Jackster: a blackjack dealer robot

Gianmarco Murru (gimu@itu.dk),
Ingrid Maria Christensen (inchu@itu.dk),
and Melissa Høegh Marcher (mhomu@itu.dk)

June 14, 2022

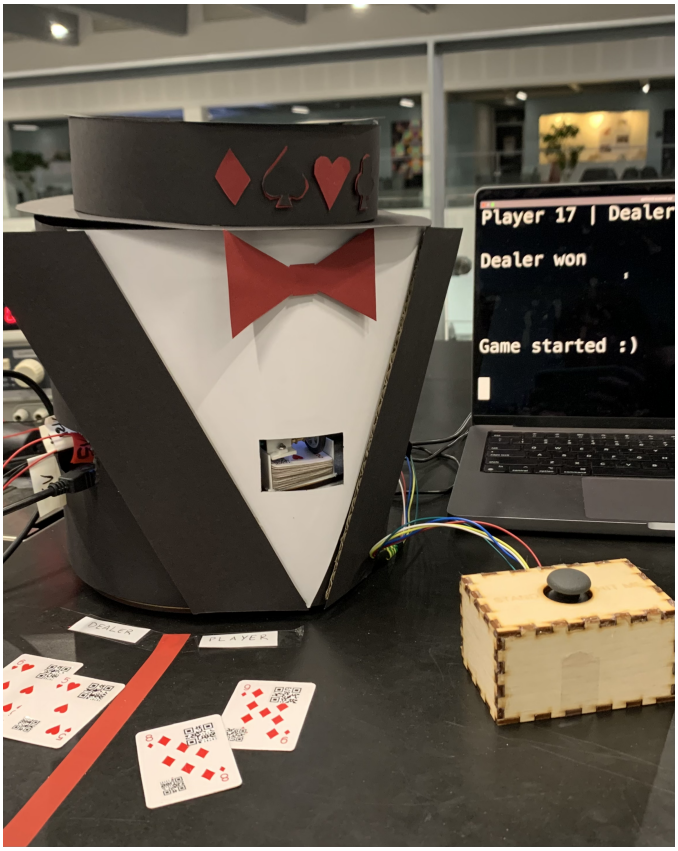

Fig. 1. Jackster: a blackjack dealer robot

1 Introduction

This report outlines the process, decisions and build of the blackjack-dealing robot, Jackster. When deciding on an idea for our final project, our main motivation was to build a robot you could interact with. As the result of an extensive brainstorm, we came up with the card dealer idea. We chose the game Black Jack, as the dealer has strict logic to adhere to.

2 Background

Blackjack is one of the most widely played casino card games in the world. The players do not compete against each other; instead, each player competes against the dealer. The object of the game is to get cards with a total value higher than that of the dealer's, without exceeding the value 21. For numbered cards, the card's number corresponds to its value. Picture cards have a value of 10. Ace can count as either 1 or 11. Each player is dealt before the dealer. For each round of dealing, a player can choose whether to "hit" (get a card) or "stand" (stop at the current value). The options "split" and "surrender" are not relevant for this specific implementation.

A blackjack dealer robot is not a new idea; robot-building hobbyist have been building card dealing robots for years [1]–[3]. Blackjack dealer robots exist as well, varying in complexity from using a full robot arm [4] to using a simple-but-ingenious LEGO setup [5].

3 Requirements

Jackster should at minimum be able to:

1) rotate between two positions
2) read the value of a playing card
3) keep both its own and the player's score
4) deal a card
5) allow the player to choose either "hit" or "stand"
6) signal to the player who won the game

The project has been a success if Jackster can run through a single game of blackjack with a single player.

4 Description

4.1 Mechanics

4.1.1 Rotation mechanism

As can be seen in Figure 2, the mechanism consists of 7 parts: three round laser-cut 3mm MDF boards of equal diameter, two identical 3D printed wheel tracks, a laser-cut turn-body structure with three attached 625zz wheels[1], and a Nema 17 stepper motor[2]. The rotation mechanism acts as a base for the robot, as all other mechanisms and electronics lie on the upper board. Naturally, this would be a large strain on the motor, had we not designed it in a way that removes as much weight from the stepper motor as possible. In

1. Something similar to: https://3deksperten.dk/trianglelab-iglidur-v-wheel-625zz.html
2. https://elektronik-lavpris.dk/p145432/42bygh910-04a-nema-17-stepper-motor/

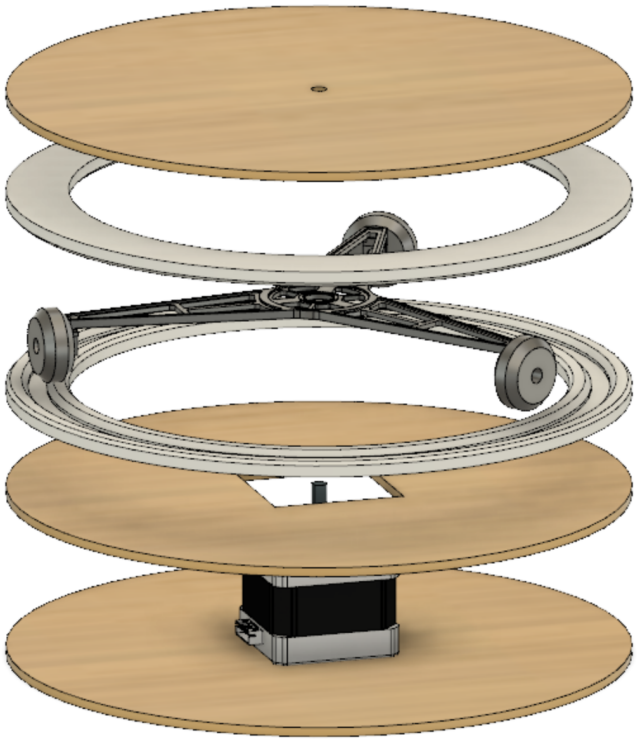

Fig. 2. Screenshot from Fusion 360 of rotation mechanism (exploded view). Note that we decided to use a laser cut turn body structure instead of 3D printed one.

our design, the upper board and upper wheel track lie on the wheels, distributing the weight across the three wheels instead. The only part of the stepper motor that touches the upper board is the tip of the rotating part. The two bottom boards are kept at a distance from each other (equivalent to the high of the motor) with four long screws and 12 nuts.

4.1.2 Card-dealing mechanism

As can be seen in Figure 3, the card-dealing consists of a card deck holder and a motor holder. We use miniature 54x39mm playing cards[3] to decrease the size of the robot. The motor holder is joined to the card deck holder with a simple rod piercing both components. The motor holder wraps around a small 5V DC motor[4] with an attached wheel, which pushes out cards when the motor turns. The wheel pushes out cards as the motor turns clockwise and pushes back any "escaped" cards as it turns counterclockwise. The mechanism's design was inspired by Mr Innovative's card dealing solution [1], but adjusted to our needs and drawn in Fusion 360. A technical drawing of the card-dealing mechanism can be found in Appendix B.

Building this mechanism, we encountered difficulties with cards sticking together. Not only does the cards' material create some friction. Friction also increases when handling the cards, as the natural oils on our fingers stick to the cards' material, making it an increasingly difficult issue to solve. Although the cards' material was not an adjustable variable in the friction-equation, we were able

3. https://www.temashop.dk/mini-spillekort
4. https://arduinotech.dk/shop/n20-micro-metal-gear-motor/

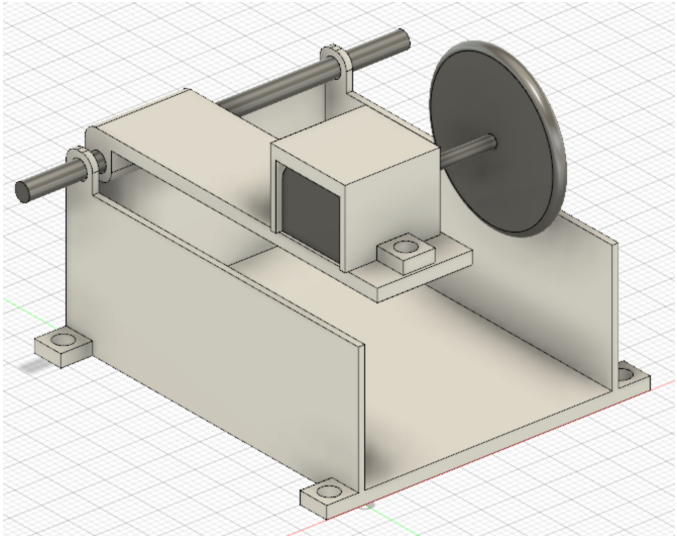

Fig. 3. Screenshot from Fusion 360 of card-dealing mechanism

to adjust the force being put on the cards, as well as the material of the wheel. The wheel is a LEGO wheel with a rubber tire. In an attempt of combating the friction between cards, we increased friction between the wheel and the cards by wrapping a rubber band around the wheel. However, this did not make any noticeable difference in the card distribution, so we scratched this idea. In an attempt of increasing the force between the cards and the wheel, we tried several different solutions.

First, we placed springs from Real lab and a square plastic plate underneath the deck of cards to push them upwards. This did not work out, as the springs were either too tight and created too much friction, or too loose and did not push up the card deck. Even if we had found perfectly tight springs, we would have needed wheels on both sides to distribute the weight evenly, which is not possible, as they would have taken up too much space and would have blocked the camera that reads the card's value.

Secondly, we tried placing extra weight on the wheel from the top, in addition to the gravitational weight affecting the current design. We did this by wrapping a rubber band around the motor holder. However, no matter how loosely the rubber band was fitted around the mechanism, friction was too high and the cards still stuck together.

Third, we bend the cards slightly at different angles to gently loosen them from one another. This did not seem to have any effect.

Fourth, we added a slight slope to the card deck by placing a piece of cardboard with an upside-down shape on the back of the card deck holder. We did this to remove the friction caused by the front edges of the cards. We also placed a small piece of cardboard below the deck of cards (on the same side as the wheel) to make the card deck even with the weight of the wheel pushing down on one side. We kept both of these cardboard fixes, as we saw a slight increase in the mechanism's ability to distribute cards.

Finally, after conducting tests described in section 5, we decided to slightly change the functionality. Instead of the card dealing mechanism "shooting" out a card onto

the table, it lightly pushes the card out for the player to grab the card themselves. While the wheel is still turning counterclockwise to push back in the remaining cards, the player will sometimes need to help push escaped cards back. Although this is set up in the user experience, it was necessary for the robot to be able to complete a game.

4.2 Electronics

4.2.1 Microcontrollers and power

We use an Arduino Uno[5] to control all electronics, except the camera used for reading card values. Two breadboards are used to create the circuit: a mini[6] and half size[7]. The card reading script can be run on a computer (either a Raspberry Pi 3[8] or a laptop), which communicates with the Arduino.

External power supply is used to power the two motors. The stepper motor is powered with 19V, while the DC motor is powered with 5V.

4.2.2 Motor drivers

To control the stepper motor rotating the base of the robot, we use a stepper motor driver model DRV8825[9]. The driver provides several safety functions such as overcurrent, short circuit, and over-temperature protection [6].

The robot uses an H-bridge model L298N[10] to control the DC motor for the card dealing mechanism. We chose to use DC motor controller because we needed the motor to rotate in both directions. This is easily done using a controller instead of wiring an H-bridge in the circuit.

4.2.3 Other electronics

We used an analog Arduino joystick[11] to allow the player to either ”hit” or ”stand” when playing a game. For reading the card values, we have used an HP Webcam 2300[12] and a white LED to provide light for the camera to read.

See appendix C for schematic drawings.

4.3 Software

4.3.1 Game logic

The game logic is defined by two different components: **Scanner** and **Arduino**. The **Scanner** is written in Python. It has access to a camera and the serial port connected to Arduino Uno. It was written following a guide by Pruthvi Hingu [7] and editing the source code related to it. The camera access is needed to read the QR codes placed on top of the cards. Every QR code contains plain text with the value of the card. The **Arduino** component is written in C++ and is the main code for the Arduino Uno. It is responsible to communicate through the serial port with the **Scanner** component sending messages to ask for a card reading and waiting for a message back that provides the value of the card. It is also the logic behind all the electronics we have, like the main stepper motor and the card-dealing DC motor.

4.3.2 Reading card values

To run the code that reads values from QR codes, you need to go through a few configuration steps. First, install OpenCV, zbar, and Numpy by running the commands `pip install opencv-python`, `pip install pyzbar` (alternatively `brew install zbar`), `pip install numpy`, and finally, `pip install pyserial`. Then, open a code editor and define the `usb_port` variable as the name of the USB port in the Arduino code editor. Run the `scanner-debug.py` script to check if the camera is set up correctly and able to scan QR values. If your computer uses the wrong camera (the laptop webcam for example), change the `camera` variable from 1 to 0 in both `scanner-debug.py` and `scanner.py`.

5 Results

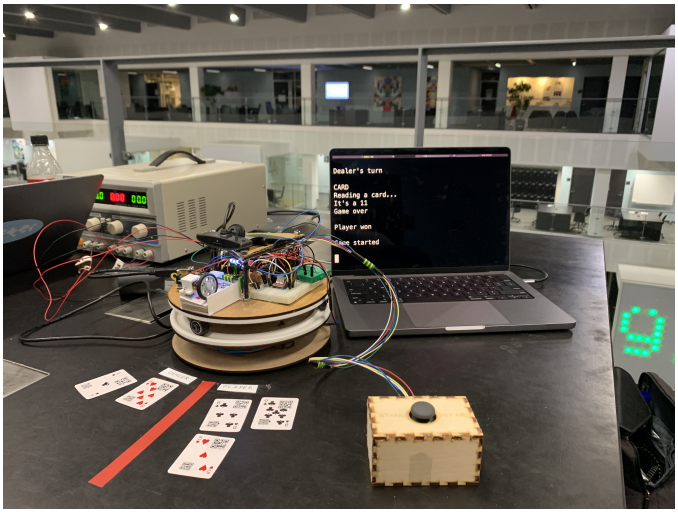

Fig. 4. Jackster after playing a game of blackjack

Overall, the functionality works as intended. Every individual mechanism and electronic component, except the card-dealing mechanism, works as intended. We are satisfied with the robot’s compactness (how everything is connected to the same Arduino and code) and weight distribution (the round, layered design).

5.1 Playing the game

This solution supports a game with one player (you) and one dealer (the robot). You start the game by pushing down the joystick like a button. The robot then reads and deals two cards to you, whereafter it rotates, and then reads and deals a card to itself. The interaction starts when the robot rotates back to you. Here, you decided whether you want to ”hit” (the robot reads and deals another card) or ”stand” (you are satisfied with your accumulated card value). You can ”hit” as many times as you want, but the game end and you lose if you get an accumulated value higher than 21. If

5. https://www.elextra.dk/p/arduino-uno-r3/H31941
6. https://www.elextra.dk/p/mini-breadboard-s%C3%A6t-170-huller-4-stk/H35419
7. https://www.elextra.dk/p/loddefri-fors%C3%B8gsbord-breadboard-400-huller/H35414
8. https://raspberrypi.dk/produkt/raspberry-pi-3-model-b-1gb-ram/
9. https://3deksperten.dk/drv8825-stepstick-1333.html
10. https://www.elextra.dk/p/2-kanal-l298n-h-bridge-dc-motor-drive-controller-til-arduino/H28064
11. https://www.elextra.dk/p/joystick-module-33v/H59806
12. https://www.power.dk/hp-hd-2300-webcam/p-254657/stock/

you choose to ”stand”, the robot will deal to itself until it has an accumulated value of at least 16. If the robot gets a value above 21, you win. Otherwise, yours and the robot's values are compared and the one with the highest value wins.

5.2 Testing

As all mechanisms, except the card-dealing mechanism, work completely as intended, we focused our testing on the card-dealing part. As described in Section 4, we encountered a lot of difficulties building this mechanism. When testing our final solution for making the robot ”shoot” out the cards, it had a success rate of 33%. After changing the mechanism to lightly push out the cards for the player to take them, it rapidly increased the success rate to 79%. In this test, we counted the card dealing as a success if the player was able to take the card from the robot without any trouble. We counted it as a failure if the robot dealt multiple cards at once which would make it hard for the player to only take the top card.

6 Discussion

Overall, we are happy with the design and build of the robot. We built the robot vertically, used a deck of small playing cards, and managed to fit a lot of electronics in a small space, which makes it both compact and portable. Out of our six requirements outlined in Section 3, four requirements (1, 2, 3, and 5) were completely met, while two requirements (4 and 6) was not met. We state that ”deal a card” was partially met, as it is only successful in some cases. This is explained in detail in Section 5.2. ”Signal to a player who won the game” was only partially met, as we did not have time to implement it on the robot itself, but instead used print statements on the computer.

Adding a few LEDs in different colors indicating whether the player won, the dealer won, or that it was a tie, would be a quick way to improve the robot. Other things that could have improved the robot include redesigning the card dealing mechanism from scratch, implementing a card-reading code that reads card values instead of QR codes, changing the size of the cardholder so a user can use any standard deck of cards, adding a small LED to the joystick that turns on when it is waiting and finally making the suit jacket casing out of fabric instead of paper.

Were Jackster to go into mass production, all 3D printed parts, as well as the laser-cut MDF boards, could be injection-molded for speed efficiently. Some pieces could even be molded as one, like the middle board and the lower track, or the upper track and the upper board (see Figure 2). In addition, all circuits should run on a custom PCB.

References

[1] M. Innovative], “Diy arduino based card dealing machine,” 2021.
[2] R. Reyes, “Card dealer robot - dispenser mechanism,” 2015.
[3] S. Kelly, “Card dealing robot,” 2020.
[4] R. Youhanna, “Robot blackjack dealer - ai11,” 2013.
[5] A. SIGCHI], “Tangiers toios: Robotic blackjack dealers,” 2021.
[6] MakerGuides, “Stepper motor with drv8825 and arduino tutorial,” Mar 2022.
[7] P. Hingu, “Build your own barcode and qrcode scanner using python,” Dec 2020. ”Code source: https://github.com/pruthvi03/Barcode-And-Qrcode-Scanner”.

APPENDIX

A: Bill of materials

	Number of units	Part	Cost per unit	Subtotal
Generel components	1	Arduino Uno	kr 219	kr 219
	1	Breadboard (400 holes)	kr 59	kr 59
	1	Mini breadboard (170 holes)	kr 25	kr 25
	-	Jumper wires	-	-
	-	Glue	-	-
	-	Double-sided tape	-	-
	1	Miniature card deck	kr 19	kr 19
	2	Sheets of cardboard for casing	kr 8,5	kr 17
Rotating mechanism	1	Nema 17 stepper motor	kr 159	kr 159
	1	DRV8825 stepper motor driver	kr 75	kr 75
	3	3mm MDF boards (Ø20cm)	-	-
	2	3D printed wheel tracks	-	-
	3	625zz wheels	kr 99	kr 297
	1	Laser cut turn body structure	-	-
	4	35 mm screws	-	-
	12	Bolts	-	-
Card dealing mechanism	1	5V DC motor	kr 36	kr 36
	1	L298N H-bridge DC Motor Drive Controller	kr 69	kr 69
	1	3D printed motor holder	-	-
	1	3D printed card holder	-	-
	1	Rod	-	-
Joystick controller	1	Arduino joystick	kr 44	kr 44
	1	Laser cut MDF box	-	-
Card value reading	1	Rasperry Pi 3	kr 339	kr 339
	1	HP HD Webcam 2300	kr 299	kr 299
	1	White LED	kr 10	kr 10
	3	65 mm screws	-	-
	6	Bolts	-	-
	1	Laser cut triangle for the camera to lie on	-	-
			Total:	**kr 1.667**

B: Technical drawings

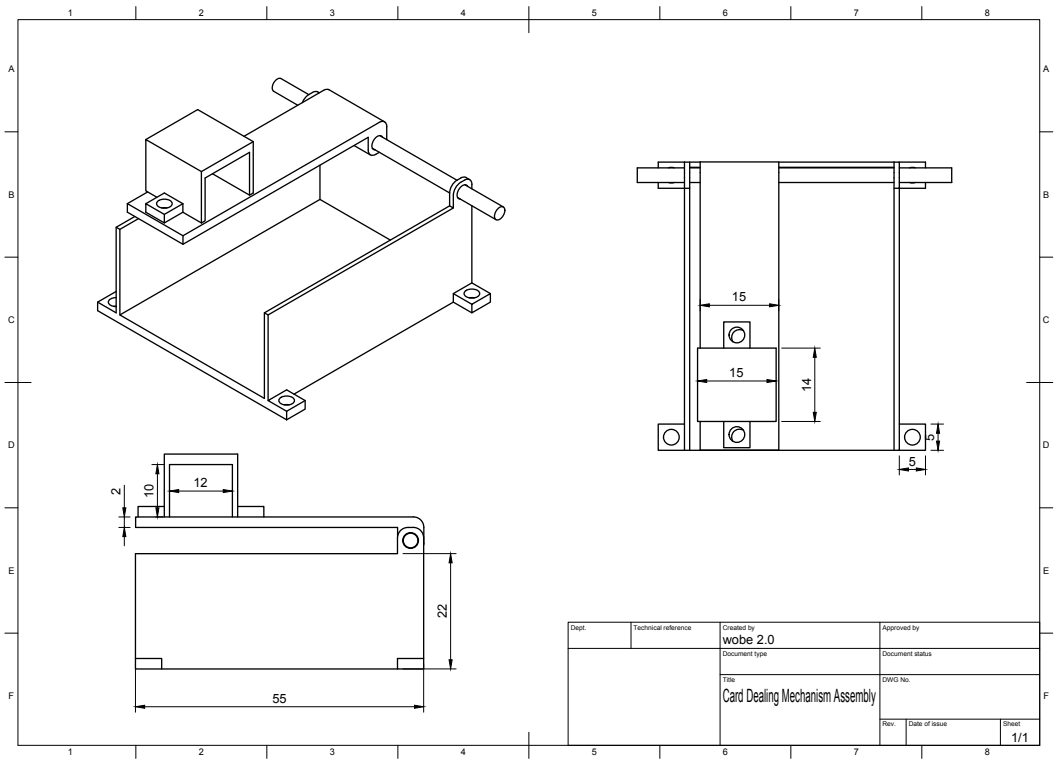

Fig. 5. Card Dealing Mechanism Assembly Drawing

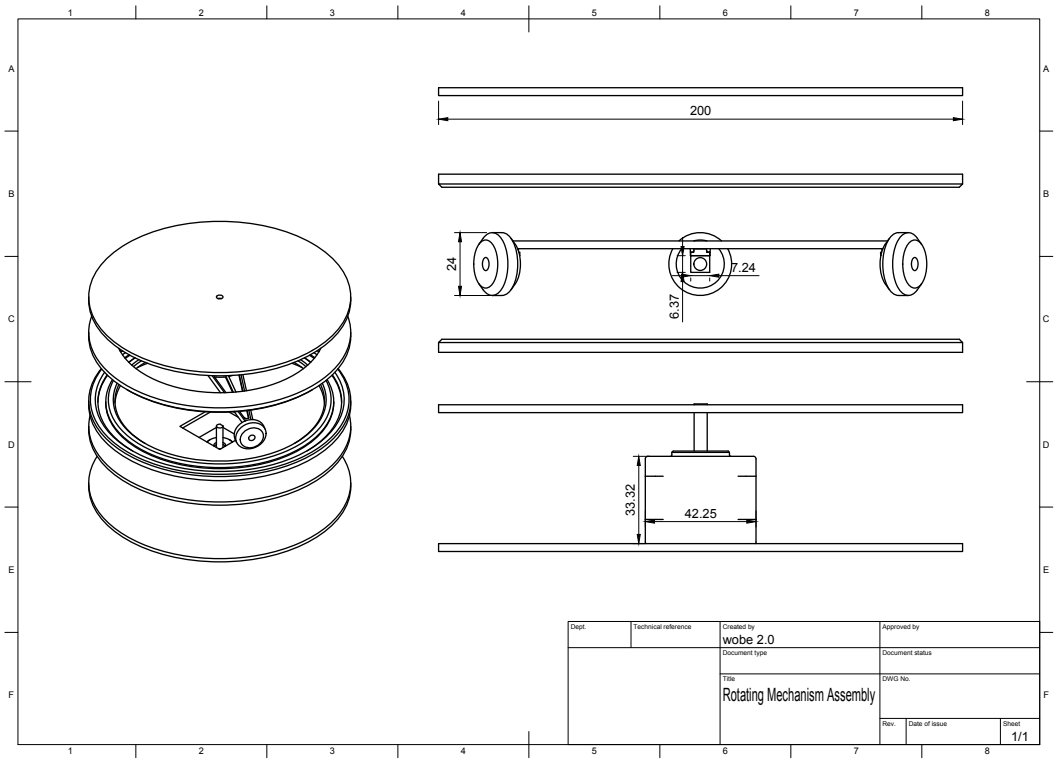

Fig. 6. Rotating Mechanism Assembly Drawing

C: Schematics

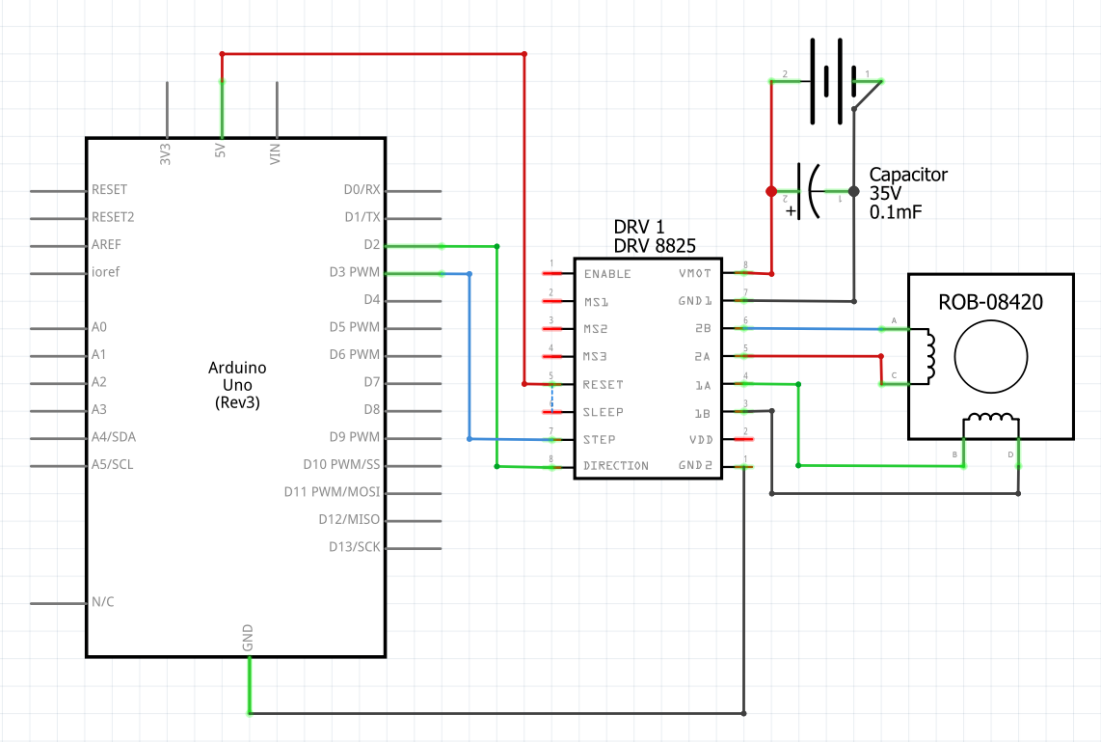

Fig. 7. Schematic view of the stepper motor used for the rotating mechanism

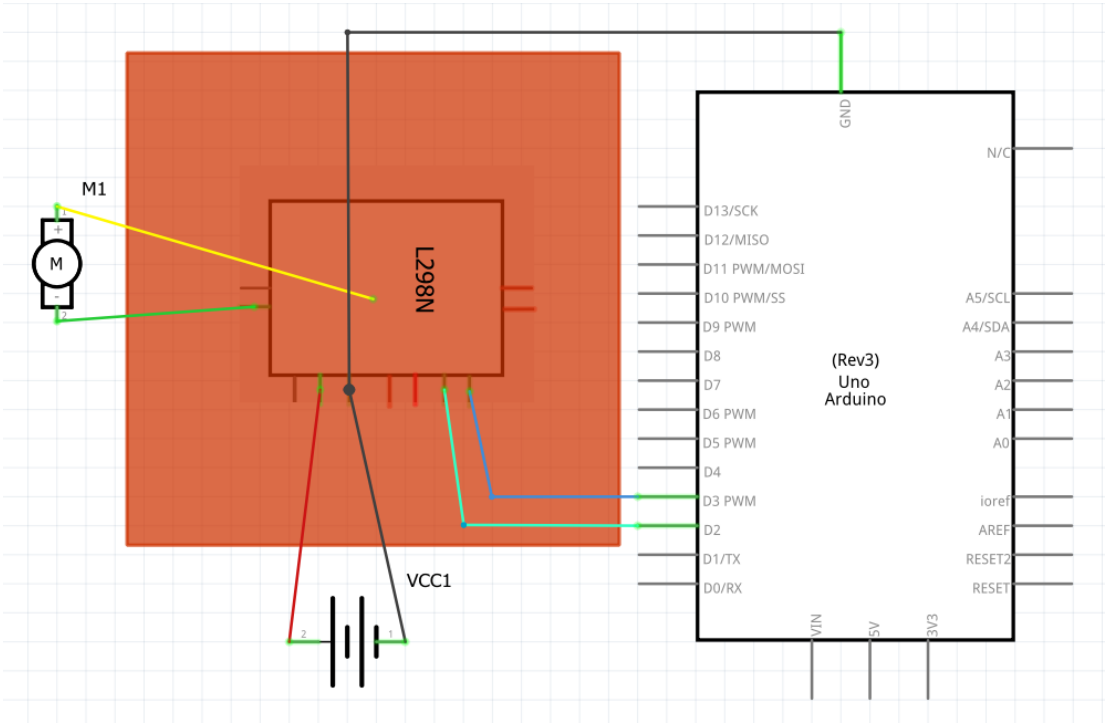

Fig. 8. Schematic view of the DC motor used for the card dealing mechanism. Note: as the model was not a part Fritzing, we used a ”buggy” version

D: Code

```
import cv2
import numpy as np
from pyzbar.pyzbar import decode
import serial, time, os

############## ARDUINO SETTINGS #############
```

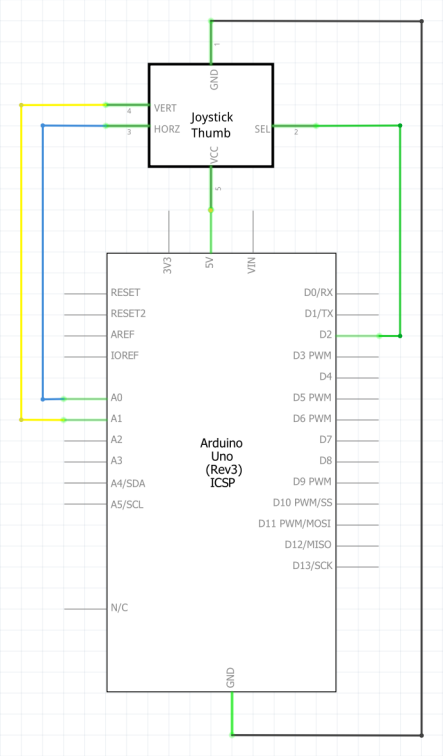

Fig. 9. Schematic view of the joystick controller

```
# Assign Arduino's serial port address
#   Windows example
# usb_port = 'COM3'
#   Raspberry example
# usb_port = '/dev/ttyACM0'
#   MacOSX example
usb_port = '/dev/cu.usbmodem21101'

camera = 0 #Depend on your machine, should be between (-1 and 1)

ser = serial.Serial(usb_port, 9600, timeout=1)

##########################################


def decoder(image):
    frame = image
    gray_img = cv2.cvtColor(image,0)
    barcode = decode(gray_img)

    for obj in barcode:
        points = obj.polygon
        (x,y,w,h) = obj.rect
        pts = np.array(points, np.int32)
        pts = pts.reshape((-1, 1, 2))
        cv2.polylines(image, [pts], True, (0, 255, 0), 3)

        barcodeData = obj.data.decode("utf-8")
        barcodeType = obj.type
        string = "Data: " + str(barcodeData) + " | Type: " + str(barcodeType)

        cv2.putText(frame, string, (x,y), cv2.FONT_HERSHEY_SIMPLEX,0.8,(0,0,255), 2)
        #print("Barcode: "+barcodeData +" | Type: "+barcodeType)

        return barcodeData

def read_card():
    data = None
    while not data:
        ret, frame = cap.read()
        cv2.normalize(frame, frame, 0, 255, cv2.NORM_MINMAX)
```

```
        data = decoder(frame)
    return data

cap = cv2.VideoCapture(camera)

def main():
    while True:
        if ser.in_waiting > 0:
            msg = ser.readline().decode('utf-8')
            print("{}".format(msg))
            if msg == 'CARD':
                print("Reading a card...")
                data = read_card()
                if data:
                    ser.write(bytes(data, 'utf-8'))
                    print("It's a {}".format(data))
                    #print("Waiting for messages...")
                    time.sleep(0.05)
        else:
            time.sleep(0.01)

if __name__ == '__main__':
    main()
```

Listing 1. Scanner component

```
// JOYSTICK PINs
#define joyX A0
#define joyY A1
#define joyP 8 //Pressed
//#define led1 10

int dealerVal;
int playerVal;
bool dealerHasAce = false;
bool playerHasAce = false;

// JOYSTICK
int xValue = 0;
int yValue = 0;
int joyButton = 0; // 0 pressed, 1 not pressed

// ROTATING MECHANISM
const int dirPinRM = 2;
const int stepPinRM = 3;
const int stepsPerRevolutionRM = 200;

// GENERAL MOTOR
const int dirPinGM = 6;
const int stepPinGM = 5;
const int stepsPerRevolutionGM = 200;


void setup() {
  Serial.begin(9600);
  pinMode(joyP,INPUT_PULLUP);
  // pins for card dealing dc motor
  pinMode(dirPinRM,OUTPUT);
  pinMode(stepPinRM,OUTPUT);

  // pins for base stepper motor
  pinMode(stepPinGM, OUTPUT);
  pinMode(dirPinGM, OUTPUT);
}

void loop1() {
  turnRobot("player");
  delay(2500);
  turnRobot("dealer");
  delay(2500);
}

void loop() {
  Serial.println("Game started");
  initiateGame();

  // PLAYER'S TURN
  turnRobot("player");
```

```
  playerGameplay();

  // IF PLAYER !DEAD, DEALER'S TURN
  if (playerVal < 21) {
    turnRobot("dealer");
    dealerGameplay();
  } else { // player lost
    playerVal = 0;
  }

  if (dealerVal > 21) { // dealer lost
    dealerVal = 0;
  }

  // END GAME
  Serial.println("Game over");
  Serial.println("Player " + String(playerVal) + " | Dealer " + String(dealerVal));
  if (playerVal > dealerVal) {
    Serial.println("Player won\n");
  } else if (playerVal == dealerVal) {
    Serial.println("It's a tie\n");
    turnRobot("player");
  } else {
    Serial.println("Dealer won\n");
    turnRobot("player");
  }

  delay(1000);
}


void turnRobot (String person) {

  if(person.equals("player")) {
      digitalWrite(dirPinGM, HIGH); // Set motor direction clockwise

      // Spin motor
      for(int x = 0; x < stepsPerRevolutionGM/4; x++)
      {
        digitalWrite(stepPinGM, HIGH);
        delayMicroseconds(8000);
        digitalWrite(stepPinGM, LOW);
        delayMicroseconds(8000);
      }

    } else if(person.equals("dealer")) {
      digitalWrite(dirPinGM, LOW); // Set motor direction counterclockwise

      // Spin motor
      for(int x = 0; x < stepsPerRevolutionGM/4; x++)
      {
        digitalWrite(stepPinGM, HIGH);
        delayMicroseconds(8000);
        digitalWrite(stepPinGM, LOW);
        delayMicroseconds(8000);
      }
    }
}

void initiateGame() {
  dealerVal = 0;
  playerVal = 0;
  joyButton = digitalRead(joyP);

  // WAIT FOR PLAYER TO PRESS JOYSTICK BUTTON
  while (joyButton == 1){
    delay(100);
    joyButton = digitalRead(joyP);
  }

  // DEAL 2 CARDS TO PLAYER
  playerVal += readValue("player");
  dealCard();
  playerVal += readValue("player");
  dealCard();

  // DEAL 1 CARD TO DEALER
```

```
  turnRobot("dealer");
  dealerVal += readValue("dealer");
  dealCard();
}

void playerGameplay() {
  Serial.write("Player's turn\n");
  if (is_hitme()) {
    playerVal += readValue("player");
    dealCard();
    if (playerVal < 21) {
      playerGameplay();
    } else {
      return;
    }
  } else {
    return;
  }
}

void dealerGameplay() {
  Serial.write("Dealer's turn\n");
  while(dealerVal < 16) {
    dealerVal += readValue("dealer");
    dealCard();
  }
}

int handleAce(String person, int val) {
  if(person.equals("player")) {
    if(playerHasAce) val = 1;
    else {
      if (playerVal + 11 < 22) {
          val = 11; playerHasAce = true;
      } else {
          val = 1; playerHasAce = true;
      }
    }
  } else{
    if(dealerHasAce) val = 1;
    else {
      if (dealerVal + 11 < 22) {
          val = 11; dealerHasAce = true;
      } else {
          val = 1; dealerHasAce = true;
      }
    }
  }
  return val;
}

int readValue(String person) {
  Serial.write("CARD"); // Ask to Computer to read a card
  while(true){
    if (Serial.available() > 0) {
      int val = Serial.readStringUntil('\n').toInt();
      if (val >= 2 && val <= 11){
        if (val == 11) {
          val = handleAce(person, val);
        }
        return val;
      }
    }
    delay(200);
  }
}

void dealCard() {
  digitalWrite(dirPinRM,HIGH);
  digitalWrite(stepPinRM,LOW);
  delay(1300);

  digitalWrite(dirPinRM,LOW);
  digitalWrite(stepPinRM,HIGH);
  delay(2000);

  digitalWrite(dirPinRM,LOW);
```

```
  digitalWrite(stepPinRM,LOW);
}


boolean is_hitme() {
  Serial.write("Waiting for player...\n");
  while (true) {
    xValue = analogRead(joyX);
    yValue = analogRead(joyY);

    if (yValue > 900) {
      Serial.write("Stand\n");
      return false;
    } else if (yValue < 200) {
      Serial.write("Hit me!\n");
      return true;
    }
  }
}
```

Listing 2. Arduino component