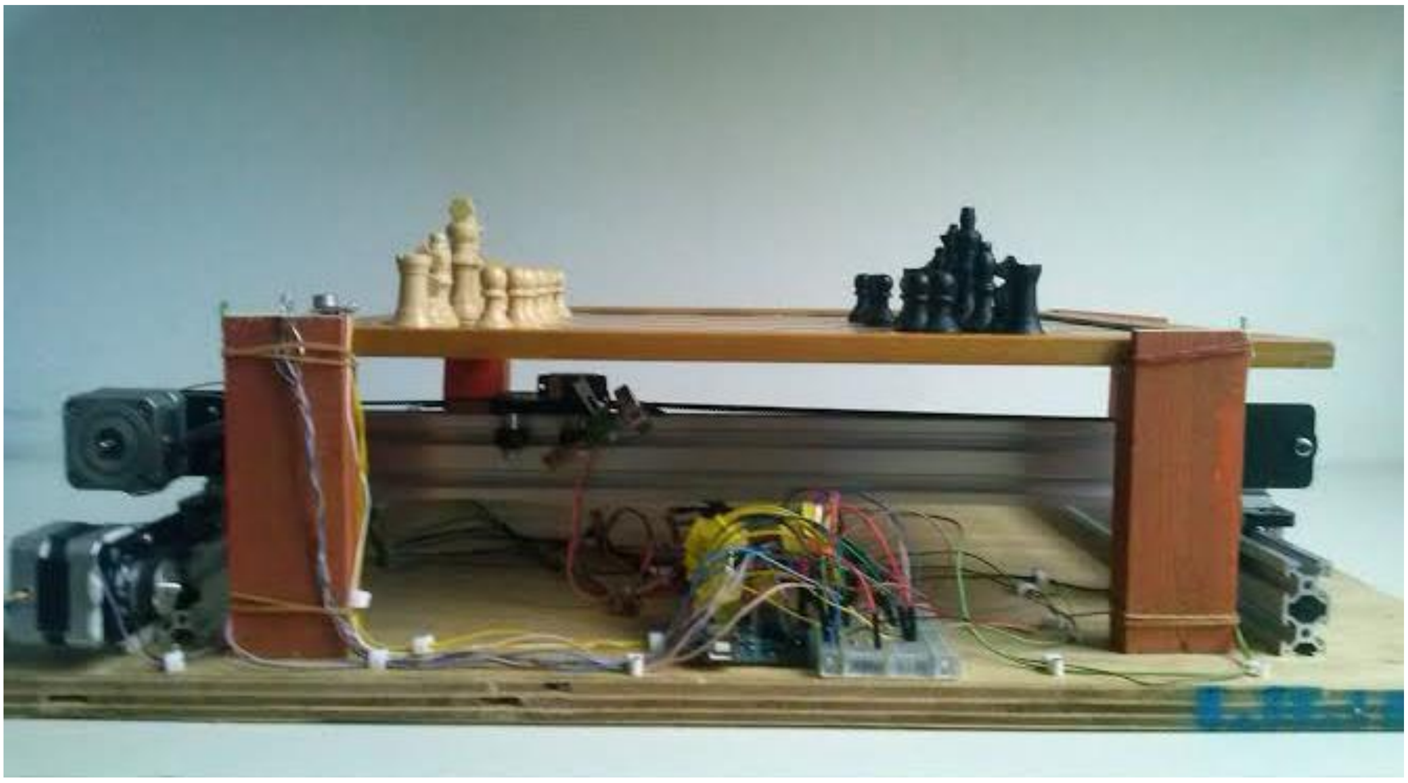
Chess Robot



Authors:	Roel Vos and Sjors Peterse
Class:	VWO 6
Year:	2013/2014
School:	Grotius College Delft
Subject:	Physics
Profile:	NT
Coordinator:	Guido Schrijvers

Preface

For our PWS, we teamed up because we have similar interests: mainly electronics, programming, maths and physics. We chose physics as profile subject, because even though we like mathematics, we preferred to do something practical. Once we decided on that, the choice for Mr Schrijvers to be our guiding teacher was quickly made. He had been our tutor and therefore we knew him pretty well.

Our first idea was making a compound bow. This is a modern bow which uses a complex system of wheels and multiple strings in order to aid the shooter. Unfortunately, it turned out we weren't allowed to make a weapon, so we had to think of something else. Then we got the idea of 'building a robot'. It took some time before we decided what kind of robot it was going to be. During the first 'PWS-workday' we got the idea of making a robot that could play the piano. After some brainstorming, we decided against this idea because we were afraid it would be really expensive.

Then in a flash of inspiration, we got an ingenious idea: "let's build a chess robot!". Enthusiasm overwhelmed us. Roel plays chess every week, and Sjors just likes the game. In the next hour or so we came up with all kinds of plans for our robot.

As soon as we had a rough idea of how everything was going to work, we went to Mr Schrijvers for approval. Wouter Tholen happened to be present when we were explaining our plans to Mr Schrijvers , and he told us about someone who had done exactly what we had just described. That's how Ernst Kouwe entered the story.

It turned out Mr Kouwe never actually had the chance to realise his idea of building a chess robot. However, when we told him about our ideas, he immediately offered to help us. We had several meetings with him to discuss our progress and to ask him questions.

We would like to give special thanks to Ernst Kouwe. We appreciated his kind advice and him making time for our fruitful meetings. His knowledge of electronics and his letting us test and use his components were extremely valuable for our project.

Thanks also to Rob van der Ven for allowing us to work in his classroom on several occasions and assisting us when necessary. Thanks to Guido Schrijvers for his enthusiastic support of our project.

Table of contents

1. Introduction

In this paper we describe the process of designing, building and fine-tuning our chess robot. We wanted to build a device that could accurately and flawlessly move chess pieces. It had to be able to understand verbal commands given making it suitable for people with physical limitations.

In chapters 2 to 5, we describe the solutions we explored for the problems we came across while designing the robot. We will list several advantages and disadvantages of all the possibilities and explain why we made the choices we did. This way we will discuss two ways to move the chess pieces (chapter 2), several ways to take care of the movements in the x and the y direction (chapter 3), how to use magnetism to reliably move the chess pieces (chapter 4), and the 'brain' of the computer (chapter 5)

In chapters 6-8, we describe the process of building the robot. We explain the problems we faced during the construction and how we solved them. In the final chapter, chapter 9 we will explain the basic functioning of the software we have written to control the robot.

2. Sub-design: the general system

There are several ways for chess pieces to be moved by a robot. In this chapter we explore the possibilities and choose the best one. There were two ideas we considered: physically moving the pieces around or using a magnet to move the pieces from underneath.

2.1. Movement with a mechanical arm

Our first idea was creating a system with an arm to move the chess pieces. The arm moves over the board and descends above pieces to grab them and move them around. This system is basically the same system humans use to move pieces. This was the reason we came up with the idea: if a human can do it, so can a robot. Copying the style used in nature (in this case how a human arm does it) is the most logical approach for many things, and we could therefore find several examples of this system on the internet (see Figure 1).

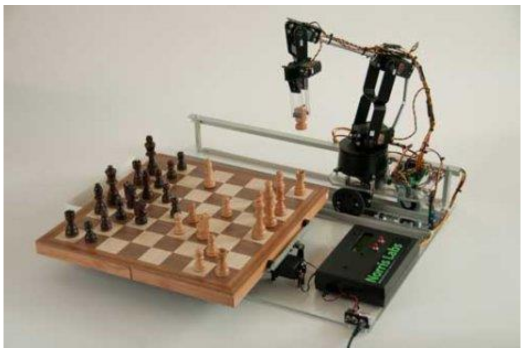


Figure 1. Example of a chess robot using an arm[1]

2.2. Movement using magnetism

We could think of several disadvantages of using an arm (see §2.3). That is why we came up with a different system: using a magnet to move the pieces from underneath the board. Every chess piece has a bit of iron attached to the bottom so that they could be moved by a magnet. The magnet moves in the x and y direction underneath the board and is switched on and off (in case of an electromagnet) or moves up and down (in case of a permanent magnet) to establish or break the connection with the pieces above it. Again there were several examples of a similar system on the internet (see Figure 2).

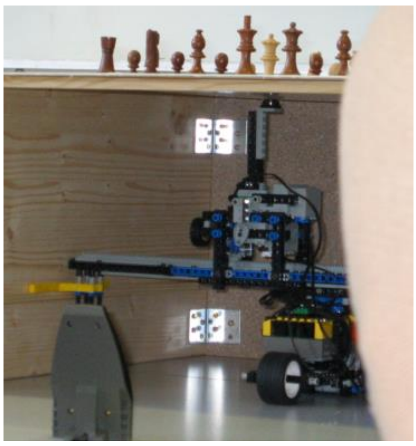


Figure 2. Example of a chess robot using magnetism[2]

2.3. Choice

The system with the arm has several disadvantages. Firstly, we thought that it would be really difficult to have the arm grab a piece without dropping it or accidently knocking over other pieces. If this happens, the system wouldn't be able to pick fallen pieces back up again. Using a magnet has a similar problem though: when moving a piece in between other pieces, there is a chance that the magnet accidently moves these pieces as well. However, with a good system, it should be possible to prevent this from happening, and even if it does happen, it might be possible to fix it.

Secondly, the entire system with the arm (xy-table, cables, pillars etc.) would have to be built above the board, and it would therefore block the players' view of the board. The system with the magnet, on the other hand, can be built entirely underneath the board, so the players are not bothered by it.

Furthermore, the system with the arm needs to be moved in an extra direction: also up and down. This means that this system would need one motor more, more materials and more work, especially since gravity makes movement in the z-direction difficult.

There is also a disadvantage of the system with the magnet. Everything is built under the board, so it will be hard to reach the system. If something goes wrong (like, for example, the cables get entangled) it will be hard to fix it as the entire board would have to be removed.

This small disadvantage does not weigh up against the many advantages of using magnetism, so we chose to build this system.

3. Sub-design: the xy-table

In this chapter we explain how we designed the xy-table, the system that moves the magnet in both directions underneath the board. We discuss three different designs for the xy-table. Each design consists of three major components (see Figure 3):

- A frame. This box-like structure is the skeleton of our robot. All the other parts are attached to the inside of the frame. It keeps everything together and gives shape to the robot.
- A 'moving beam' within the frame, below the chess board. It can move up and down the y-direction (see Figure 3).
- A system that moves the magnet across the moving beam, in the x-direction (see Figure 3). We call this system the 'magnet mover'.

Making a number of designs of the xy-frame and choosing between the resulting designs actually turned out to be the most difficult part of the project. The availability of the materials was important as well as the cost of the materials, as this would determine most of the project's expenses. We couldn't really test any of our ideas because we would have to buy the materials first. Therefore, we had to compare the advantages and disadvantages of the several possibilities solely on the basis of data from internet sources. As already mentioned, there were three ideas we seriously considered. Every new idea solved a major problem we expected to get with the previous idea.

3.1. Wood

First we considered making most of the system out of wood. The frame is a box (see Figure 3) that consists of four sides shaped in such a way that they can be put together easily (see Figure 4).

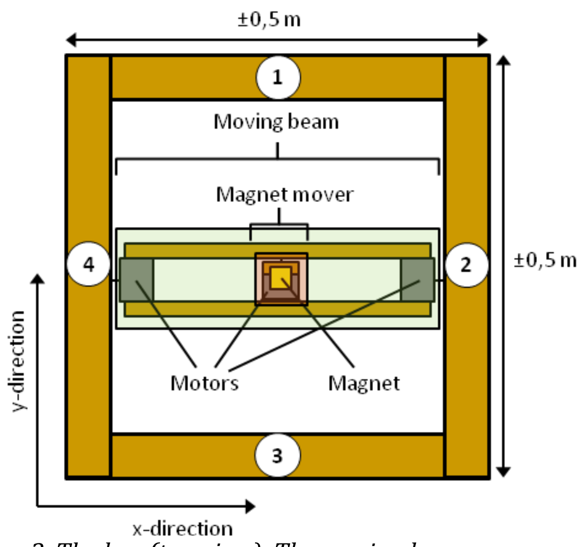


Figure 3. The box (top view). The moving beam moves up and down in the y-direction and the magnet mover moves left and right in the x-direction.

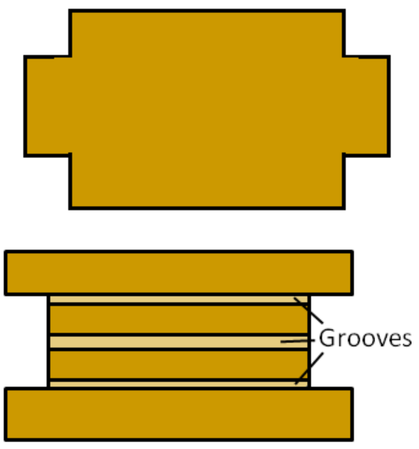


Figure 4. Top: front view of sides 1 and 3. Bottom: front view of side 2 and 4

Two opposite sides of the box are just normal pieces of wood (sides 1 and 3) while the other two sides (sides 2 and 4) are made up of layers (see Figure 5) to form a system of grooves (see Figure 6). This system seemed difficult to make at first, as we needed a way to make a groove through over half a meter of wood. The solution was actually pretty simple: divide the wooden board in different layers and glue them together (see Figure 5).

In both side 2 and 4, the grooves go all the way from side 1 to side 3. Inside groove 1 (see Figure 6) there is a wheel (or a rack and pinion) attached to a motor via a rod going through the opening (see Figure 7). We were not sure yet whether we would put a motor on both sides or we would put one of the rods in an empty socket. For now we will assume that we put a motor on both sides.

In between the motors, attached to both of them, is a wooden structure: the moving beam (see Figure 3 and Figure 8). When the motor runs, the wheel rolls in its groove, and the moving beam will move up and down in the y-direction (see Figure 3).

Inside grooves 2 and 3 (see Figure 6) there is a piece of wood (shaped like the groove) which slides through the groove when the system moves (see Figure 7). This piece is of wood is attached to the motor and makes sure the wheel turns while the moving beam does not.

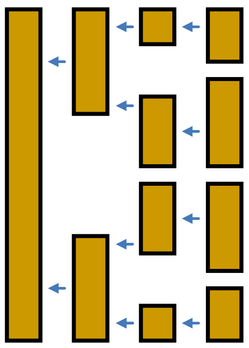


Figure 5. Cross section of side 2 and 4. The way the layers are built up

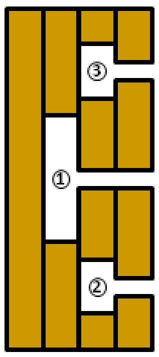


Figure 6. Cross section of side 2 and 4. overview of the grooves

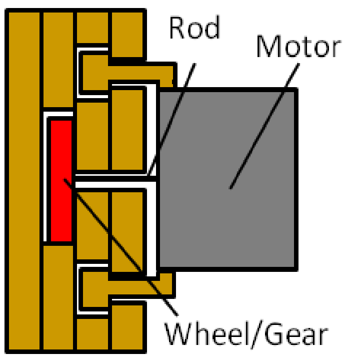


Figure 7. Cross section. The motor slides in the grooves

This system of grooves and motors deals with the movement in the y-direction. We also need movement in the x-direction. To achieve this, we wanted to build a similar system inside the moving beam (see Figure 8). The moving beam consists of two bars of wood that are attached to both motors (the wooden structures in the right and the left of Figure 9). These bars are layered in the same way as side 2 and 4 of the box (see Figure 6).

Again, there is one groove for a wheel or a gear to roll in (groove 1 Figure 6), and two grooves to keep the motor stable (groove 2 and 3, Figure 6). A motor is attached to one of the wheels, and the other wheel runs freely in an empty socket (see Figure 9). When the motor turns, the wheel rolls and the magnet mover moves in the x-direction (see Figure 3).

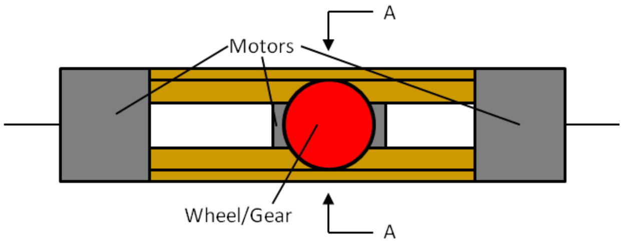


Figure 8. Side view of the moving beam. The front plate (see Figure 9) is not shown to improve clarity.

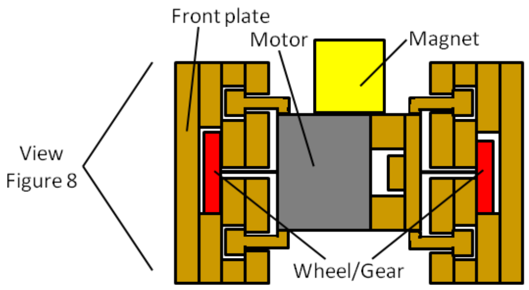


Figure 9. Cross section A-A (indicated in Figure 8) of the system in picture 7.

Although it seemed to be a good system at first, we were afraid that there would be a lot of friction when wood moves over wood. Maybe we could have solved this problem by using strong motors to compensate for the friction. However, we were afraid that the wheels would start slipping. After all, a wheel relies on friction to move, so it can start slipping if there are some forces working against it. These slips would cause the system to think that it is at a different location than it actually is, which could cause it to unsuccessfully move pieces or even move the wrong pieces.

We thought we could solve this problem by using a rack and pinion. The pinion can only move in steps, and it skipping an entire step is fairly unlikely. However, we came to the conclusion that a pinion and a matching rack were hard to get by, especially since there were some other requirements it had to meet (like the correct size etc.). We thought of making them ourselves out of wood, but we were afraid the pinion wouldn't perfectly run over the rack (because of teeth of different sizes or inconsistency in the "gaps" between the teeth). This could cause problems in the accuracy of the system.

3.2. Screw thread

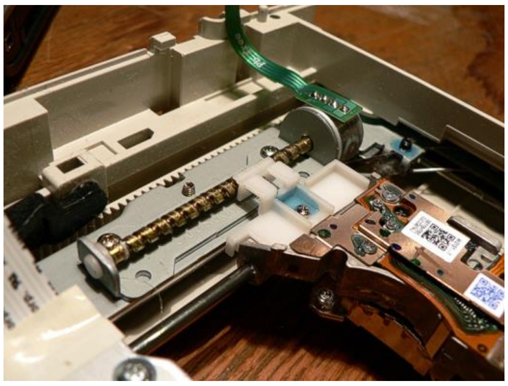


Figure 10[3]. CD driver. The screw thread in the CD driver moves the laser over the CD

We figured that wood would have some problems with accuracy and friction. So we came up with a different system: using a screw thread. We were inspired by a CD driver that was given to us by Mr Schrijvers. The CD driver uses a screw thread to move a laser over the CD (from the middle of the CD to the edge, while the CD rotates underneath it, see Figure 10). It moved using a motor and worked pretty well, so we figured we could use a similar system for our xy-table.

We didn't work this idea out as well as the previous one, because we couldn't really make any of the components ourselves, and we were therefore dependent on what we could buy. This limited how much we could design, because if we got into too much detail, we would need really specific parts, which are not for sale.

Our design was very similar to the system in the CD driver. Just like with the wood, the frame was a box-like structure inside which we would build the robot (see Figure 11). There are several iron rods and a motor attached to the sides of the box (see Figure 11). The motor shaft is attached to a screw thread, so that when the motor runs, the screw thread turns. On the screw thread, a nut is attached, and attached to the nut is the moving beam. When the motor runs, the screw thread turns, the nut moves (see Figure 12) and with it the moving beam.

There are two circular iron rods attached to the box as well (basically a screw thread without the screw part, so with a smooth surface). On these iron rods there is a spacer (like a nut with a smooth inside) which is attached to the moving beam in the same way as the nut. When the moving beam moves, these spacers slide over the iron rods. This prevents the moving beam from turning along with the screw; it keeps the moving beam horizontal. On the moving beam we wanted to make another similar system, with a screw thread and rods moving the magnet mover in the other direction.

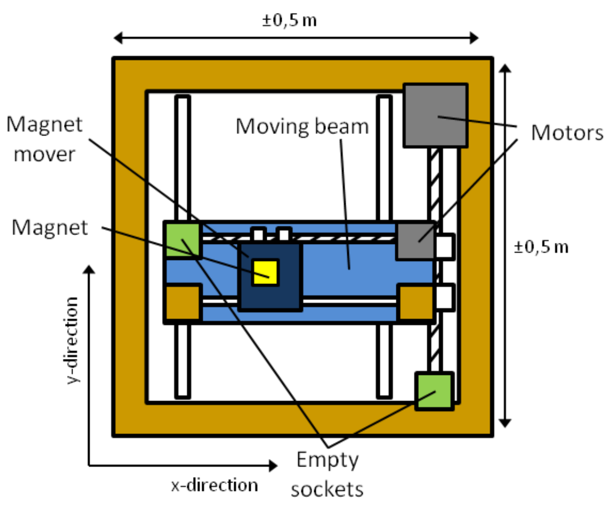


Figure 11. The system with the screw thread.

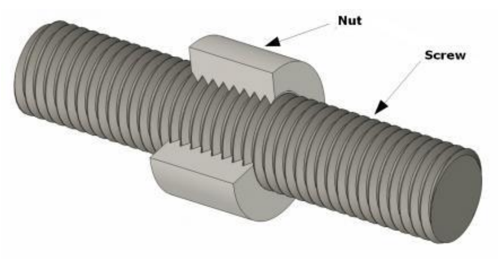


Figure 12. The nut moves over the screw when the screw turns.

A problem with this system was that we needed some pretty specific parts that we couldn't find anywhere. The screw thread couldn't be too heavy, while it still needed to be pretty long (about half a meter minimum) and it needed to have a very high bending stiffness. These were some criteria that were hard to combine. Another problem with this system was the nut. We needed a nut that fitted the screw thread. However, the main problem was that we couldn't think of a way to attach something to an ordinary nut. Glue probably wouldn't be able to handle the forces that it would need to handle, and there weren't a lot of other options. Thus we concluded that this system wasn't really usable.

3.3. V-slot

This time we decided to do things the other way around: have a look what was for sale, and then determine how we could use that in our system.

We came across a type of linear rail called V-slot (see Figure 13). It is a variation on the more regular T-slot (see Figure 14). T-slots are used in the construction of a large diversity of things, from frames of cupboards to robots. Several beams can easily be attached to each other by sliding nuts in the grooves.

The difference between the V-slot and the T-slot is the edge of the grooves. Whereas a T-slot has rounded edges (see Figure 14), a V-slot has a 45 degree slope inwards (see Figure 13). The advantage of this slope is that it can be used to roll wheels over the V-slot. When this is tried with the T-slot, the wheels will slip out of the rails often and they will wear out a lot more quickly.

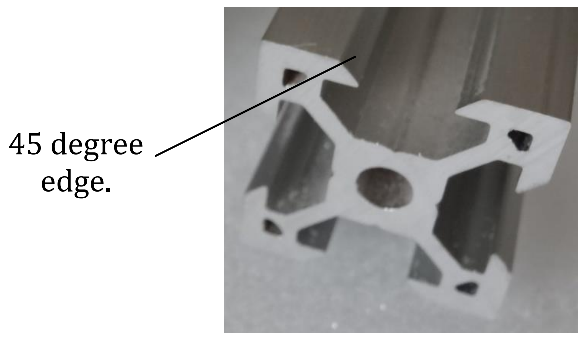


Figure 13[4]. A V-slot. The edges have a 45 degree slope so that a wheel can roll over the V-slot without slipping out.

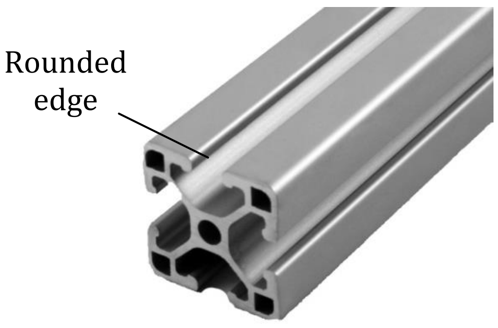


Figure 14[5]. A T-slot. The edges are rounded off instead of angled.

The OpenBuilds Part Store[7] (an internet shop) sells V-slots, along with all the other parts needed to make a linear actuator (a system that can make a linear motion). Using the parts from the OpenBuilds Part Store had one big advantage: they are designed to make linear actuators, and there were even tutorials of how to make one (see Figure 15). Therefore, we could be absolutely sure that this system would work. We wouldn't need to be afraid of friction, because the V-slots are really smooth and the wheels have ball bearings to make them run smoothly. Furthermore, the problem we had with the screw thread (the parts weren't for sale) was obviously no problem with this system: we had already found a shop that sold all the parts we needed.

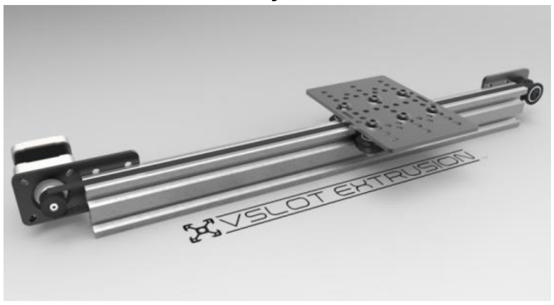


Figure 15[6]. An example of a linear actuator using V-slots.

There was one downside to using V-slots: everything was really expensive. Furthermore, the parts had to be shipped from America. This meant that the delivery fee was really high, and that it took very long for an order to be delivered. We decided to use V-slots anyway, because we figured that high-quality materials would inevitably also be expensive.

We wanted to make a detailed design before making an order. This was because we needed to know what components we needed. We didn't want to make an order twice, as we would have to pay the shipping fee again.

First we made a rough design. We needed three V-slots of half a meter each and a cart on each of them. Two of them are put on the sides, and the last one is attached to their carts (see Figure 16). This last V-slot is the moving beam.

Next, we started to work out the details by making a three dimensional drawing in Google SketchUp (see Figure 16) using models that were available online[8][9][10][11]. The advantage of doing it with a three dimensional drawing was that we had a good overview of which components we had already put in, and which components were still missing. If, for example, we forgot to attach the wheels to the carts, it would look weird, so we would notice the mistake quickly.

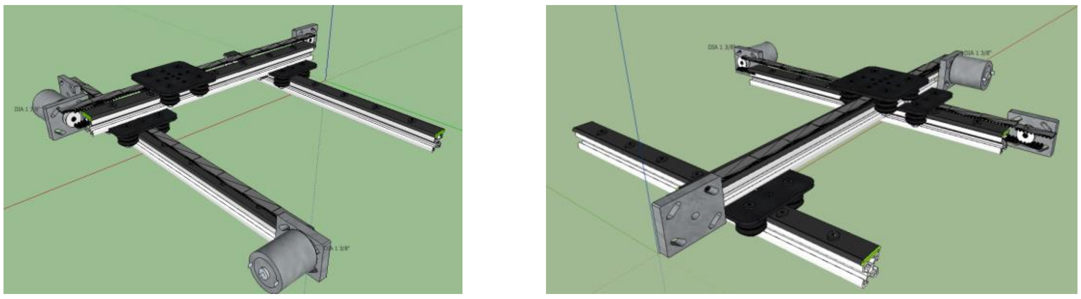


Figure 16. The SketchUp model seen from two different angles. We did not want to use the black strip on top of the V-slots (the OpenRail), but all the available SketchUp models had one.

We wanted to use belts and pulleys to move the carts. We expected that one motor would be enough to move the moving beam (middle V-slot). This meant that a system with belts and pulleys had to be attached to one of the static V-slots (y-movement) and the moving V-slot (x-movmement). Those two V-slots use the same system to move the carts.

To one side of the V-slot a motor is attached, and to the other side an idler pulley (a pulley that can turn freely, see Figure 16). The motor has a cylinder-shaped extension attached to its shaft, to give it more grip on the belt (see Figure 17). A belt is attached to one side of the cart, goes over the shaft of the motor, and inside the hollow V-slot (see Figure 17). It comes out at the other side, goes over the idler pulley, and back to the cart, where it is attached to its other side (see Figure 18). This way, the belt makes a full circle together with the cart. When the motor turns, the belt will move, pulling the cart over the V-slot.

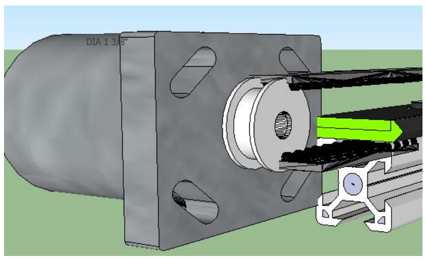


Figure 17. The motor in our SketchUp model. The belt (the black thing with ridges) is supposed to go around the white cylinder.

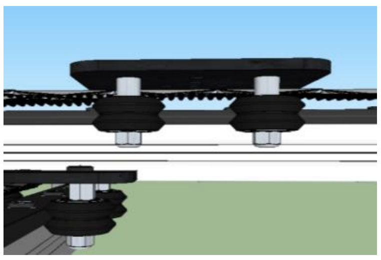


Figure 18. The cart riding over the V-slot in our SketchUp model. The belt should be attached to the cart on both sides.

4. Sub-design: magnetism

For the magnet we have explored two possibilities: an electromagnet that turns on and off and a permanent magnet that moves up and down.

The main advantage of an electromagnet is that it is possible to simply turn the power on and off to activate and deactivate it. A permanent magnet has to be moved up and down, for which we would need to build a separate system. Another advantage of an electromagnet is that we can be completely sure that it has no influence on the pieces when it is turned off; if there is no power, the entire magnetic field disappears. A permanent magnet, on the other hand, always has a field, and if it is moved away, the field will become weaker, but it will always be present.

4.1. Electromagnet

We preferred to use an electromagnet for the reasons mentioned above. But when we discussed this with Mr Kouwe, it turned out that he preferred a permanent magnet because he had heard that an electromagnet uses a lot of electricity. We tried to do research on this, but it was really difficult to find useful information. Part of the problem was that we didn't have any idea how strong 1 Tesla (magnetic flux density) is, and we therefore did not know how much Tesla we would need. But the main problem was that we could find next to no information about it. So we just decided to calculate the necessary current ourselves (see B of the attachments).

We concluded that the electricity use of an electromagnet was no problem: a 9-volt battery could supply the magnet for about four hours (when the robot is constantly active). Even so, the electromagnet that we bought turned out to be too weak to move the pieces. We suspect this was because it had too few windings, as that was the only requirement no information was given on, and all other conditions were met.

4.2. Permanent magnet

Using a permanent magnet is the most obvious choice, so we of course had already considered it. However we had some difficulties designing a system that could move the magnet up and down. At first we thought that we couldn't really use the rotation of a motor directly, because we pictured the movement as shown in Figure 19. Because of the circular motion, before really moving down, the magnet moves to the right. So it moves right while it still has an influence on the chess piece above it, moving the piece to the right as well. This means that the piece would be slightly off the centre of the square, which could cause problems later on

The solution for this actually turned out to be really simple. Instead of putting the magnet "on top" of the shaft (see Figure 19) we could just put it at the side, so that the direction of the movement would be straight down at first. However, it still had to face upwards, so we had to fix an "arm" to the axis so we could put the magnet on top of that (see Figure 20).

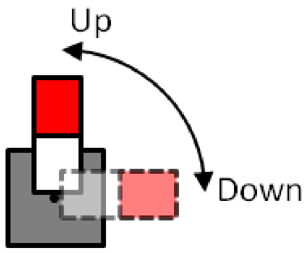


Figure 19. Our first idea, the problem was that it would move the piece as it went down

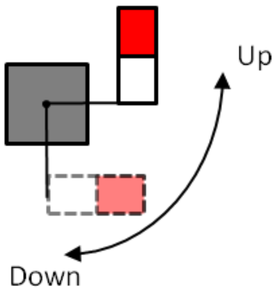


Figure 20. The solution. By putting the magnet on the side, it moves down directly

5. Sub-design: the brain of the robot

We needed something that could interpret the commands we gave it, and control the motors on the basis of those commands. Something like a little computer. So we chose to use a microcontroller called the Arduino. It controls our robot. It gets input from both the computer and the buttons on the board (see chapter 6, Table 2h), processes that data, and then outputs signals that control the motors.

5.1. Choice

There are a lot of alternatives to Arduino, like Raspberry Pi or BeagleBone. They all function more or less the same. We chose for Arduino because we already had one and as they are fairly expensive, we did not want to spend more money on another microcontroller.

5.2. The system

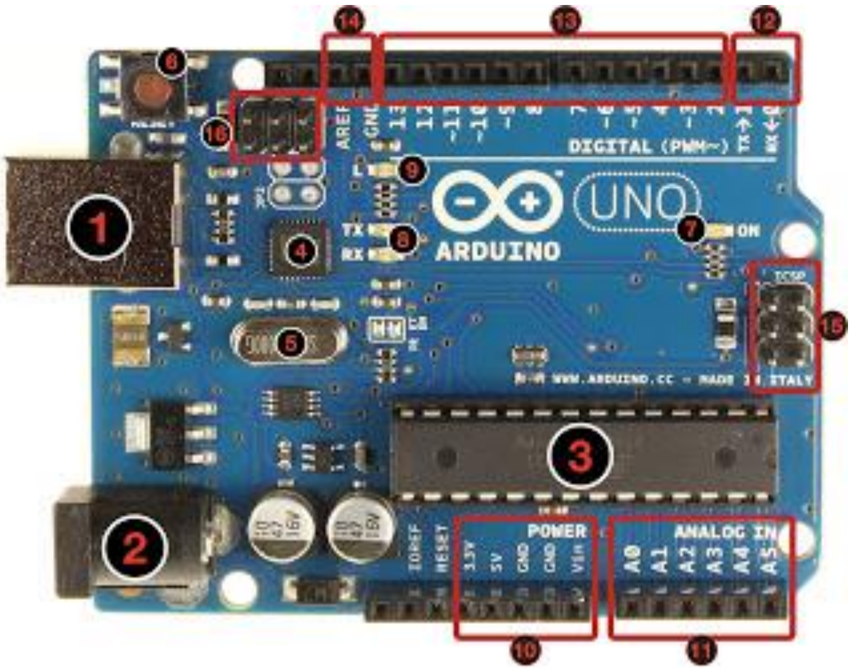


Figure 21. The Arduino. The different components have been numbered and are explained in the text below.[12]

This section gives an explanation of the functioning of the Arduino board itself (see Figure 21). The numbers in the text refer to the numbered components in Figure 21.

Number 1 is the USB port. A computer can be connected to the Arduino via this USB port. This connection can be used to upload programs or to power the Arduino. We also use it to communicate with the Arduino while the program is already running, as we need a computer to input the moves.

Number 2 is the power jack and can be connected to a wall socket via an adapter. We need it to run the two motors that move the xy-table (see Table 2c and d) as the USB port can't supply enough power. It can also be used to power the Arduino itself, if it needs to work without connecting a computer. However, our program requires connection with a computer anyway as we use it to input the moves.

Number 3 is the chip of the Arduino. This is where information is processed, data is stored and calculations are made. It executes the commands in the program that is uploaded via the USB port.

The thin black rectangles on the sides (numbers 10 to 14) are the pins. They can be used for in- and output. Here, cables can be plugged into the sockets and then Arduino can be told to either put a voltage on a certain pin or read the voltage that is on a pin. The pins at 12 and 13 are digital pins. When they are used as output they can either give a voltage of 5 Volt (high) or 0 Volt (low). When they are used as input, they can differ between two values: high (when the reading is above 3.3 Volt) or low (when the reading is below 3.3 Volt). The digital pins are the ones we use most (see E of the attachments).

There are two pins at 14: AREF and GND (written next to the pins, see *Figure 21*). GND stands for ground, this pin is always at 0 volt. All the voltages from the other pins (both input and output) are compared to ground (so 5 Volt on a pin means 5 Volt more than ground). Most, if not all of the other pins used as output end their circuit in ground (multiple wires can be connected to it using the breadboard, see Table 2g). AREF is a pin that we needed to use the microphone (see E of the attachments). However, the microphone didn't work (we used the laptop's built-in microphone instead, see chapter 8), so we are currently not using this pin.

The pins at 11 are analog pins. They are like the digital pins (12 and 13) except that instead of two values, they can both input and output 256 different values (between 0 and 5 Volt). These, too, we only needed for the microphone (see E of the attachments), so we are not using them either.

The pins at 10 supply power: they can handle much higher currents than the other pins. The two pins on the right of 10, Vin and GND (see Figure 21) are directly connected to the power source at number 2. Arduino cannot generate or work with the voltages necessary to run the motors (see Table 2c and d), but it can conduct them. These pins are therefore connected with the drivers, that can work with these high voltages (see §5.3). The three pins on the left of 10, 3.3V, 5V and GND (see Figure 21), are pins powered by the Arduino itself. This means that they can handle a lot less power than the other two power pins (still a lot more than the "normal" pins though). We use them to power the feedback buttons (see Table 2h) and the servo motor (for moving the magnet up and down), which do not require a very high voltage.

Number 6 is the reset button. It can be manually pressed, and makes the Arduino restart the entire program. The same happens when the power is cut off and plugged back in.

6. Components

6.1. OpenBuilds Part Store

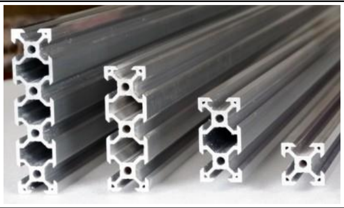
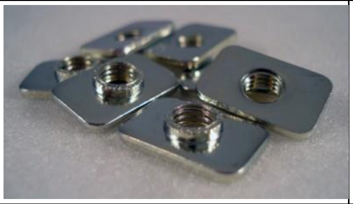
		
a) V-Slot[13] This is the linear rail system. We cut it up in 3 parts of half a meter, 1 of them (the mobile V-slot) is moved by Stepper motor 2 (see Table 2d), the other two (the static V-slots) have a fixed position on the base.	**b) Timing Pulley**[14] One of these is attached to the shaft of each Stepper motor (Table 2c and d), so that the motor can pull the timing belt (Table 1c).	**c) Timing Belt**[15] These are attached to the mini V plates, and allows the motor to move the mini V plate over the V-Slot.
d) Smooth Idler Pulley Kit[16] These are on the other side (the non-motor side) of the timing belt, so that it works like a conveyor belt.	**e) Motor Mount Plate**[17] These attach the motors to the V-slot.	**f) Idler Pulley Plate**[18] These attach the idler pulleys to the V-slot.
g) Mini V Plate[19] These move over the V-slot like a cart. We have 3 of them: 2 support the moving V-slot and the last one is attached to the magnet.	**h) Mini V Wheel Kit**[20] These wheels are attached to the mini V plate (in the large holes, see Table 1g) so that it can move over the V-slot.	**i) Belt Crimp Clamp**[21] These are used to attach the timing belts to the mini V plates.
		
j) Eccentric Spacers[22] Attached to two wheels of a mini V plate to make sure that the wheels 'grab' the V-Slot tightly, but not too tightly.	**k) Tee Nuts**[23] These are designed to fit exactly in the V-Slot, and are used to attach a lot of other parts to the V-slot.	**l) M5 Screws**[24] These are used to keep a lot of components together.

Table 1. The parts we got from the OpenBuilds Part Store.

6.2. Electronics

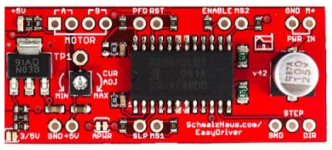
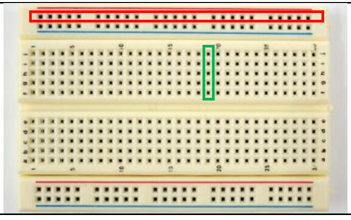
		
a) Arduino Uno[25] This is the brain of the robot. It determines which motor runs, and for how long. See Chapter 5 for more information.	**b) Stepper motor driver**[26] As the Arduino cannot handle the voltage necessary to drive the stepper motors, two of these drivers (one per motor) are needed. At the signal of the Arduino it lets current through from another power source.	
c) Stepper motor 1[27] This is the weaker one of the two Stepper motors. It was given to us by Mr Kouwe, and it moves the Mini V Plate (see Table 1g) with the magnet only. It will be referred to as “the old motor”.	**d) Stepper motor 2**[28] This Stepper motor had to be stronger than Stepper motor 1, because it has to carry a lot more load (it moves half a meter of V-slot, see Table 1a). It will be referred to as “the new motor”.	**e) Servo**[29] This is a very low-power motor that only puts the magnet up and down, and can be powered by Arduino directly.
		
f) Wires These wires connect all the electronic to the pins of the Arduino (see chapter 5).	**g) Breadboard**[30] The Breadboard can connect several wires. In the middle two blocks of plugs, the wires can be connected vertically (green), and at the sides the wires can be connected horizontally (red).	**h) Button**[31] Two of these serve as a reference point for the Arduino, so that it can determine the magnet’s position.
i) Adapter The stepper motors can´t be powered directly by Arduino, so we need this old router adapter to power them.	**j) Regular LED** We use two of these regular LEDs to indicate whose turn it is.	**k) RGB LED** The RGB LED can be different colours: red, green, blue and combinations of those. It is used to communicate with the users (see chapter 8).

Table 2. The electronics.

6.3. Miscellaneous

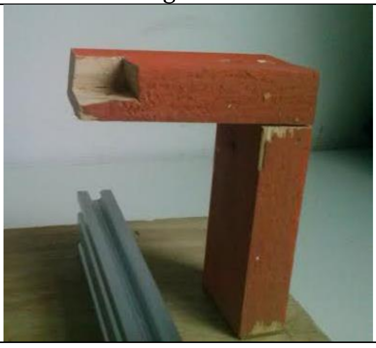
a) Chess pieces We used small, wooden chess pieces. They are light, so they can be moved easily. All the pieces contain a nail in the centre to make them magnetisable.	**b) Magnet** This is a very strong magnet with a hole with screw thread in the back. It was given to us by Mr Van der Ven.	**c) Pillars** Two corners of the chess board are supported by these pillars. We sawed out a corner of the pillar, to keep the chess board in place.
		
d) Diving boards Two corners of the chess board are supported by diving boards, because normal pillars would be hit by the moving V-slot. They are normal pillars with a piece of wood attached to the top, in order to reach the board.	**e) Base** The base is a wooden rectangle of about 60 cm by 60 cm. All the other components are attached to the base, and it therefore keeps everything together.	**f) Angle bracket** We used these brackets to put the drivers (see Table 2b) on their sides.

Table 3. all the other components.

7. Realizing and testing design: construction

This chapter describes the actual construction of the separate elements of our chess robot. The testing of these elements has not always been described as many of them worked immediately.

7.1. The mechanics

The first step was to build the xy-table. Basically all we needed for this were the parts of the OpenBuilds Part Store (see Table 1). However, we took several weeks between the decision that we were going to use the parts from the OpenBuilds Part Store and actually ordering them. This was partly because everything was really expensive (in total about €200, see D of the attachments), so we didn't want to order anything unnecessary. But the main reason was that the items had to be shipped from America. This costs about €30 for only a few parts, and the price goes up really slowly if for more (about €40 for everything we needed). This means that forgetting something would cost us about €30, so we wanted to be absolutely sure we ordered everything we needed.

Figure 22. The box in which we received the components

Figure 23. The separate parts that were in the box (Figure 22)

When our order arrived (see Figure 22) there was a nasty surprise waiting for us: we had to pay another €34 import rights (see D of the attachments), while we had already paid €40 shipping costs. On top of that, the Tee Nuts (see chapter 6, Table 1k) we had ordered happened to be out of stock, so they hadn't sent them. Also, it turned out that there were some parts we had forgotten to order. This was our own fault, of course. We decided to write a letter of complaint (see C of the attachments) in which we requested them to send us the parts we were missing, including those we had forgotten to order, for free as a compensation for the import rights. They granted our request.

A while later, the rest of the parts arrived. At this point we had all we needed for the xy-table, so we could finally start building.

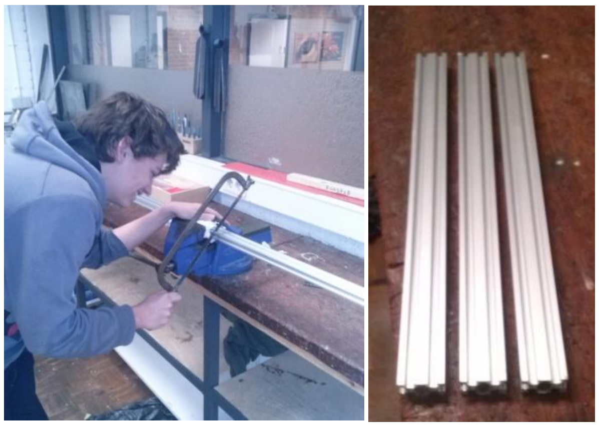


Figure 24. The aluminium V-slot had to be sawn in three parts

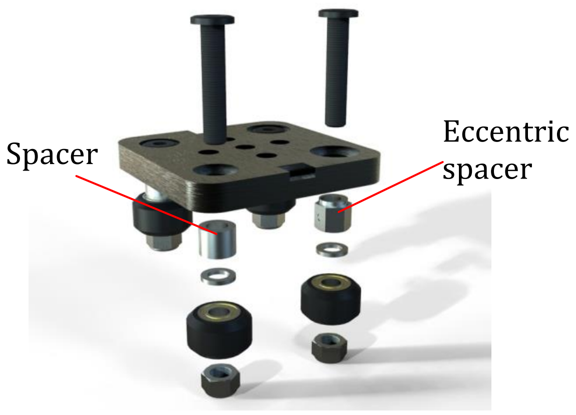


Figure 25.[32] *Attaching the wheels. Two wheels of each Mini V Plate have an eccentric spacer instead of a normal spacer.*

The first thing we needed to do was cut the aluminium V-slot (see Table 1a) in pieces. We received it as one 1,5 meter long part, while we needed 3 parts. So we had to saw it in three pieces of half a meter first (see Figure 24). The next step was putting the wheels (Table 1h) and the Mini V Plates (Table 1g) together (see Figure 25). We then slid the Mini V Plates on the V-slot. Using the eccentric spacers (see Table 1j), we made sure they rode smoothly without being able to wobble even the slightest bit. After this we had three, half a meter long V-slots with a cart riding on each.

To realise movement in two directions we used Tee Nuts (see Table 1k) to fix one V-slot to the top of the carts of the other two (see Figure 26). At this point, the middle V-slot could be moved back and forth over the other two, and the cart on the middle V-slot could be moved left and right. This way the middle Mini V Plate could reach every position inside the two dimensional system. However, they still had to be moved manually, and we wanted them to move using motors.

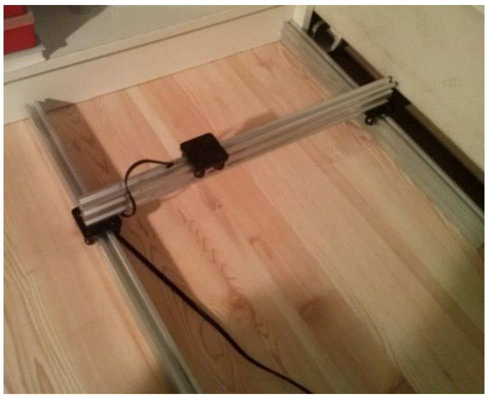


Figure 26. The xy-table. The cart in the middle can be moved manually to all positions within the two dimensional table. The black line is the timing belt (see Table 1c). More about the attachment of the timing belt in §7.2

7.2. The electronics

We needed two motors to carry out all necessary movements: one for the moving beam and one for the magnet mover (see §3.3). A third motor was needed to rotate the magnet (see §4.2). We had already done a lot of research on motors before. There are basically two motors that can be used with the Arduino: Servo motors and Stepper motors. The main advantage of a Servo motor is that it has a feedback system, while a Stepper motors does not[33]. This means that, if due to an external force a Servo motor skips a few steps, it will know that it hasn't moved the past few steps, so it will do a few extra steps as compensation. A Stepper motor, on the other hand, will think it is further than it actually is, and it will stop before it reaches its destination.

However, a Servo motor also has a major disadvantage: they can only rotate in 180 degrees, and they need to be physically and electrically modified in order to be able to rotate in 360 degrees. Stepper motors have no such limit[33]. This made the choice of using a Stepper motor fairly simple. A 180 degrees movement is not enough to pull in half a meter of belt (or we would need a gear with a circumference of a meter), and this was necessary to move a cart to the other side of the V-slot.

However, we still didn't know what to buy as there are quite a lot of different Stepper motors, and we didn't really know how to compare them without being able to test them. So we made an appointment with Mr Kouwe, as he has quite a lot of stuff like this. He brought several Stepper motors to see which one was best. One of them happened to fit perfectly on our Motor Mount Plates (see Table 1e). Mr Kouwe gave us this motor, and we could keep it if it turned out to be useful, which it did (see Table 2c).

There was still quite a lot of stuff we needed before the xy-table would be able to move by itself. We had one motor, but we still needed a second one in order to move in both directions. Furthermore, the motor we already had didn't work yet, because the Arduino can't supply the power necessary to run these motors. To make it work, we needed to buy a Stepper motor driver (see Table 2b) and we needed to have a power source.

Getting a second motor was fairly easy. We now knew more or less what we wanted, and Mr Kouwe had told us where he got his motor from: Floris[34], an online electronics shop. We ordered another motor from this shop which was like the one we already had, but slightly different (see Table 2d). The main reason we chose a different one is that the wires of the one we already had are a little short. One of the motors will be attached to the moving V-slot, and therefore needs quite long wires: it needs to be able to reach the driver from both sides. The wires of the new one are much longer.

Figure 27. An old PC. The box in the front is it's adapter.

As for the power source, Mr Kouwe advised us to use an old PC adapter. No sooner said than done, we got a PC adapter (see Figure 27). However, when we tried to make it work, no current came out for an unknown reason, so we assumed it was broken. As we started looking for an alternative we came across an old adapter that belonged to a router which was no longer in use (see Table 2i). Instead of wires, it has just one plug, which can be plugged directly in the Arduino. An advantage of this is that the computer can be unplugged, without switching off the Arduino. However, we still need a computer to enter the moves.

The only thing we still needed were two Stepper motor drivers. We found satisfying ones at Floris.cc. We ordered them, and when they arrived, we could finally start to wire things up.

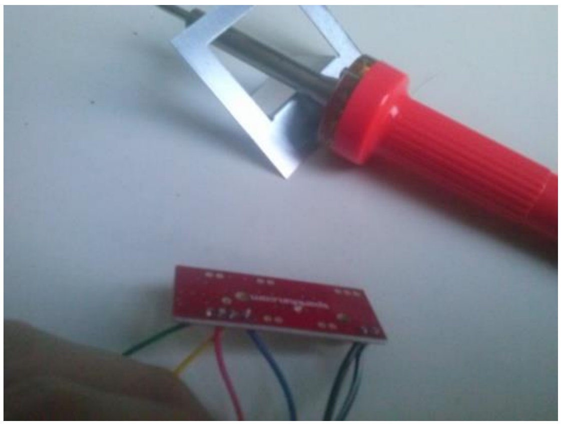


Figure 28. Attempting to solder wires to the drivers. It is hard to see, but the little bits of solder don't look like the little volcanoes they should look like at all.

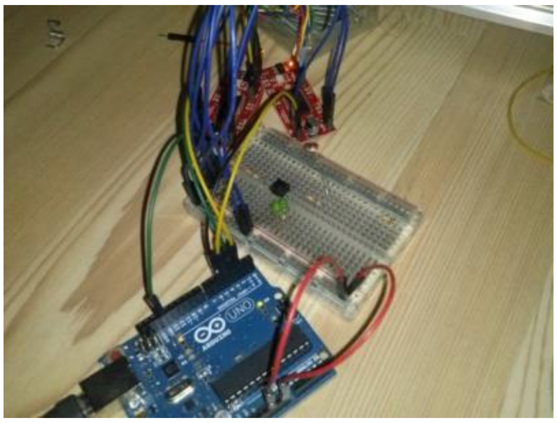


Figure 29. The wiring of the motors. The white object is the breadboard (Table 2g). The blue one is the Arduino (Table 2a) and the red ones in the back are the drivers (Table 2b)

We could just plug the wires in the right holes of the Arduino and the breadboard. But the drivers did not have pins in which we could plug cables so easily. Instead, they just had holes with conducting edges. The wires had to be soldered inside those holes. When we tried this ourselves, it looked like the wires could come lose any moment (see Figure 28). We asked Mr Kouwe to teach us the art of soldering, but after showing us the basics, he ended up doing it for us. When the soldering was done, we could finish the wiring of the motors (see Figure 29 and E of the attachments).

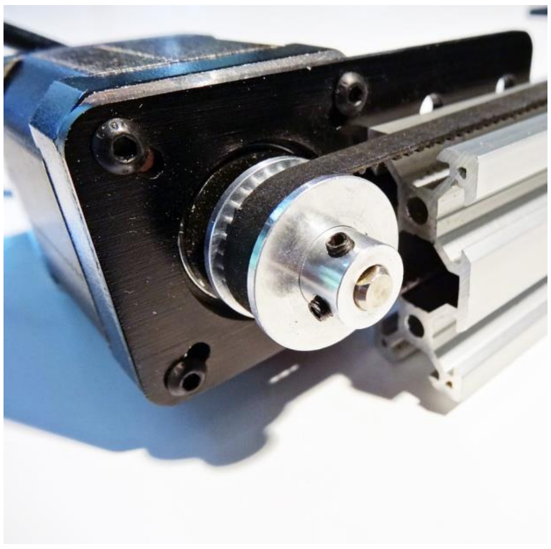


Figure 30[35]. The motor attached to the motor mount plate. We only needed two holes to keep the motor tightly attached.

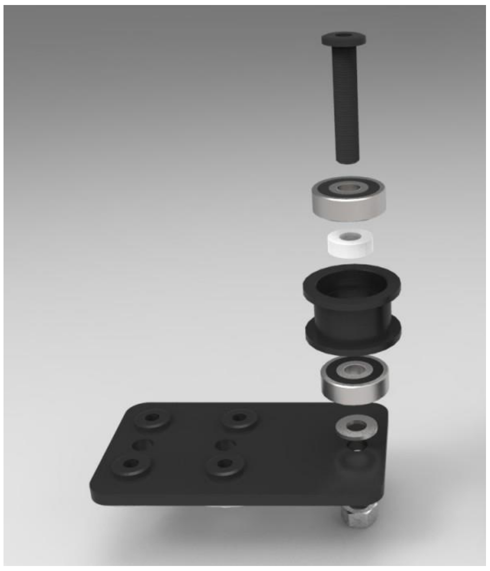


Figure 31[36]. Attaching the idler pulley. The black wheel is the one the belt runs over. The rest are spacers and ball bearings to make the pulley turn as smooth as possible.

Next, we put the motors and the idler pulleys (see Table 1d) in place. For this, we needed the motor mount plates (Table 1e) and the idler pulley plates (Table 1f). They both have three pairs of holes on one side to attach them to the V-slot with tee nuts and screws. However, the other side is different. The motor mount plate has one big hole to put the axis of the motor through, and 4 smaller holes on the sides to put screws through (see Figure 30). The idler pulley plate only has one small hole as it only needs one screw to attach it (see Figure 31).

To the moving V-slot, we attached a motor mount plate with the new motor (because of the longer wires) to one side, and an idler pulley plate with an idler pulley to the other. Next we attached the old motor and an idler pulley to one of the static V-slots. The only thing still necessary before the robot could move by itself were the timing belts (see Table 1c). But before we could attach those, we needed to put the timing pulleys (see Table 1b) on the axis of the motors.

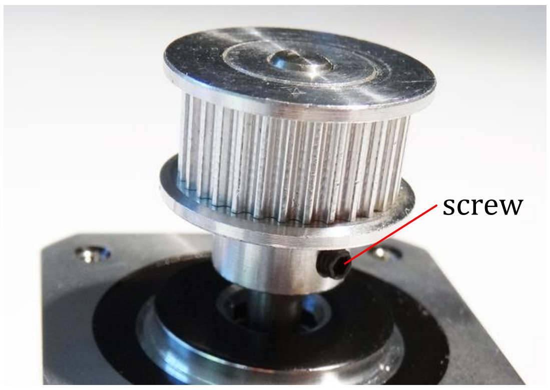


Figure 32[35]. The timing pulleys. The screws in the pulley are used to make sure the pulley can't slip off the axis of the motor.

The pulleys had two holes with really small screws in them. We had to fasten these screws in order keep the pulley from sliding. The screws didn't have a head, but instead they had a hexagonal socket in the screw itself. We went to buy a hex key, as the ones we had weren't nearly small enough. Fortunately, the smallest hex key that was for sale was small enough. With it, we managed to fasten all the Timing Pulleys.

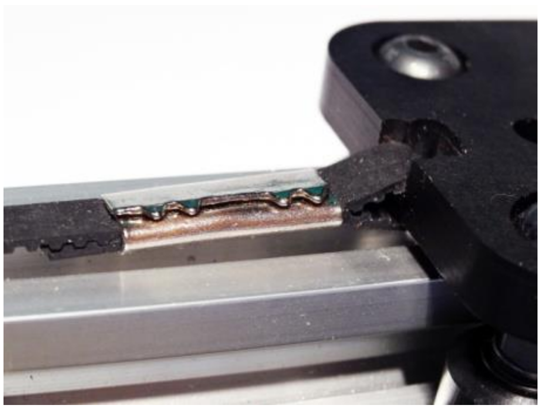


Figure 33[37]. The belt is attached using a belt crimp clamp. It loops through the hole in the mini V plate and the two ends of belt are kept in place by a belt crimp clamp

First, we attached the belt to one side the mini V plate using a belt crimp clamp (see Figure 33). Then we put the belt over the timing pulley and through the V-slot (they are hollow). From the other side of the V-slot, we took the belt and put it over the idler pulley, and through the hole on the other side of the mini V plate. We pulled it really tight and attached a belt crimp clamp to keep it in place. After cutting off the surplus belt, we repeated this process for the other V-slot with a motor.

Finally, we could test the motors with the xy-table. It turned out that the old motor was not strong enough to move the entire half a meter of V-slot. Fortunately, the new motor (the one we had bought because of its longer wires) was strong enough to do this. So we swapped the two motors. The disadvantage of this was that the motor with the short wires was now moving. But we were lucky again: when we put the driver in the middle, the wires were just long enough to reach the driver from both sides of the V-slot.

After letting the motors run for a while, we realised that there was another problem. The new motor vibrated heavily, and the motor mount plate and the entire V-slot vibrated along with it. This made a lot of noise, which got really annoying after a while. We stuffed some foam between the motor and the mounting plate to absorb the vibration (see Figure 34). This actually worked a lot better than expected: the Motor Mount Plate and the V-slot stopped vibrating completely, and the noise died away.

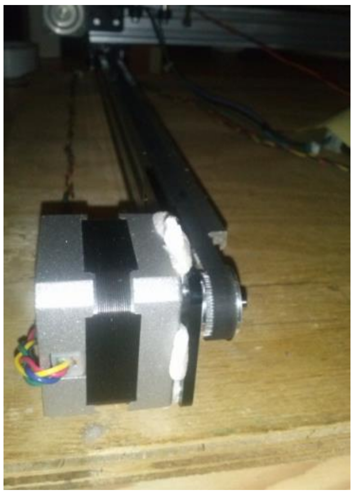


Figure 34. The new motor. We stuffed some foam between the motor and the motor mount plate, effectively stopping the noise.

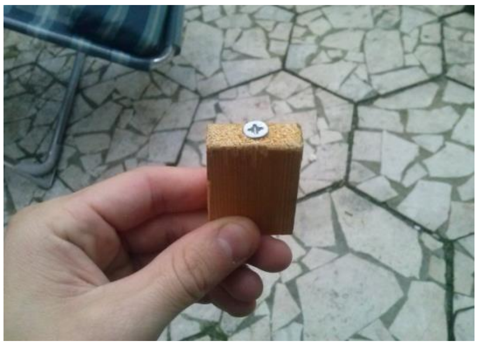


Figure 35. A dummy chess piece. The magnet had to be able to move this piece of wood.

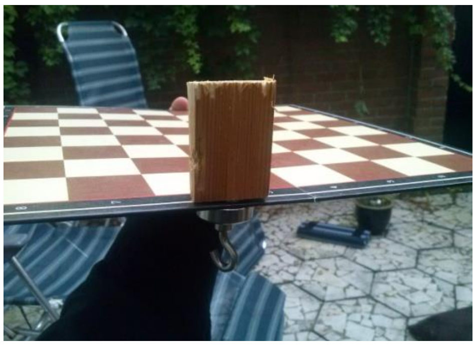


Figure 36. Testing the magnet. It hangs beneath the board using magnetism only

At this point we had an xy-table that could carry out movements by itself. However, we did not have anything that had to be moved yet, so the next step was attaching the magnet. However, the electromagnet we had bought turned out to be far too weak to move a dummy chess piece (see Figure 35). We didn't want to waste money by buying another weak magnet, so we went to Mr Van der Ven (our technology teacher) to see if he could help us. He had several permanent magnets, and we could borrow them or even keep them if they turned out to be useful. When we tested them, only one was strong enough (see Figure 36). It was a small, round magnet with a hole with screw thread in the back (see Table 3b).

Originally we had decided that we needed an electromagnet, but now the only usable magnet we had was a permanent magnet (see Table 3b). We had just figured out how to solve the problem that a permanent magnet would move pieces as it went down (see §4.2). Furthermore, we already possessed a motor that was perfect for this task(see Table 2e). This was a weak Servo motor (so it had 180 degrees rotation limit). The motor was weak, but moving a small magnet does not require a lot of power. Furthermore, the magnet only needs to be rotated for about 60 degrees, so the 180 degrees rotation limit is no problem. On top of that, the Servo motor could be powered directly by the Arduino, so we didn't need some complex system with drivers (see E of the attachments).

Now there was no real reason left not to use the permanent magnet we got from Mr Van der Ven. We put a screw in the back of the magnet, to make it easier to attach to the arm of the motor. Somehow we managed to attach the magnet using rubber bands only (see Figure 37). Next we used double sided tape to glue the servo motor to the Mini V Plate on the middle V-slot (so the Mini V Plate that could move in all directions, see Figure 37). Finally, we had a robot that could move a magnet to all the positions in the two dimensional frame. However, we still needed something like a box to screw everything to and to put the chessboard on.

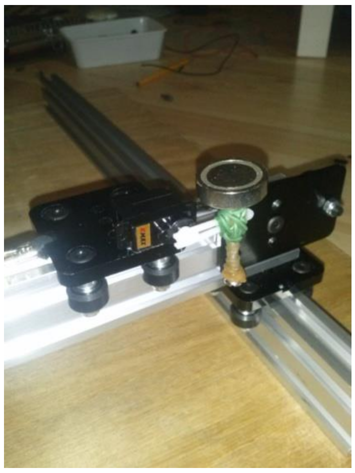


Figure 37. Attaching the magnet. The servo is glued to the Mini V Plate and the Magnet is attached to the arm of the motor using elastic bands.

7.3. The frame

It was really important that the two V-slots on the sides were completely unmovable. Until now we had just kept them in place manually. Next to that, all the electronics (Arduino, bread board, drivers) were only kept together with the wires. If we wanted to move our robot, the Arduino, the drivers or the bread board could drop, and we would have to plug the cables all over again. Therefore, we needed some sort of frame to which we could nail/screw everything.

At first we considered making an entire box like the one we wanted when we were still thinking of making the robot out of wood (see §3.1). However, we came to the conclusion that this was not necessary. Instead we chose to just put a wooden plate underneath, and several pillars on the sides to put the chessboard on.

So we sawed off a wooden rectangle (from here on called the base) that was about 60 centimeters long and wide. Next we screwed the two static V-slots thightly to the base using Tee Nuts (see Table 1k). We used the adhesive bottom of the Breadboard to attach it to the base. The Arduino had holes, through which we could put nails. We put both the Arduino and the Breadboard on the side of the base (see Figure 39) so that the mobile V-slot wouldn't move over them (as the wires would be hit by it). However, the drivers had to be reached by the motors, so we had to put them more towards the middle (see Figure 39). To solve the problem of the wires sticking out, we put the drivers on their side (see Figure 38) using angle brackets (see Table 3f).

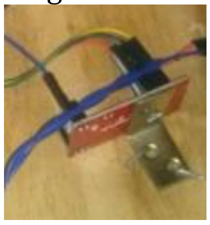


Figure 38.A driver on its side

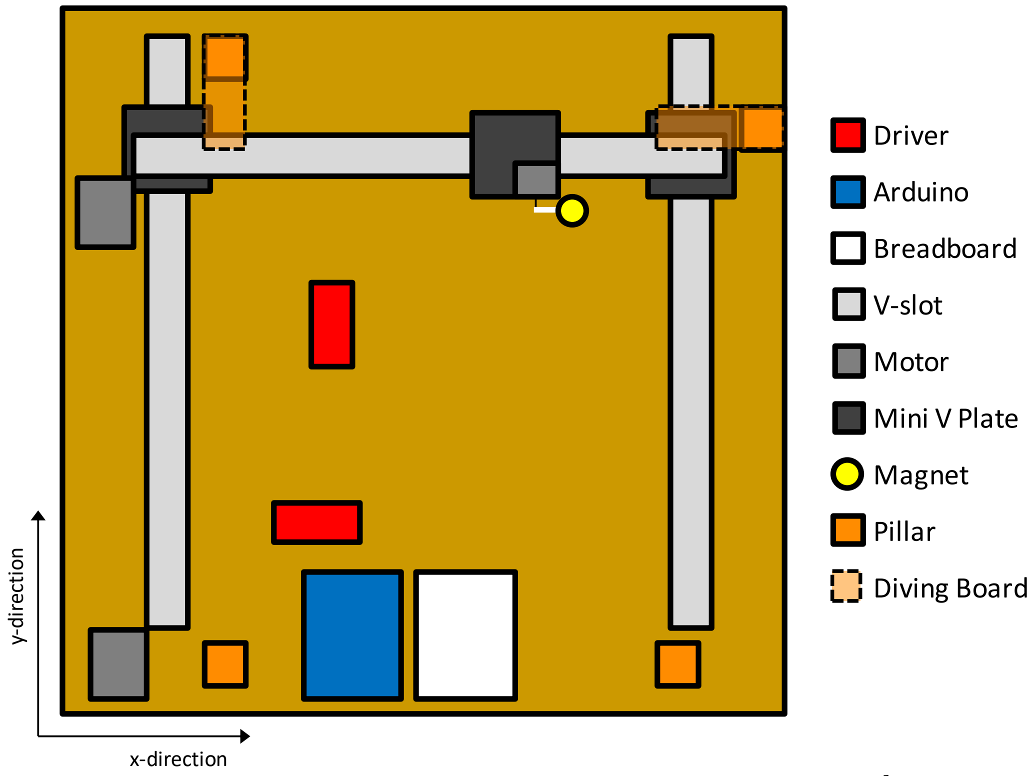


Figure 39. A ground plan of the robot. The diving boards are explained in Table 3d.The mobile V-slot is now almost in the position furthest away from the Arduino.

We needed pillars to put the chessboard on (see Table 3c). So we got one long beam and sawed off four equally long pieces. Two of them could be put directly underneath the chessboard (the two at the bottom of Figure 39). The other two, however, were a bit more difficult. If we put these pillars directly under the chessboard (so at the end of the diving boards in Figure 39), they would block the mobile V-slot in its movement. To solve this problem, we sawed off two more pieces from the beam, and glued them on top of the pillars (see Table 3d and Figure 41 on the right hand side in the back). This formed a structure that looked a little like a diving board, so we called these attachments to the pillars the diving boards.

We didn't just want the pillars to be support for the chessboard, but we also wanted them to keep the chess board in place. The chessboard is not allowed to move, because if it did, the robot would become inaccurate. To ensure that the board's position was fixed, we cut out corners of the normal pillars and the diving boards (see Figure 40). The chessboard rests in the gaps created by the cut out corners, and will be kept in place by the edges of the gaps (see Figure 41). We didn't want the pillars to move or to come lose, so we used both large screws and wood glue to attach the pillars to the base. And finally, our chess robot was more or less done. It could move the chess pieces by itself, and only some minor improvements would have to be done later on.

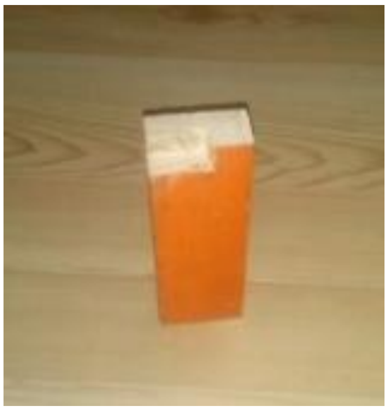


Figure 40. A pillar. A corner has been cut out to keep the chessboard in place

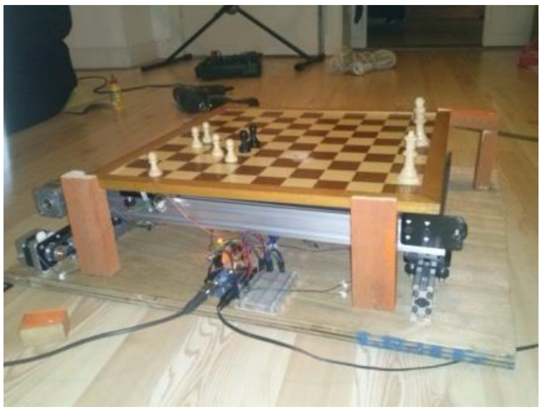


Figure 41. The chessboard on the pillars. The chess board cannot move at all, because it is stopped by the edges of the cut out the corners.

When we started programming we realised that there was a problem: we had nowhere to put the captured pieces. The chessboard we used had edges, so we couldn't move the pieces off the chessboard. Next to that, the V-slots were too short for the robot to move further than the edges of the chessboard. At first we thought we would need to use a smaller chessboard, but then we realised that there was a much simpler solution: we could just turn the chessboard around. On the backside there is draughts board (see Figure 41). This is basically the same as a chessboard, except that it has 100 squares (10 by 10) instead of 64 (8 by 8). This meant that if we used an 8 by 8 square on the draughts board as the chessboard, we would still have 36 squares left to put the captured pieces. There are only 32 chess pieces of which two (the kings) can't be captured, so 36 extra squares was enough to store the captured pieces.

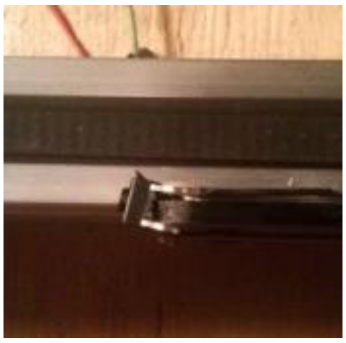


Figure 42. A feedback button attached to a Motor Mount Plate.

While programming, we thought that it may be handy if the Arduino had a way of determining that it is at its starting position. This was necessary because there is always a possibility that the motors miss a few steps for some reason. If it can go back to its starting position every now and then, we can prevent small deviations from piling up and becoming big deviations. So we glued a feedback button (see Table 2h) to both motor mount plates (see Figure 42). When a command is given where a piece has to be moved to the line closest to a feedback button, the motor responsible for moving in this direction will keep running until the mini V plate presses the button. This sends a signal to the Arduino, who then knows that the magnet is exactly at the last row. For the wiring of the buttons, see E of the attachments.

8. Realizing and testing design: voice control

Before we used voice control to control the chess pieces, we had to type in the commands. In the first working version, a piece could be moved by typing "e2e4". We could capture a piece by typing an 'x' in between the two squares, so "f5xe4". Later on, we made it remember the positions of all the pieces and had it check whether there was an enemy piece on the second square. So it was not necessary anymore to type an x in between the squares. The next program improvement was switching to voice control.

8.1. Voice control 1.0

We downloaded a program called 'BitVoicer'[38]. It is a voice recognition program that was designed to work together with an Arduino. The user has to specify beforehand which phrases the program should recognise, and some data for every possibility. When the program is activated, it will wait until it recognises one of these phrases. When this happens it sends the data from that phrase to the Arduino.

Figure 43. The microphone.

We had bought a microphone for the voice control (see Figure 43). It was a special microphone: it could be connected to the Arduino. The Arduino picks up the signals from the microphone and send them to the computer, where BitVoicer translates it. Unfortunately, the microphone did not work. We have no idea why, but BitVoicer never received any signal, no matter how much noise we made. Fortunately, the laptop itself turned out to have a pretty good microphone, and BitVoicer could also work with this microphone.

Our first version of voice control did exactly the same as our program without it, except that instead of typing the moves, we had to say them. At this point, there were still a lot of improvements to be made.

8.2. Voice control 2.0

We wanted to be able to leave out the first square in our commands, and instead name the piece that had to move. So instead of 'D3 to E5' we wanted it to understand 'knight to E5'. Doing this involved a lot of programming, because it needed to know what kind of moves each piece could do, as it had to be able to determine if a move was possible. Also, we wanted it to keep track of whose turn it was and reject all moves from the pieces of the wrong colour.

We thought that it might be handy if the robot was able to communicate with the users, so that they know when the robot rejects a move. For this we used an RGB LED (see chapter 6, Table 2k). This LED can become several different colours and by changing its colour, it can send messages (every colour has a meaning). Also, we wanted it to show whose turn it was at every moment. For this, we used two regular LEDs (see Table 2j), one for each player. When white's LED is on, it is white's turn and the same goes for black and its LED. This means that one of the LEDs is always on, and the other one is always off. For the wiring of the LEDs, see E of the attachments.

In the example 'knight to E5', the RGB will be off before anything is said. When the BitVoicer recognises the phrase, it will send a signal to the Arduino. The Arduino will look at all the squares a knight's jump away from 'E5' to see if there are any knights of the right colour. Possible fields are 'G4', 'G6', 'C4', 'C6', 'F5', 'F3', 'D5' and 'D3'. If there are no knights, it rejects the move and the RGB will turn red. If there are two knights, the RGB will turn blue and the player has to specify which knight he wants to move. If there is only one knight, the RGB will turn off, and the robot will move the piece.

8.3. Manual

When starting or restarting the Arduino, move the pieces to their starting positions. The white rooks should be at A1 and H1, and the black rooks at A8 and H8 (see the marks at the sides). The rest of the pieces follow from the potions of the rooks. The chessboard is marked with a black line. The squares outside this black line form the graveyard, where captured pieces move to. Do not move the pieces by hand during the game as Arduino will not know where they are.

8.3.1. Voice control in English

Types	**File**	**Rank**
Pawn	Alfa	One
Knight	Bravo	Two
Bishop	Charlie	Three
Rook	Delta	Four
Queen	Echo	Five
King	Foxtrot	Six
	Golf	Seven
	Hotel	Eight

Table 4. The commands for regular moves. We used the NATO phonetic alphabet because letters sound too much alike.

File	**Rank**
Nine	At
Colon	Question Mark

Table 5. The commands for moving captured pieces.

Colour	**Meaning**
Off	Listening, no commands received yet. Or: Input accepted. Executing move.
Green	Input accepted
Red	Move rejected: no pieces can move there.
Blue	Move rejected: two (or more) possibilities. Please name the square of origin, or a piece type in case of promotion.
Purple	Super User activated

Table 6. The colour codes of the RGB LED.

Regular moves:

1) Start by saying the type of piece you want to move. The RGB will turn green. If not, try to speak more clearly, or pronounce it differently.
2) Within five seconds (the green light will turn off if you are too late), say the file and rank you want your piece to move to (See Table 4). If your move is accepted the RGB will turn off and the move will be executed. If your move is refused the RGB will turn either red or blue (see Table 6).

Castling:
Say "castling long" for castling queenside, and "castling short" for castling kingside.

Control Z:
In the case that your move was misunderstood and a piece has been moved to the wrong square (or a wrong piece has been moved to the right field), simply say "Control Z", to undo the move. However, if you capture a piece and say "Control Z", then the captured piece stays where it is. It can be moved back using Super User.

Super User:
If you want to do a move that is against the chess rules, do the following:
1) Say "Super User". The RGB will turn purple.
2) Within five seconds, say the file and rank of the first square. The RGB will turn green.
3) Again within five seconds, say the file and rank of the second square. The RGB will turn off and the move will be executed.

Super User should not be used to capture a piece. You can, however, move captured pieces in the game again and vice versa (By saying "Super User" first). For the names of the captured files and ranks, see Table 5.

Calibrate:
If you noticed that one of the motors made a strange sound, it is possible that it got stuck for a while, and that the robot no longer knows where the magnet is. By saying "Calibrate", it will move back to starting position, and you can play on as usual.

8.3.1. Voice control in Dutch

It is possible to use Voice Control in Dutch. It has the same functions and the RGB colours mean the same, but the commands are all in Dutch, see Table 7.

Types	**File**	**Rank**	**Specials**
Pion	Anna	Éen	Korte rokade
Paard	Bernard	Twee	Lange rokade
Loper	Cornelis	Drie	Kalibreer
Toren	Dirk	Vier	Control Zet
Dame	Eduard	Vijf	Super gebruiker
Koning	Ferdinand	Zes	
	Gerard	Zeven	
	Hendrik	Acht	
	Apenstaartje	Negen	
	Vraagteken	Dubbele punt	

Table 7. The commands of the Dutch voice control

9. Realizing and testing design: the software

The Arduino is the brain of our robot (see chapter 5). However, it still needed to be told what to do. This could be done through writing a program on the computer and uploading it via the USB port. The Arduino will go methodically through the program and execute all the commands.

9.1. Programming stages:

When we started this project, we did not know exactly how far we would go with the chess rules. Best case scenario, we would include an artificial intelligence that could be played against.

Our first program did not think about the moves at all. Input had to be typed in the connected computer and it had to consist of two squares (for example: e2e4). All the robot could do was move the chess piece from the first square (e2) to the second square (e4). To capture a piece, we had to enter a separate move. First we had to type a move that moved the captured piece to the graveyard (we had also given these squares coordinates, see §8.3, the manual). Next we had to enter the actual move. Having to capture pieces like this is quite annoying, so the first improvement we made was that the robot would automatically capture a piece when we typed an x in between the moves. For example: when we typed in e2xe4, the robot first moved the piece at e4 to a square in the graveyard. Next, it moved the piece at e2 to e4.

Now, the robot was able to understand fairly normal chess notation. The only difference is that chess players usually leave out the first square and name the piece instead. So instead of e2e4, they usually say pawn to e4. However, for the robot to understand this notation, it would need to know the chess rules. This would be a lot of programming, so we decided to do the voice control first (see chapter 8). This resulted in voice control 1.0 (see §8.1).

After voice control 1.0 had been implemented, we worked on the chess rules. This resulted in 'Voice control 2.0' (see §8.2). After that, we started including the details of chess. Castling (a move involving both the king and a rook) was made possible, as well as Super User (enabling a player to ignore the chess rules when doing a move) and 'en passant', a special way of capturing a pawn with another pawn. Next, we added the possibility of undoing a move (Control Z).

9.2. The Result

The entire program is given on a USB stick (see A of the attachments). A short summary of the program is given in Figure 44.

```
1.  void serialEvent()    // when something has been heard
2.  {
3.          decoder();  // Translates the input
4.          if (flow  ==  1)    // If the input is a move, decoder sets flow to 1
5.          {
6.                  if (slain)   //checks if something is slain
7.                  {
8.                          capturerun();  // sets attacking piece on rest square
9.                          trash(x2, y2);  // moves slain piece to graveyard
10.                 }  // end if (slain)
11.                 zet();  // does the actual move (and in case of slain, the final
                    third part of the move)
13.         }  // end if (flow == 1)
14. }  // end serialEvent
```

Figure 44. The program. Each function refers to a separate part of code (except for the if-functions).

In the text below, we explain what happens in the most important part of our program (see Figure 44). The line numbers refer to the lines in Figure 44. We will go in a little more detail than the comments (in grey).

Every time BitVoicer (see chapter 8) recognizes a vocal command, it sends a signal to Arduino. When Arduino receives this signal, it will start the function serialEvent (line 1). The code beneath the functions in serialEvent is basically all that happens in our program.

Decoder() (line 3) is a function that translates the signals Arduino receives from BitVoicer into something the program can work with. It could be a pawn, knight, bishop, rook, queen, king, but also other commands like castling, control Z, Super User or calibrate (see §8.3). Decoder() also listens to the second part of the move ("Echo four") and translates that to something usable. When Decoder() has received an entire move, it will call the function possible() which checks whether the move is possible according to the chess rules, and if a piece is captured. If the move is according to the rules, the variable 'flow' will be set to 1. If something is captured, the variable 'slain' will become true, if not, it will become false.

Next, the program checks whether flow has been set to 1 (line 4). If this is not the case, because the command is invalid or incomplete, the program will immediately skip to line 13 and the serialEvent function will end. When the Arduino receives a signal again, it will restart serialEvent.

If flow had been set to 1, because a valid and complete command had been received, the program will run the code in between line 4 and line 13. First, the slain variable is checked (line 6). If this is true, capturerun() is executed (line 8). We thought it would seem more natural if an attacking piece moves towards the victim first, before capturing it. This is what capturerun() does. It moves the attacking piece to the square next to the victim (a temporary position, the 'rest square').

The function Trash(x2,y2) (line 9) moves the chess piece on coordinate (x2,y2) to the graveyard. (x1,y1) is be the beginning position of the attacking piece, and (x2,y2) is the ending position of that piece. The to be captured piece is always at the ending position (except in case of en passant). So when a piece is captured, it needs to be moved from (x2,y2) to the graveyard.

After that, the function if (slain) ends (line 10). So regardless of 'slain', the next function, zet(), will be executed. Zet()actually moves the piece to its ending position. If, as described before, a piece is captured and the attacking piece is put on a temporary position, zet() will move the attacking piece to its ending position (so x2,y2). If nothing was captured, zet() will move the piece directly to its ending position.

10. Conclusion

10.1. The Chess Robot

We successfully built a chess robot that can move chess pieces over the board, and can understand vocal commands. It knows the rules of chess (including all the “special” rules) and will reject any move that is not allowed (except when in Super User mode). Using LEDs the robot communicates with the users.

The chess pieces are moved around the chess board with a magnet beneath the board. This magnet is activated or de-activated by moving it up and down using the rotation of a motor.

The magnet is moved horizontally with an xy-table that can move the magnet into the two horizontal directions x and y. The brain of the robot is an Arduino. This Arduino receives its commands via a laptop. The laptop receives and translates the vocal commands into commands that can be understood by the Arduino.

The chess game is played on a draught board. A draught board has more squares than a chess board. The residual squares are used as graveyard for the captured chess pieces.

10.2. The process of designing, realising and optimising

During the design process, we have evaluated several designs for the xy-frame. We finally chose for using a V-slot system because this would be the most reliable solution.

We have considered two magnet systems: the electromagnet and the permanent magnet. At first, we wanted to use an electromagnet. We have determined the necessary strength of the electromagnet with calculations and bought one and we have tested it. However, it was not strong enough, probably because the number of windings was too low. After that, we tried out several permanent magnets and chose for a very strong one. This meant that we needed to construct a system to activate and de-activate the magnet by moving the magnet up and down. We did this by attaching the magnet to the shaft of a rotating motor.

We have developed the software gradually. We started with a relatively simple program that was able to follow the most straight forward commands given via the laptop. After that, we have implemented voice control. Lastly, we extended the program with chess rules and several special functions like a ‘Super-User’ function to ignore the chess rules and an ‘undo’ function in case the robot misinterprets a command.

10.3. Recommendations

However, there are still several improvements to be made before the robot is perfect. Most of these improvements are improvements in the software. For example, we want the Arduino to be able to recognise when one of the players is in check, and refuse all moves that do not solve this.

Furthermore, we want the Arduino to recognise checkmate and send a message when this is the case. Right now, it does not matter whether or not you are in check, you can still move any random piece. Another improvement that we are still working on is the promotion of a pawn when it reaches the end of the board. We want it to be physically replaced by a captured queen.

We want to improve the Control Z function as well. Right now, if this function is activated after a move where a piece was captured, the captured piece will not return automatically. We still need to employ Super User to do this. We want the control Z function to automatically return a captured piece to their original position.

Also, we would like to create a function that returns all the pieces to their starting position. This way, after a game is finished, the pieces will be returned by this single command. At the moment, Super User would need to be employed a lot of times to move all the pieces back. With all these improvements, the robot would be complete if two paralysed players would want to play chess against one another.

However, this is not very likely to happen, as there are not too many paralysed people. Even if there are two of them who want to play chess, they often can ask someone to move the pieces for them (as they usually have people taking care of them). A final improvement would therefore be implementing a chess engine (an artificial intelligence) in our robot. This way, the robot cannot just move the pieces, but also counter these moves. If one paralysed person wants to play chess, and there is no one to play with, he or she can play against the robot. However, writing a chess engine ourselves is far too difficult. It might be possible to use an existing chess engine in our program, but we have to somehow get it to communicate with the rest of our program. Also, the Arduino only has about 32kB storage, of which we already used about 10kB for our program. We are therefore restricted to a chess engine of maximum 22kB. These exist, there are even chess engines of less than 2kB. However, these programs are extremely compact and therefore difficult to understand. So at the moment, implementing a chess engine is too difficult for us.

11. Log

Date	Activity	Who	Duration
2 July 2014	Brainstorming	SR	2 hours
3 July	First topic	SR	2 hours
6 July	Working out design wood in detail	R	3 hours
9 July	Plan of action, contact Mr Kouwe, research motors	SR	2 hours
19 Aug	Making several designs, answering questions from Mr Kouwe	SR	1½ hours
20 Sept	Discussing our progress with Mr Kouwe at the Grotius College	SR	3 hours
21 Sept	Testing magnetism with chess pieces and nails	S	2 hours
15 Okt	Research magnetism and power consumption electromagnet	R	5 hours
16 Okt	Making a list of necessary items from the OpenBuilds Part Store	SR	3 hours
17 Okt	Making a three-dimensional Sketch-up drawing of the design	S	2 hours
23 Okt	Thinking about power source (with stepper) and voice control, estimating costs	SR	6 hours
24 Nov	Receiving components, assembling some of the components	S	1 hours
25 Nov	Sawing Aluminium V-slots	RS	½ hours
25 Nov	Going to lenta and hubo to get some missing components (screws, bolts etc.), writing a formal letter of complaint (see C of the attachments)	S	2½ hours
multiple	Learning programming language C	S	3 hours
1 Dec	Writing the basic program	R	5 hours
4 Dec	Building the XY-table (without the motors for now), making a table of contents for the report and making an appointment with Mr Kouwe	SR	6 hours
6 Dec	Meeting with Mr Kouwe, showing what we have and looking at his motors to see whether they meet our requirements	SR	2 hours

14 Dec	Writing the program in which the robot is controlled by typed moves from the computer	R	6 hours
15 Dec	Finding out how the drivers work, soldering a few wires and testing the drivers/motors with a few programs.	SR	5½ hours
23 Dec	Sawing the base, buying minor parts (screws etc.) from Lenta, putting the components together, making motors move independently, xy-table is now complete	SR	6 hours
24 Dec	Fixing the xy-table to the base, connecting the cables, putting the electromagnet in place, soldering the reset buttons and putting them in place and making a start with sawing the pillars	SR	10 hours
multiple	Cleaning the room after working sessions	S	2 hours
25 Dec	Programming	S	4 hours
26 Dec	Programming	S	4 hours
27 Dec	Sawing the pillars and fixing them to the base, programming horse movement and capturing pieces	RS	8 hours
28 Dec	Writing chapter The Core System	R	2 hours
30 Dec	Writing sub-chapter xy-Table - Wood	R	3 hours
30 Dec	Programming, calibrate system and debug	S	3 hours
21 Jan 2014	Finishing sub-chapter xy-table - Wood, translating piece on magnetism, writing the chapters Introduction, Components and Microcontroller	RS	6 hours
22 Jan	Finishing translation Magnetism, writing the chapters Actual Construction, Screw Thread and Programming	RS	6 hours
5 Feb	Downloading a voice control program, having it communicate with the Arduino and trying to get the microphone to work	RS	5 hours
6 Feb	Translating log, putting sources in the major document and making some minor changes in the report	R	1½ hour
9 Feb	Voice control functioning. Making the Arduino remember the position of the pieces. Starting with possible program	RS	7 hours
10 Feb	installing/soldering lights on board, programming them and starting with voice control 2.0	S	4 hours
11 Feb	voice control 2.0 fully functional	S	5 hours
11 Feb	Numbering figures, sources and formulas. Doing a few layout improvements	R	4 hours

12 Feb	Optimizing the program, making it smaller	S	5 hours
13 Feb	separating black from white captured pieces, preparing for castling, promotion, en passant, Super User and Undo. Labelling wires.	S	3 hours
13 Feb	Rewriting chapters 2-4 so that the text cites to the figures	R	5 hours
15 Feb	Rewriting chapter 5 and 6 and doing some layout improvements	R	5 hours
15 Feb	Replacing broken USB port (soldering the broken one off, and soldering the new one on), Super User, castling	S	6 hours
16 Feb	Splitting chapter 7 in paragraphs and rewriting chapter §7.1 and §7.2	R	6 hours
16 Feb	en passant, general program improvements	S	4 hours
17 Feb	Finishing §7.2 and rewriting §7.3	R	4 hours
17 Feb	taking pictures, writing §3.3, Dutch voice control, Undo function	S	4 hours
18 Feb	Rereading and improving chapter 2. Changing chapter 3 to make it better understandable	R	6 hours
18 Feb	Taking notes of how everything is wired, general program improvements	S	2 hours
19 Feb	Writing chapter 8, Taking pictures, scan receipts, failed attempt to change structure of entire program	S	9 hours
19 Feb	Improving §3.3 and chapter 8. Filling in the rest of the components and writing introduction	R	7 hours
20 Feb	Rereading and improving chapter 4 and 5	R	3 hours
21 Feb	Writing chapter 1	R	1 hour
21 Feb	Writing chapter 9	S	5 hours
22 Feb	Rereading and improving chapter 1,7 and 8	R	7 hours
23 Feb	Rereading and improving chapter 9, putting in the attachments and writing the conclusion	R	6 hours
24 Feb	Making the drawings of the wiring, rewriting conclusion	R	7 hours
24 Feb	Reading and improving the entire paper	S	5 hours
Total	**Sjors: 155 hours**	**Roel: 165 hours**	

Attachments

A) The program:

On the attached CD, there are 3 folders:

• ChessProgram	contains our complete program.
• Servo	contains a library that is referred to in the beginning of our program. In this library, several new functions are defined that are used to control the servo.
• BitVoicer11	contains a similar library with new functions that enables the Arduino to communicate with BitVoicer.

To open the program, go to www.codebender.cc and click Log In (top right corner). Username: sjorspeterse password: schaakrobot1. On the left, there are several documents with a smiley next to them (underneath the heading projects). Click the document called "Chess Robot final".

We made several new functions within the program to make it more organized. These functions refer to code further in the program. This is a list of the current functions with a short description of their use:

autoreset (byte a)
This function is called in zet(). It checks whether the magnet has to move to rank 1 or file H (that is where the reset buttons are). 'a' can be either '1'or '2', depending whether a piece should be moved away from rank 1 or file H (1) or should be moved to rank 1 or file H (2).

capturerun()
Checks if the attacking piece and the victim are more than one field apart. If so, the attacking piece is placed on the rest square. The knight is an exception and will never move in such a way (although we might change that later on).

decoder()
Translates the input received from BitVoicer to a type (pawn, knight, super user, calibrate etc.) and a field (sets x2 and y2).

feedbacklight(byte z)
gives the RGB led light a specific colour. The value of z determines the colour. 0 turns it off, 1 is green, 2 is red, 3 is blue, 4 is purple.

loop()
This is the loop that is continually run if nothing else happens. It keeps track of the time so that the input can be reseted after 5 seconds.

magnet(boolean z)
Moves the servo motor(the magnet) up (z = HIGH), or down (z = LOW). Makes use of the Servo library. (See USB stick)

movem(int x, int y)
Handles the actual movement of the stepper motors. Integer x is the number of half squares that the magnet mover has to move. If x is positive, the magnet mover will move in the direction of rank ':'. If x is negative the moving beam will move in the direction of rank 1. For example, if x = -4, the magnet mover moves 2 squares in the direction of rank 1. y does the same in the y-direction. If y is negative, the moving beam will move in the direction of file '?'. If y is positive, the moving beam will move in the direction of file H. Movem(6,6) will move the magnet diagonally three squares in the direction of square 'H:'. We chose to use half squares, because it is easier that way to move a piece in between two squares, which is needed quite often (knight, captured pieces).

possible()
Checks if a given move is valid. It has a look at the given final position, and then looks if there is a piece of the given type that can move there. See §8.2. If there are no possibilities, the RGB will turn red. If there is one possibility, the move will be executed. If there are multiple possibilities, the RGB will turn blue and the player has to name the square the piece has to come from. This function will probably call for a 'checkCheck()'function in the future, to see if a particular move doesn't leave the king in check.

reset()
This function keeps the motors running until both reset buttons are pressed. This way the robot will know exactly where the magnet is. This is what happens when the 'Calibrate' command is used.

serialEvent()
This function is called whenever data is send from BitVoicer. It is the backbone of the program; see chapter §9.2.

setup()
This function is executed as soon as the Arduino is turned on or reset (with the button, see chapter 5). It tells the Arduino how the components are wired, and calibrates the position of the magnet.

switchturn()
This function changes the variable 'turn' to 0 (white) or 1 (black). It also controls the LEDs on the board that indicate whose turn it is.

trash(byte a, byte b)
Moves the piece that on square (a,b) to a square in the graveyard that isn't yet occupied.

xreset()
Keeps the x-motor running until the x-reset button is pressed. Called by movem()

yreset()
Keeps the y-motor running until the y-reset button is pressed.

zet()
combines the magnet() and movem() functions to physically move the pieces. Also covers some special moves like the knight, and capturing en passant.

A) Calculation electromagnet

The magnetic flux density[39] B is given by:

$$B = \frac{\Phi}{A} \tag{1}$$

Where:

B	flux density (Tesla or T)
Φ	magnetic flux (Weber or Wb)
A	area over which the flux is divided (m^2)

The magnetic flux of an electromagnet[40] is given by:

$$\Phi = \mu N I \tag{2}$$

Where:

Φ	magnetic flux (Wb)
μ	magnetic permeability of the medium (how well the material "conducts" the magnetic field lines) (Vs/A)
N	number of turns in the coil
I	current through the coil (A)

If we combine (1) and (2) we get:

$$B = \frac{\mu * N * I}{A} \tag{3}$$

We want to know the current I which can be determined from equation (3):

$$I = \frac{BA}{\mu N} \tag{4}$$

We want to get rid of the area A in equation (4). We can do this by expressing area A in the distance r in between de electromagnet and the chess piece. Over short distances we can assume that the magnetic flux divides spherically. An approximate calculation is enough as we only need a rough idea of how much current we need. We can therefore conclude that the area A is equal to the area of the sphere with radius r[41]:

$$A_{sphere} = 4\pi r^2 \tag{5}$$

If we fill (5) in into (4) we get:

$$I = \frac{4\pi r^2 B}{\mu N} \tag{6}$$

The value of the magnetic permeability μ of vacuum (μ_0) is equal to $4\pi * 10^{-7}$[42] (by definition). The value of μ of most materials lies really close to μ_0; the value of μ of air is 1.00000037[42] times bigger than μ_0, and the value of wood is 1.00000043[42] times μ_0. These differences are really small, so the calculations will be sufficiently accurate if we just use μ_0. If we fill this in in equation (6) we get:

$$I = \frac{4\pi r^2 B}{4\pi * 10^{-7} N} = \frac{10^7 r^2 B}{N} \tag{7}$$

Now we just need to guess some values. The distance between the electromagnet and the chess piece will probably be about 1 cm, and we assume that the coil has 100 windings. We estimate that the necessary magnetic flux density is 20 mT, based on the fact that a typical refrigerator magnet has a flux density of about 5 mT[43]. When we tried a few different refrigerator magnets, we came to the conclusion that these were more than strong enough to move a separate nail from underneath the board, but not strong enough to move a nail with a chess piece on it. If we fill in these values into equation (7) we get:

$$I = \frac{10^7 (1 * 10^{-2})^2 * 20 * 10^{-3}}{100} = 0.2 \text{ A} \tag{8}$$

However, we weren't sure whether 0.2 ampere is doable or not. So we did research on how long it takes for a 9-volt battery to deplete at this current. A 9 volt battery can supply about 400 mAh[44] (milliampere-hour, this means that the battery can supply 1 mA for 400 hours). If we divide this by the current that runs through the coil (see equation (8)), we can calculate how long it takes before a 9-volt battery depletes:

$$t = \frac{0{,}4}{0{,}2} = 2 \text{ hours} \tag{9}$$

This means that a 9 volt battery can run the electromagnet for about two hours. However, even if a new command is given immediately after the robot has executed the previous one, the magnet would still only be switched on during about half of the time. That means that a game can last four hours without needing to replace the battery. In practice, people usually need some time to think about their next move, so it would usually take way longer than four hours before the battery runs out.

So we went to buy an electromagnet that would conduct the right current at the voltage we were using. Unfortunately it turned out to be too weak after all. We suspect that this was caused by the number of windings in the electromagnet, as this was the only thing no data was given on when we bought it, and all the other conditions we had assumed were met (distance between magnet and piece, current etc.).

B) Letter of complaint

Dear sir, madam,

I'm sorry to say that I was not pleased with my order.

Firstly, when the postman arrived to deliver the package, he demanded a sum of €34,42 (= $46,47), which is nearly as much as the original shipping costs. I have included proof of this payment.

Secondly, the 2x 25 Tee nuts were not included. The description states that they were expected to be released on the 13th of November. My order was placed on the 11th of November, so I assumed that it would take two extra days for the Tee nuts. They were highlighted in yellow (see attachment) on my 'Packing Slip for Order #1770', but that doesn't do me any good. I've payed for the Tee nuts, and my project can't be finished without them.

As it turned out, I have forgotten to add a few parts. These are:

2x Low Profile Screws M5 (25 Pack) (10 mm)
2x Smooth Idler Pulley Kit.

My suggestion is the following:

You send me the 10 mm screws, the smooth idler pulleys and the Tee nuts, free of charge. I will then forget about the rest of the extra money I had to pay (which is $46,47- 2 x $4,50 - 2 x $4,85 = $27,77).
Consider that money my part of your shipping costs.

I am sorry it had to go this way, I am quite enthusiastic about your modular building system, and I hope I can soon finish the final exam project for my high school.

Yours faithfully,
Sjors Peterse

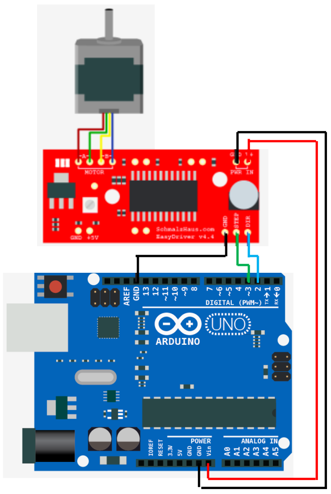
C) The expenses

OpenBuilds Part Store:				
Parts	$195.55	USD	=	€142,28
Shipping	$52.90	USD	=	€38,49
Import rights				€34,42
Total				**€215,16**
Floris:				
Stepper Motor				€16,50
Stepper Motor Driver				€12,50
Microphone				€6,50
Shipping				€5,00
Total				**€53,00**
Other:				
Electromagnet:				€17,62
Screws, bolds etc.				€4,85
Total				**€22,47**

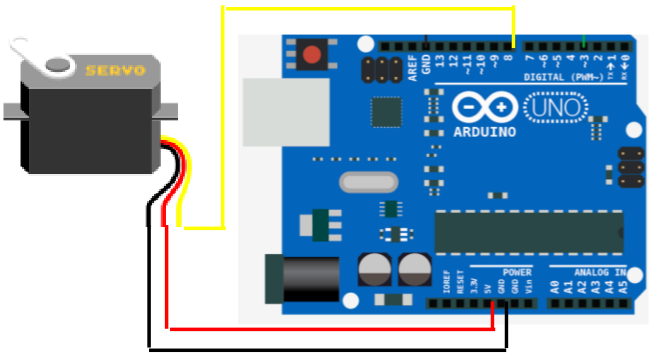
Total: €290,63

D) The wiring

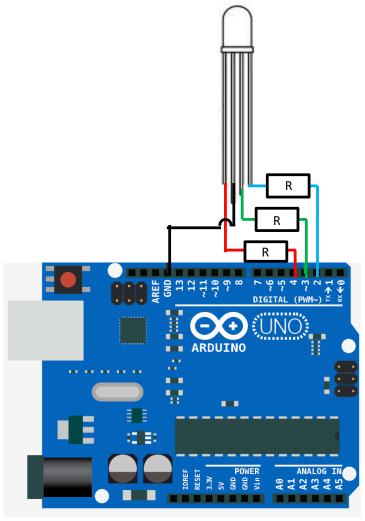
The following drawings are temporary drawings of how to connect the separate components. This means that the wires plugged into the digital pins of Arduino (the pins for digital in- and output, see chapter 5) could end up in a different pin in the complete drawing of the wiring (next page).



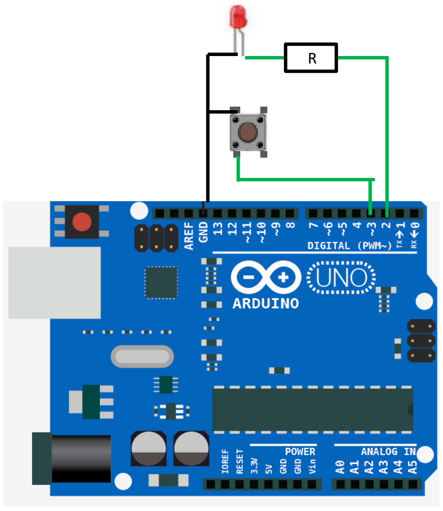
Connecting the Stepper Motor and the Stepper motor driver



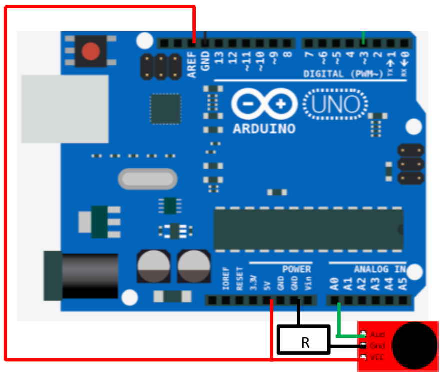
Connecting the Servo Motor that moves the magnet up and down



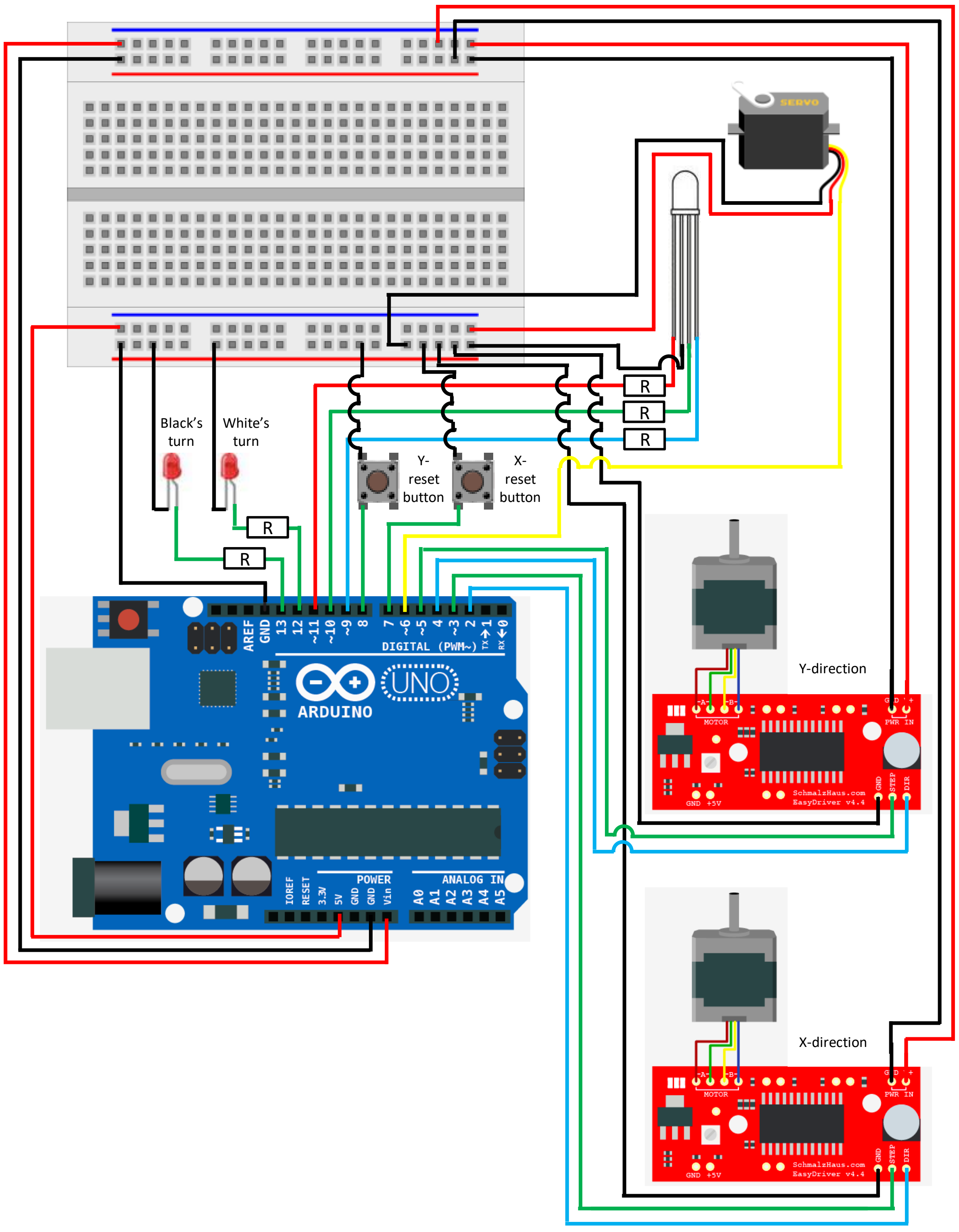
Connecting the RGB LED that communicates with the user



Connecting the buttons and the LED.



This is how we connected the microphone. However, we couldn't get it to work, so the microphone is not shown in the complete drawing of the wiring (next page).



This is a drawing of the complete wiring of our robot. The microphone is not included, as we couldn't get it to work. The grey rectangle on the left of the Arduino is the USB port. A computer is plugged in there to send the commands to Arduino. The black rectangle underneath the grey one is the power jack. A 12 Volt power source needs to be plugged in here to power the Stepper Motors.

Sources

General sources

Wood and screw thread (chapter 3)
http://nl.wikipedia.org/wiki/Eigenschappen_van_hout#Natuurkundige_eigenschappen
http://en.wikipedia.org/wiki/Linear_actuator

Magnetism
http://en.wikipedia.org/wiki/Weber_(unit)
http://www.irm.umn.edu/hg2m/hg2m_a/hg2m_a.html
http://nl.wikipedia.org/wiki/Magnetische_permeabiliteit
http://en.wikipedia.org/wiki/Magnet

Choosing motors
http://www.amci.com/tutorials/tutorials-stepper-vs-servo.asp
http://en.wikipedia.org/wiki/Stepper_motor#Pull-in_torque
http://www.rcpowers.com/community/threads/motor-angle-where-to-position.3562/
http://www.ehow.com/how_5153689_cheap-stepper-motors.html

Electronics & programming instructions
http://arduino.cc/en/uploads/Tutorial/button.png
arduino.cc/en/Reference/ServoWrite
http://bildr.org/2011/06/easydriver/
https://www.sparkfun.com/products/10267
http://www.linengineering.com/line/contents/stepmotors/pdf/Product_Guides/Lin_RG_StepMotorBasics.pdf
http://arduino.cc/en/Reference/FunctionDeclaration

Components
http://arduino.cc/en/Main/arduinoBoardUno
http://openbuildspartstore.com/nema-17-stepper-motor/
http://openbuildspartstore.com/v-slot-300mm-clear-in-stock/
http://www.parallax.com/Store/Robots/RoboticComponents/tabid/198/ProductID/102/List/0/Default.aspx?SortField=ProductName,ProductName
https://www.sparkfun.com/products/9964
http://www.robosoftsystems.co.in/roboshop/index.php/pittman-hybrid-hy200-1713-stepper-motor.html

Voice Control
http://www.makeuseof.com/tag/3-ways-to-add-speech-control-to-arduino-projects/
http://www.bitsophia.com/BitVoicer.aspx
http://www.instructables.com/id/Speech-Recognition-with-Arduino/
http://www.bitsophia.com/Files/BitVoicer_v12_Manual_en.pdf (the manual)

References

[1]http://cdn.ubergizmo.com/photos/2010/6/ChessBot.jpg
[2]http://www.coli.uni-saarland.de/courses/lego-02/pics/chess2-orig.JPG
[3]http://en.wikipedia.org/wiki/File:Dvd_pohon_vozika_hlavy.jpg
[4]http://openbuildspartstore.com/v-slot-20-x-20mm/
[5]http://www.amazonsupply.com/faztek-15qe1515ul-aluminum-t-slotted-extrusion/dp/B008MQA11C
[6]http://openbuildspartstore.com/motor-mount-plate/
[7]http://openbuildspartstore.com/
[8]https://www.thingiverse.com/thing:69333
[9]http://sketchup.google.com/3dwarehouse/details?mid=adcc54c9dae390b73f0509d36d57d51b&prevstart=0
[10]http://sketchup.google.com/3dwarehouse/details?mid=38c79d1d84b30eb6f54e1b6f41fdd78a&prevstart=0
[11]http://sketchup.google.com/3dwarehouse/details?mid=1fb6d0648d7351dfb5c2b6cd1c5299fb&prevstart=0
[12]http://creativec0d3r.blogspot.nl/2012/09/introduction-to-arduino-part-1.html
[13]http://openbuildspartstore.com/v-slot-20-x-40mm/
[14]http://openbuildspartstore.com/gt2-2mm-aluminum-timing-pulley-30/
[15]http://openbuildspartstore.com/gt2-2mm-timing-belt/
[16]http://openbuildspartstore.com/smooth-idler-pulley-kit/
[17]http://openbuildspartstore.com/motor-mount-plate/
[18]http://openbuildspartstore.com/idler-pulley-plate/
[19]http://openbuildspartstore.com/mini-v-plate/
[20]http://openbuildspartstore.com/mini-v-wheel-kit/
[21]http://openbuildspartstore.com/belt-crimp-clamp/
[22]http://openbuildspartstore.com/eccentric-spacers/
[23]http://openbuildspartstore.com/tee-nuts-25-pack/
[24]http://openbuildspartstore.com/low-profile-screws-m5/
[25]http://www.arduino.nu/arduino.jpg
[26]http://www.schmalzhaus.com/EasyDriver/EasyDriver_v42/EasyDriverV42_Guide.png
[27]http://www.robosoftsystems.co.in/roboshop/index.php/pittman-hybrid-hy200-1713-stepper-motor.html
[28]http://floris.cc/shop/en/motor-servo-solenoid-pump/315-stepper-motor-with-cable.html
[29]http://www.sussex-model-centre.co.uk/uploads/Futaba_S3003_Servo.jpg
[30]http://weirdcanada.com/2011/09/new-canadian-istagamble-which-house/breadboard/
[31]http://www.vectors4all.net/vectors/electronic-calliper-button-clip-art
[32]http://openbuildspartstore.com/mini-v-wheel-kit/
[33]http://www.lirtex.com/robotics/servo-motors-information-and-control/
[34]http://floris.cc/shop/en/
[35]http://openbuildspartstore.com/gt2-2mm-aluminum-timing-pulley-30/
[36]http://openbuildspartstore.com/idler-pulley-plate/
[37]http://openbuildspartstore.com/belt-crimp-clamp/

[38]http://www.bitsophia.com/BitVoicer.aspx
[39]http://nl.wikipedia.org/wiki/Tesla_(eenheid)
[40]http://nl.wikipedia.org/wiki/Elektromagneet
[41]http://nl.wikipedia.org/wiki/Bol_(lichaam)
[42]http://en.wikipedia.org/wiki/Permeability_(electromagnetism)
[43]http://en.wikipedia.org/wiki/Tesla_(unit)
[44]http://answers.yahoo.com/question/index?qid=20110115160655AAX6qCt