spot using suction at the arm's end.

**1.1 Briscula - Rules Overview**

The game is played with a special deck, containing 40 cards divided equally into 4 colors - Spade, Coppe, Bastoni, Denari [Figure 1]. One card is selected at random at the beginning of the game and is placed face up under the deck. Its color is called "Briscola", giving the game its name. In this paper we consider the 2-player variant of the game. Both players start with three cards. Every turn both players play a card and at the end of the turn draw a card from the deck. After both players play their cards, the second player wins if his card shares the color with the Briscola card or shares the color with the rst card and has greater strength, which is based on the card number. The winning player gets points corresponding to the played cards value. After 20 rounds, whoever gets more than 60 points wins (a 60-60 score results in a draw).


Figure 1. The 4 aces of Briscola

**2. Playing the Game**

First task was to develop an artificial agent that would be able to play the game in a virtual environment. Due to the lack of external opponents, we developed several progressively stronger AI agents and matched them against each other to determine their strengths. Final version was also matched against five human opponents to further evaluate its performance.

***Mr. Random***

The rst agent plays cards completely at random. While this strategy seems inadequate, it both provides a basic baseline, and demonstrates an interesting property of the game: the game's variance is so high, that even this agent wins more than 5% of the games against the best agent (and significantly more against others), simply by having superior cards. High degree of chance, explains why progressively better agents have diminishing returns in their win rate.

***Mr. Greedy***

This agent tries to maximizes the score after the current round. It opens the round with the lowest card, while taking with the

strongest card, if possible, when second. While this provides the biggest boost in performance, its liberal spending of strongest cards does not lead to optimal play.

***Mr. Heuristics***
Agent implements author's expert knowledge using if-else rules. The wide set of rules includes holding strong cards until a valuable card is played, trying to start each round second if possible (as seeing your opponent play lets you know how to best respond) and being careful in which situations to open the round with a valuable card.

***Mr. Probable***
The use of heuristics can be amplified by predicting the likelihood of cards the opponent might be holding. Predicting that the opponent has a strong card of one color, might lead the agent to open the round with another. This was done by weighting the influence of each conflicting if-else rule, by the probability that it applies. The play was then determined by the "strongest" rule.

The prediction of card probabilities however, is not trivial. By counting the cards already played, we can determine cards left in the deck and calculate the base probability that any of them is in the opponent's hand, given his hand size. This probability can be further modified by two factors. First, we can exclude some types of cards from their hand, given their past plays. This step assumes that the opponent has an elementary knowledge of the game, and will play the obviously good play, given opportunity. For each card in opponent's hand, we track when was it drawn, and what plays were made since then. This allows us to predict more precisely what kind of card it is. Second, cards of high strength or Briscula color tend to get "stuck" in player's hand, as players wait for a good opportunity to play them. This means that their likelihood of being in a hand is greater than the base probability would suggest, especially in the later game. Their probability was weighted with an empirically determined weight, that moved from 1 to 1.5 as the game progressed.

***Mr. Calculator***
To avoid the if-else behavior, an agent can try to calculate different game branches and then decide for one that most likely leads to the desired outcome. There are two popular frameworks for this task: variants of the min-max algorithm and the Monte-Carlo tree search. We decided to try the former and leave the latter for future work.

The base version of the min-max algorithm [8] works with perfect information and thus had to be adapted for this probabilistic case. Instead of using the probabilistic variant of min-max, that would have a huge branching factor, we tried to transform the problem into a perfect information one. Three cases were considered. 1.) In the last three rounds, all cards are drawn and thus we have the case with perfect information. 2.) When only a few cards remain in the deck, we can do an exhaustive search of all possible orders of cards in the deck and all subsets of cards that can be in the opponent's hand, and do a simple min-max search for each possibility, averaging the results. 3.) In remainder of the game we sampled 100 dierent hands the opponent could have each round, with regards to the probability described in the previous subsection. For each of the possibilities the min-max search is performed and the results were averaged. In all cases, the search depth was set to three rounds, as reliability of our information on the opponent decreases with time. The heuristic used at the end nodes was simply the number of points accumulated in those rounds.

This variant performed best of all described and matches expert human play. The contribution of each min-max use case was individually assessed by replacing it with heuristics for that part of the game, and it was determined that all three parts contribute to the game-play improvement.

**3. Recognizing Cards**

In order to be able to play the game in real-life, it is essential to detect cards on the table and cards that are picked up from the deck. In this image recognition problem we assume that the card's images are constant and that they are placed on a mostly uniform background - table. The problem gets complicated due to the fact, that the card's images can be very similar, they frequently overlap in practical play and they can be sometimes covered by the opposing player's hands. Here we present several attempted approaches, ranging from the simplest to the beyond state-of-the art deep learning.

***Removing Background***
Since the cards are on the table and thus the surface color does not change much, we first tried to remove the back- ground color from the image and detect cards left on the table. A predetermined threshold, based on the RGB values of pixels was used. This

approach proved unreliable, as the subtle changes in lightning (lights in the room, clouds over the sun) could change the color scheme enough for the threshold to fail to remove enough of the background.

***Comparing Differences***

Similarly, since most of the image is static (table, deck of cards) and the only changes are the two cards being placed on the table, we looked at the history of images and tracked changes between them. Ideally, when a new card is placed on the table it should be the only changed part of the image and could easily be detected. Once exact card position is known, any template matching technique could be used to identify the card. The same approach could solve the overlapping cards problem, as they could be identified one by one, as they are played. In reality however, this approach did not work either. Due to the camera noise, most of the image was constantly changing. Second and bigger problem was that when a card was placed on the table, hands and their shadows went over half of the table, changing pixel values in the image, complicating the use of this approach.

***Edge Detection***

Another attempt was to detect the edges of the objects on the table using Canny Edge detector [2]. To detect a card from the edges we used Hough Line Transform [3] in order to detect straight lines that could later be combined into square shapes to form cards. This improved the results significantly, however due to the camera noise and wood pattern of the table, the edges were often miss-detected.

***SIFT***

SIFT [5] is a scale-and-rotation-invariant image-recognition algorithm, which means that the object in the image can be rotated or scaled and the method should still be able to detect it. SIFT finds so called interesting points (features) in the image, these

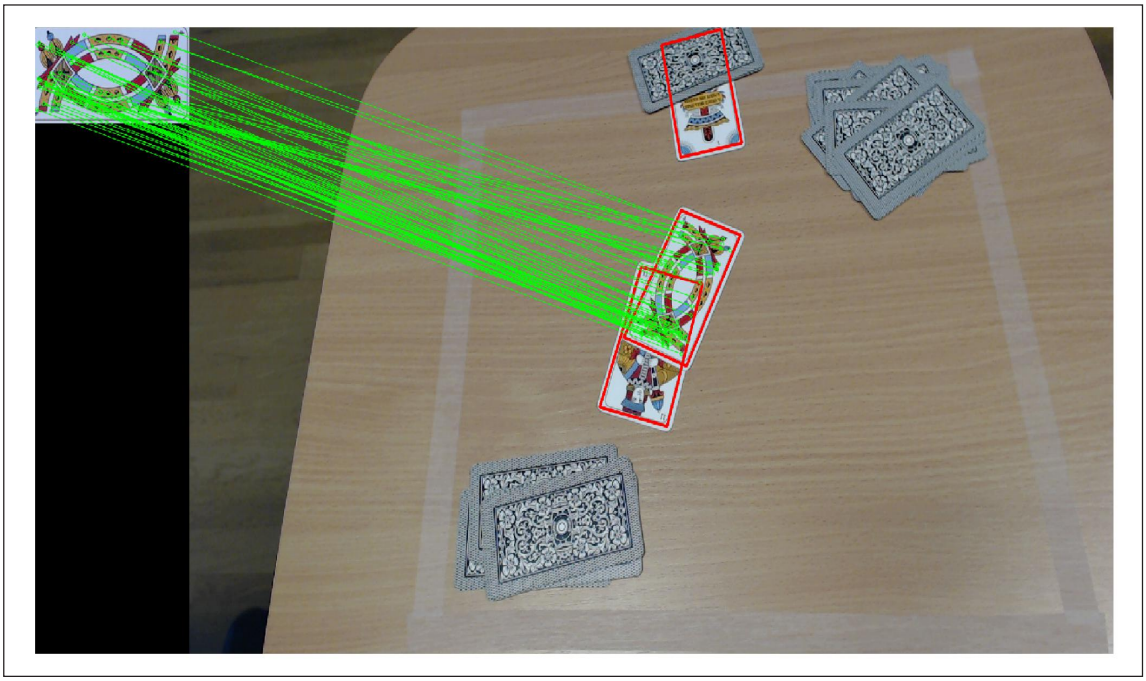

Figure 2. Detecting Spade 4 card. Matching features from template image to camera view

are usually shapes of edges, stores all features from the sample images and then compares them with the features found in the new images, as seen on Figure 2. By comparing features position, the algorithm can also determine the image position, rotation and scale.

This algorithm proved to work much better than previously described methods. It's detection rate was high, detecting even partially obscured (overlapped) cards. However, it has two drawbacks. For each frame in the video (image) it has to compare image (table) with 40 template cards. Depending on the resolution this can be very slow, ranging from 1 to 15 seconds. Sadly the

accuracy is correlated with the size of the image, which means that for good predictions high resolution images need to be used, which slows detection significantly. Second problem was distinguishing between lower "Denari" cards.

***Deep Learning***

In the last decade deep learning has become the dominant ML approach for multiple domains, with different deep learning architectures achieving almost human level precision for problems regarding computer vision.

The standard approach is using several layers of convolution, which is similar to what SIFT does, and then combining several fully connected layers in order to classify the features. This works ne for image classification, but is unable to detect objects on the image. A naive approach would be to have a small sliding window that goes through the whole image and classifies every part of it. This would be accurate, but extremely slow. Several approaches have been developed in order to tackle such problems: YOLO [6], Faster R-CNN [7], SDD [4]. Mentioned papers all go through the image only once (working with 30-60 fps), but still achieve comparable results to slower window CNN approach.

To test how well deep-learning approach works on our problem we implemented the YOLO architecture. Architecture consists of roughly 120 layers of convolution, pooling, regularization and fully connected layers. The training started with pretrained weights, obtained from VOC 2017 object detection. We manually labeled around 1000 card images, using the VOC format, and then trained the network for 3 days on NVIDIA's GeForce GTX 1080 graphic card. The trained network performed relatively fast, achieving around 15 fps, which is more than enough for real time detection. Detection accuracy was high when there were only 1 or 2 non-overlapping cards on the table [Figure 3], however it had problems with overlapped card. After investigating, we found out that since the network is trained with bounding boxes that are always aligned with the $x$ and $y$ axis, if the card is tilted at an angle, only half of it will be in the bounding box. Therefore the network is unable to learn to tightly detect a card and when they overlap the overall error is smaller if it just combines the two cards into one bounding box.


Figure 3. Detecting cards with YOLO is fast and reliable if objects do not overlap

To solve this problem we started working on a modified architecture that in addition to bounding box also predicts the angle at which the bounding box is rotated. The initial results on generated data (photos of cards stitched on top of different backgrounds) show promising results, where for most of the single cards in the image the network correctly predicts the rotation of the bounding box. The network works a bit worse where there are two overlapping cards but still manages to recognize a large percentage of images. We believe that with some more time, larger set of training images, tweaks and optimization of the architecture we could achieve close to 100% accuracy for the detection using this new architecture.

### 3.1 Robotic Arm and Cameras

The last step in bringing the agent to the real world is the presence of sensors and actuators. This component is composed of two cameras and a simple robotic arm. The robotic arm has 4 Degrees of Freedom created by 4 servo motors, that are controlled by Wemos D1 mini board. The board acts as a web client, receiving the commands from the main server and executing movement (controlling servo motors) actions. At the end of the arm there is a suction pump for lifting and dropping the cards. All movements are predetermined and described with sets of motor's rotation degrees. Two movement patterns exist: drawing a card and placing it on one of the three predetermined spots and picking the card from one of the three spots and dropping it at the center of the table.

The system also has two cameras, first one to overlook the table - tasked with detecting the cards played by the opponent. The second one is behind the arm, turned from the floor up. Before dropping a card on the table, arm is rotated so the card is over the

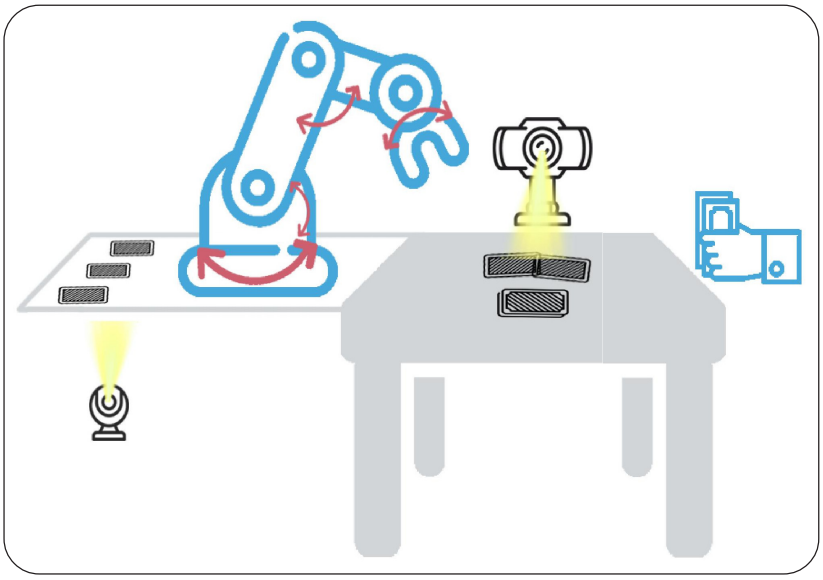

Figure 4. Server controls the robot arm. There are two cameras to oversee the table and picked up cards from the deck

| | R | G | H | P | C |
|---|---|---|---|---|---|
| R | - | 26 | 13 | 14 | 6 |
| G | 74 | - | 21 | 18 | 15 |
| H | 87 | 79 | - | 42 | 34 |
| P | 86 | 82 | 58 | - | 40 |
| C | 94 | 85 | 66 | 60 | - |

Table 1. Win percentage (%) of row agent against the column agent. Agents: Mr. Random (R), Mr. Greedy (G), Mr. Heuristic (H), Mr. Probable (P), Mr. Calculator(C)

second camera and can be identified. The system is schematically presented in Figure 4 and its first prototype is recorded and can be seen on the web [1].

The main logic for controlling the arm and taking actions is on the server, coupled with CV model and AI in order to take appropriate actions.

## 4. Results

We started by assessing the comparative strength of the different AI variants. Each played 1000 games against each other of variance in the games, as even random agent got surprisingly many wins and the best agent Mr. Calculator is achieving only 66% win rate against simple ones. In repeated experiments we noted, that win rate fluctuates ± 2% between runs.

Next we compared the play strength against 10 human opponents of different skill levels. Each played 10 games against Mr. Calculator. Results are listed in Table 2 and show an average 69% win rate of the AI against the human opponents. Volunteers that played, commented that the skill level of the agent is quite high, with some room for improvement in regards to increasing the agent's risk aversion. While the sample size is too small for definitive conclusions, we can assume that the agent is at least on par with average human players of the game.

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | avg. |
|---|---|---|---|---|---|---|---|---|---|---|
| 80 | 90 | 80 | 90 | 85 | 70 | 60 | 60 | 25 | 50 | 69 |

Table 2. Win percentage (%) of Mr. Calculator against 10 human opponents of roughly increasing strength

To test the CV component we recorded several human games from the same angle as the final system uses. We then manually compared the cards predicted by the CV with the actual ones. The best two approaches were SIFT and YOLO algorithms. The first worked awlessly in all cases, except differentiating some of the Denari cards. The second could awlessly recognize all cards, when they were not overlapped. Overlapped cards had roughly 50% accuracy. In the end we decided to use the SIFT algorithm for our rst system prototype, since second player usually overlaps the first card.

## 5. Conclusion

In this work we described three dierent components (from different computer science fields) of a system that is able to play the Briscola card game against the human opponent in a real-life setting. For each component we individually tried different approaches, creating a strong AI agent and a serviceable CV and robotic component. While the current version should be able to reliably play the game, all components still have room for improvement - we plan to test the Monte-Carlo search tree and improve the deep learning architecture. We hope that we will be able to successfully present a live demonstration at the paper's presentation.

## References


[1] Robot prototype. https://dis.ijs.si/wpcontent/uploads/2018/10/briscola/briscola AI.mp4, 2018.

[2] Canny. J. (1986). A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, (6) 679-698.

[3] Hough, P. V. Method and means for recognizing complex patterns. 18 1962. US Patent 3,069,654. (December).

[4] Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C.-Y., Berg, A. C. (2016). Ssd: Single shot multibox detector. *In:* European Conference on Computer Vision, p. 21-37. Springer.

[5] Lowe, D. G. (1999). Object recognition from local scale-invariant features. In Computer vision, 1999. The proceedings of the seventh IEEE International Conference on, volume 2, p. 1150-1157. IEEE.

[6] Redmon, J., Divvala, S., Girshick, R., Farhadi, A. (2016). You only look once: Unified, real-time object detection. *In*: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, p. 779-788.

[7] Ren, S., He, K., Girshick, R., Sun, J. (2015). Faster r-cnn: Towards real-time object detection with region proposal networks. *In* Advances in Neural Information Processing Systems, p. 91-99.

[8] Russell, S. J., Norvig, P. (2016). Articial intelligence: a modern approach. Malaysia; Pearson Education Limited.