UNIVERZA V LJUBLJANI
Fakulteta za strojništvo

Cene Trček

AUTOMATING ROOF WINDOW BLIND

Project for Mechatronic Actuators

Ljubljana, June 2025

Contents

List of Figures

List of Tables

1 Introduction

My parents have a big roof window in their bedroom, which we installed the window about 10 years ago. Although the window is meant to provide plenty of natural light, it didn't - because the blind was kept closed most of the time. We found opening the blind difficult due to sheer size and height of the window. While it was possible to use a stick to operate it, the process was impractical and time-consuming.

When brainstorming ideas for project work, my mom suggested automating this blind, which seemed like a great idea! Project was just complex enough to complete within the three weeks I had left.

1.1 Problem definition

To set the stage, let's first take a look at the initial situation of the window, shown in figure 1. The bottom edge sits 2 meters above the floor, while the top reaches just below the ceiling, around 4 meters high. The blind is 1.5 meters wide, which causes skewing of aluminum bar, if it's not moved from the center. Project's requirements were the following:

- The blind should open and close automatically - with movement triggered either by button or by time interval
- It should be controllable without using a phone (after I suggested a web interface)
- It must operate safely
- It should cause minimal damage to the window frame

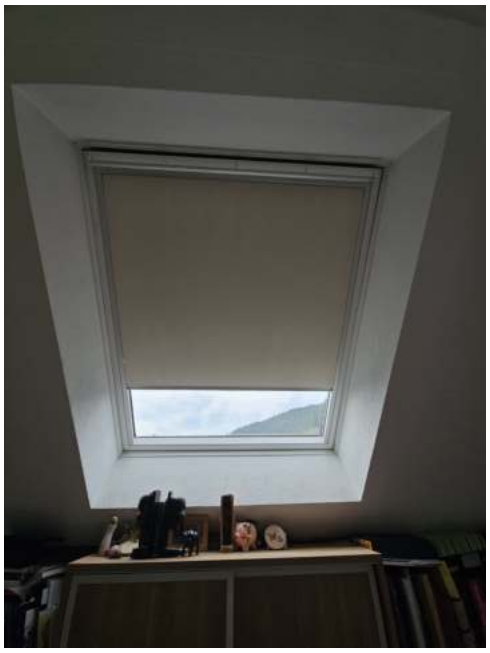
Figure 1: original state of the roof window

A major limitation was that I couldn't order any new components, as the deadline was tight and time was running out. Fortunately, I had salvaged a wide range of components over the years, because, as they say, everything becomes useful eventually.

2 The plan and components used

Now that I knew what to do, it was time to figure out how to do it. The hardest part was determining how to move the blinds up and down using motors while keeping the setup as neat as possible. One advantage was that the window does not need to be opened often, which simplified cable management. After inspection of window and blind structure, I settled on placing two motors on bottom edge of window and connecting it to blind via rope. I could theoretically install a motor behind the blind, but I figured out that this would be challenging to do due to small amount of space available plus, it would significantly damage window in case of project failure. To control the motion of the blind, I considered two options: closed-loop control with distance sensor for precise position monitoring or implementing a finite state machine. I chose the latter, as precise positioning isn't necessary, and it's simpler to implement. To support this approach, I added two limit switches at the ends of the window to let the system *know* when it has reached the top or bottom position. A rough placement plan of the components is shown in Figure 2.

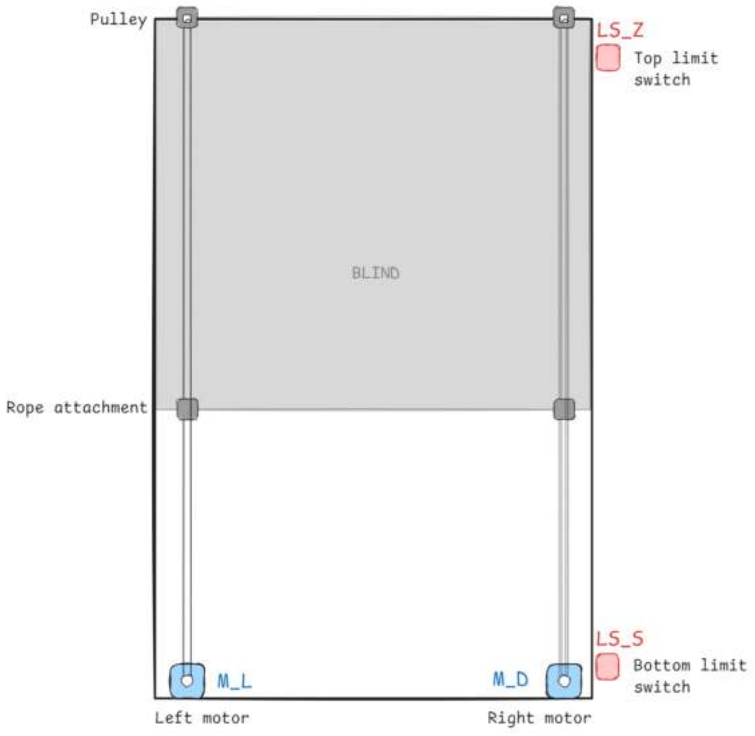

Figure 2: Plan for part placement

2.1 `17HS8401` Stepper motors

I planned to use two DC motors that *I knew I had somewhere*, but I just couldn't find them. Only other motors that seemed to be powerful enough were some steppers that I had left from my laser cutting machine project, which I didn't need anymore as I bought better machine since then. This also meant I could re-purpose laser cutter's Arduino UNO microcontroller and it's CNC shield (more on that in section 2.3).

I couldn't commit to using stepper motors without first confirming whether they could generate enough force to pull the blind down. The blind is quite large, and so I thought that it likely requires significant force to move.

To estimate actual force required to lower the blind, I performed a basic experiment using weights shown in Figure 3. A rope was attached to the blind and passed over a smooth aluminum rod. I then hung weights to the other end until the blind started to move.

The blind started moving at around 2.5 kg of mass. We can simply compute force required to pull blind down using Newton's second law of motion:

$$\begin{aligned} F_b &= m \cdot g = 2.5 \cdot 9.81 \\ F_b &\approx 25\ N \end{aligned} \tag{1}$$

I have two NEMA 17 `17HS8401` [1] stepper motors with following characteristics:

- Holding torque min 52 N/cm
- Detent torque max 2.6 N/cm

I was unable to find an official pull-out torque curve for the motor. However, according to one source, low-speed torque can be approximated as 80% of the holding torque [2] . Thus:

$$T_{\text{ls}} = 52 \cdot 0.8 \approx 40\ Ncm$$

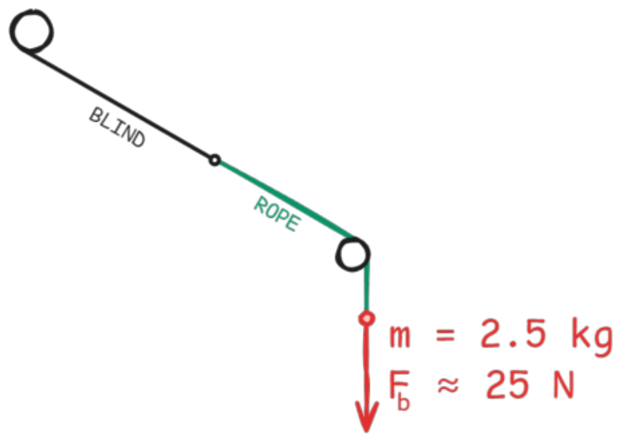

Figure 3: Experiment used to measure the force needed to pull the blind down

The GT2 pulley that is mounted on stepper (visible in figure 10) has major radius of 0.65 cm, so force transferred to rope is:

$$F_m = \frac{T_{ls}}{r} = \frac{40}{0.65} = 60\ N$$

This means that two motors working together will be able to lower the blind, as each would only need to supply $\frac{F_b}{2} \approx 13\ N$ of force. This estimate is really crude but It's good enough for just figuring out if these steppers will do the job.

Nevertheless, I acknowledge that **DC motors** would be a more suitable choice for this application, as they caused some unwanted effects described in section 4.

2.2 Arduino UNO

This is a standard Chinese knockoff of the Arduino UNO microcontroller, which is widely used in the hobbyist community due to its versatility in low-budget projects. It is based on the ATmega328P chip and offers a simple, well-documented platform ideal for prototyping, learning, and integrating basic sensor or actuator control. Its wide ecosystem of libraries and examples makes it especially beginner-friendly.

2.3 Keyestudio CNC Shield V2

I used Arduino in combination with the Keyestudio CNC Shield V2 shown in figure 4. It mounts directly on top of the Arduino and provides a neatly organized interface for driving stepper motors. Its primary purpose is to control small CNC machines in conjunction with the GRBL library, which parses G-code received over serial communication into step and direction signals for the stepper drivers. Shield itself provides:

- pins for mounting three stepper drivers
- pins for connecting three bipolar stepper motors to said drivers
- support for adjusting stepping rate
- pins for connecting limit switches
- pins for sending PWM signal to tool (like spindle or laser module)
- Power supply input
- Miscellaneous I/O and serial pins

All pins that are intended for switches and tool control are just regular digital IO pins that can be reused for any other task. We just need to trace which pin from Arduino is connected to what pin on shield.

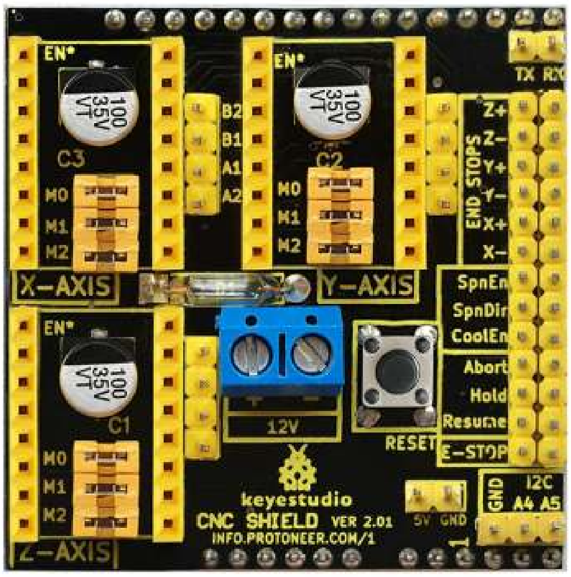

Figure 4: Keyestudio CNC shield V2 [3]

2.4 Limit switches

Initially, I considered using `W-15-102C` (see figure 5) limit switches salvaged from old washing machines. The idea was to mount the switches on the window frame so that the aluminum bar of the blind would press the tiny switch button. In practice, this turned out to be tricky as the switch was too big to fit in the narrow gap between the window frame and the blind profile (see figure 6). I also tried attaching the switch directly to the frame and triggering it with a 3D-printed part glued to the edge of the blind profile. That didn't work either, as the parts kept coming unglued. Best solution would be to screw 3D printed part directly into profile, but that would require disassembling the entire thing.

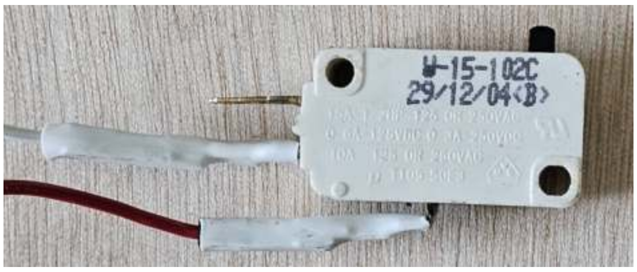

Figure 5: `W-15-102C` switch

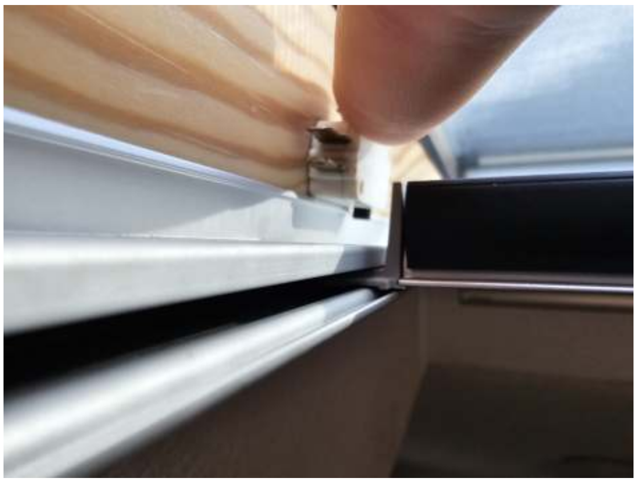

Figure 6: How I envisioned placing the switch

When I was desperately trying to find solution, I got idea to use reed switches that are triggered using magnetic force instead of mechanical contact. These switches have 2 tiny ferrous leads - reeds placed close together in glass tube, as shown on figure 7. When exposed to magnetic field, they snap together and allow electric current to flow trough. Once magnetic field is removed, reeds retract back to their original position and circuit is broken once again. To prevent corrosion and displacement of metal components, reeds are sealed in vacuum inside of a glass tube.

These felt perfect for application, because I could just attach magnet to the side of blind profile instead of gluing some 3D parts on here. Next big challenge was obtaining them since at this point. I remembered we have few dysfunctional bike computers lying around - these use reed switches to detect rotation of the wheel (see figure 8). It can compute rotational speed of wheel from these pulses and finally, by entering front wheel diameter, translational velocity of biker.

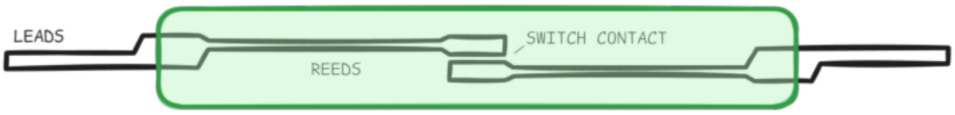

Figure 7: Reed switch

Figure 8: Reed switch used by bike computer to count revolutions of wheel [4]

2.5 Cables

To connect switches with Arduino I used nice and tiny copper wires form an unused Ethernet cable. The problem however is, that these cables come in bundles of 4 twisted pairs, which is too many for my application, I found it really effective to attach one end of cable to vice and other end to a drill to untwist entire bundle into separate cables. Then I prepared 2 cable pairs, all of (mostly) white color and twisted them back using same method (cables can be seen on figures **??** and **??**). Because I didn't want to solder wires onto CNC shield yet, I used some jumper cables to connect ends of long white wires of switches to the CNC shield.

Motors luckily had their own cables already, which are also long enough too!

3 Execution of plan

3.1 Designing finite state machine

Before I began to mount anything onto the window frame, I programmed Arduino with its control logic. As mentioned before in section 2, I decided to use finite state machine approach over closed loop control. Idea was, that blind would automatically open at certain time of the day, for example at 11.00 and stay open till sunset. Because Arduino UNO can't access to real time clock, nor did I have time to purchase RTC module, I intended to connect my Raspberry pi pico W (RPi) and Arduino using serial (UART) communication. Both boards have designated `TX` and `RX` pins that serve this purpose. Let's formalize description of state machine:

- Device idles in `STOP` state
- If Arduino receives `UP` command over serial, it moves to `UP` state
- similarly, if it receives `DOWN` command, it goes to `DOWN` state
- In `UP` state, system moves blind up until it triggers top limit switch
- in `DOWN` state, it lowers blind until it triggers bottom limit switch.
- At any time device may receive `STOP` command, at which it must immediately halt movement and return to `STOP` state

On figure 9 shows state diagram for this machine. Variable names in the image, which are also used in the code later, are a mix of Slovenian and English abbreviations, for example `LS_Z` means *Limit switch zgornji* and so on.

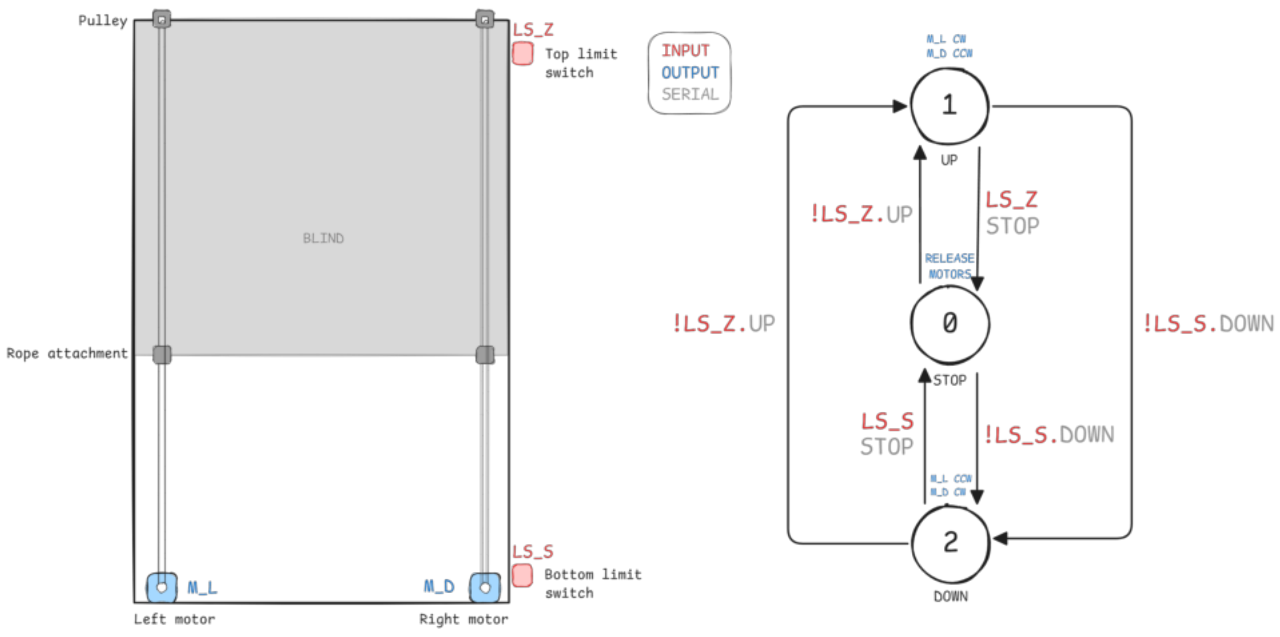

Figure 9: First version of state machine

3.1.1 Development environment

I *could* totally just use Arduino IDE to write code in, which works great. Especially version 2 of this program, which is based on visual studio code editor, that was developed and open sourced by Microsoft. However, I was interested in finding alternative to that. I found quite popular extension for just mentioned Visual Studio code, called *PlatformIO*.

It is an open-source ecosystem for IoT development that supports multiple platforms, frameworks, and boards. It provides a unified, cross-platform build system and library manager. It also allows user to test and compile code and write it to board's memory. Main advantage I saw over Arduino IDE is, that it runs in visual studio code, which means that I don't need additional software plus, all my settings, themes and AI copilot work there [5]. One drawback however is a slightly more advanced setup on the first run, but that was not too much of an issue. Actual Arduino program is written in c++ language using PlatformIO's Arduino framework. Full source code can be found on my GitHub repository [6], but in next chapter you can read `main.cpp` file.

3.1.2 State machine code V1

```
#include <Arduino.h>
#include <AccelStepper.h>
#include <math.h>

#define DRIVER 1 // Use DRIVER = 1 for step/dir drivers
#define LS_Z 11  // Z_lim
#define LS_S 10  // Y_lim
#define ENABLE 8

AccelStepper M_L(AccelStepper::DRIVER, 2, 5);  // STEP = 5, DIR = 2
AccelStepper M_D(AccelStepper::DRIVER, 3, 6);  // STEP = 6, DIR = 3

int maxSpeed = 1000; // Steps per second
unsigned long millisPrev = 0;
unsigned long t = 0;
int speed = 0;

typedef enum
{
  STOP,
  UP,
  DOWN
```

```
} State;

typedef struct
{
  State now = State::STOP;
  State prev = State::STOP;
} StateMachine;
StateMachine Window;

int speedRamp(double milis, double transitionTime, double finalSpeed){
  // Speed of motor increases slownly by following an S curve
  return (int)round(finalSpeed * (1 - exp(-pow((milis / (transitionTime / 2)), 2.0))));
}

void setup()
{
  Serial.begin(9600);
  pinMode(LS_Z, INPUT_PULLUP); // LOW when swithch is activated
  pinMode(LS_S, INPUT_PULLUP);
  pinMode(ENABLE, OUTPUT); // HIGH -> steppers can move freely

  M_L.setMaxSpeed(maxSpeed); // Steps per second
  M_D.setMaxSpeed(maxSpeed);
}

void loop()
{
  if (Window.prev != Window.now)
    Serial.println(Window.now);
  Window.prev = Window.now;

  // SERIAL CONTROL
  if (Serial.available() != 0)
  {
    String tmpstate = Serial.readString();
    tmpstate.toUpperCase();
    tmpstate.trim();

    if (tmpstate == "UP")
    {
      Window.now = State::UP;
      millisPrev = millis();
    }
    else if (tmpstate == "DOWN")
    {
      Window.now = State::DOWN;
      millisPrev = millis();
    }
    else if (tmpstate == "STOP")
      Window.now = State::STOP;
  }

  // exit movement states
  if ((Window.now == State::UP && digitalRead(LS_Z) == 0) || (Window.now == State::DOWN && digitalRead(LS_S) ==
  ↪  0))
  {
    Window.now = State::STOP;
    break;
  }

  // IO logic
  switch (Window.now)
  {
  case State::STOP:
```

```
  M_L.setSpeed(0);
  M_D.setSpeed(0);

  digitalWrite(ENABLE, HIGH); // RELEASE motors
  break;

  // This is effectively an OR statement
  case State::UP:
  case State::DOWN:
      t = millis() - millisPrev;
      speed = speedRamp((double)t, 1000.0, maxSpeed) + 100; // Slowly increase speed using S-curve

      digitalWrite(ENABLE, LOW); // ENABLE motors

      M_L.setSpeed(Window.now == State::UP ? speed : -speed);
      M_D.setSpeed(Window.now == State::UP ? -speed : speed);

      M_L.runSpeed(); // Run steppers with current speed
      M_D.runSpeed();
      break;
  }
}
```

There are few not-so-clean things in code, for example the way I'm handling states, but hey, it works. You might have also noticed that I'm using `AccelStepper.h` that allows using steppers with stepper driver, which is not supported in *classic* `Stepper.h` library. For first tests I was connected to Arduino with USB cable and was sending serial commands this way. I did not write RPi controller script yet, although I made a simple program to test whatever my PlatformIO based toolchain works and to test its internet connectivity.

3.2 3D printed parts

First, I modeled motor holder in PTC Creo. The process was relatively straightforward, as dimensions for NEMA 17 motors are freely available online. I then 3D printed it on my Prusa MKs with black PLA filament, to match the rest of motor assembly. Motor holder is shown on figure 10. Next was piece that attaches rope to blind (figure 11). I made it in a way it just snaps into slot of blind aluminum profile, which makes it really easy to remove, while still making connection strong enough to actually pull blind reliably. I printed this part with white PETG. Last 3D printed part was pulley that holds rope at top of the window and lets it run smoothly (figure 12). This piece was a bit trickier, because I wanted it to snap onto handle of the window. To do that without breaking, I had to print it using PETG, because it is more flexible compared to PLA filament. I also made part of the print that *hughs* handle with 100% infill. I was pleasantly surprised by the strength of the hook! On top part of holder I mounted a metal pulley with ball bearing and secured it with screw and nut. This way, the friction generated is minimal.

3.3 Mounting actuators and first tests

I mounted motor holder with 2 screws to bottom frame of window, which is all of permanent damage done to window. All other parts can still be moved around / replaced without any visible marks. First test was a mild failure - the blind did **move**, but not much because stepper motors were skipping steps. First thing I noticed was that rope was rubbing against a part of pulley 3D print. I solved that by reorienting print into a different position, which eliminated rubbing. But steppers still had hard time moving blind.

The root of the issue was microstepping, which allows the stepper motor shaft to rotate in smaller increments than a full step, but at the cost of reduced torque. This occurs because power is distributed to the motor coils in a sinusoidal pattern, unlike in full-step mode where it is shared more evenly between two coils. The step size is configured by placing jumpers on the `M1`, `M2`, and `M3` pins on the CNC shield, as shown in Figure 4. Different jumper combinations result in different step resolutions, as shown in Table 1.

Since all three pins were enabled, motor was operating in `1/16` of a step mode, which delivers smallest torque. To maximize torque, I unplugged all 3 jumper caps, which set driver to operate in full step mode. After that, motors were finally able to move blind successfully! This was certainly a big milestone, however I still had to manually stop blind at its limit positions.

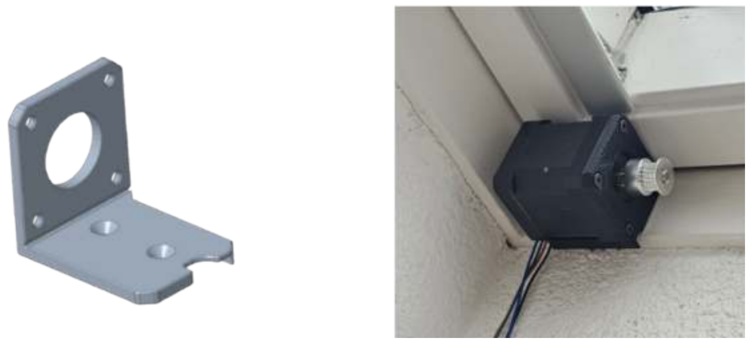

Figure 10: Motor holder

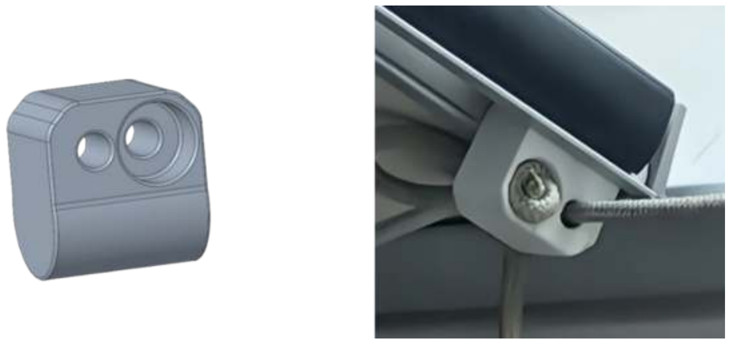

Figure 11: Rope attachment

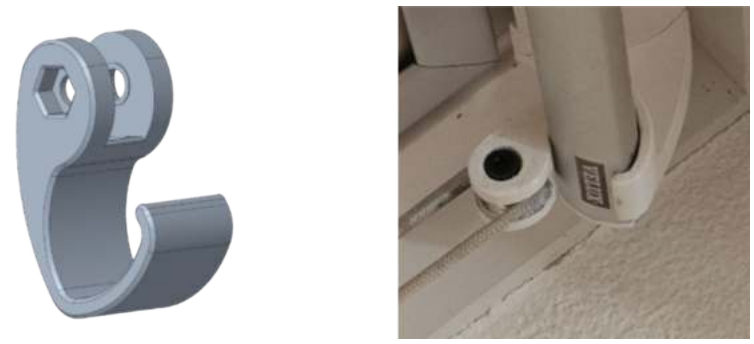

Figure 12: Pulley

Table 1: Microstepping Resolution Truth Table [7]

MS1	MS2	MS3	Microstep Resolution	Excitation Mode
0	0	0	Full Step	2 Phase
1	0	0	Half Step	1-2 Phase
0	1	0	Quarter Step	W1-2 Phase
1	1	0	Eighth Step	2W1-2 Phase
1	1	1	Sixteenth Step	4W1-2 Phase

3.4 Installing limit switches

I have soldered reed switches to long cables and protected them using heat shrinking tube - that way damaging it is less likely and it also insulates otherwise exposed electrical contacts. After that, installation of limit switches wasn't too troublesome as I just hot-glued them to rail. I placed limit switches on such spots, that motors could reliably pull the blind without skipping steps.

With that done, when sending `UP` command, blind moved all the way up and then stopped right there, just as intended!

3.5 Actually controlling blind movement without PC

At this point I still had three goals to complete:

1. Make the blind controllable without phone

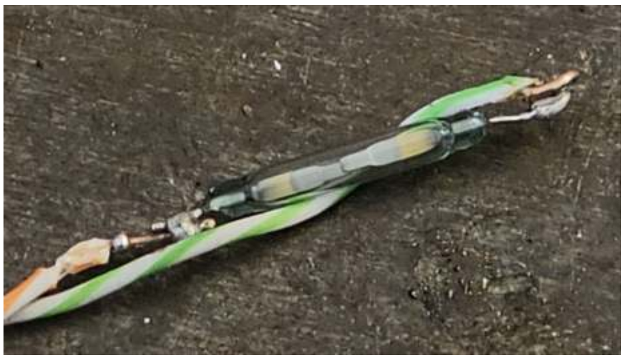
Figure 13: Reed switch before it was inserted into protective tube

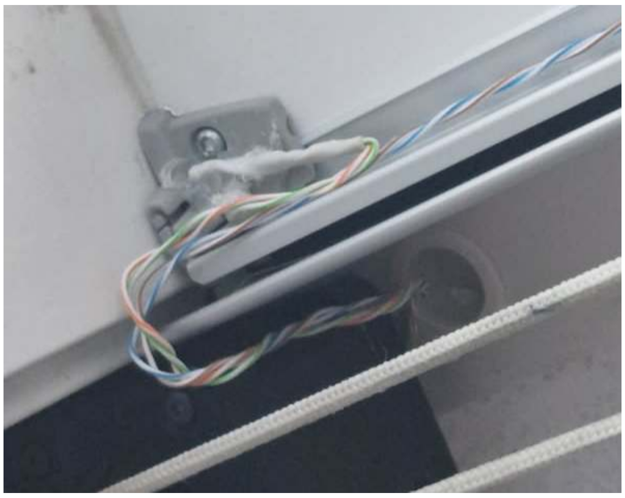
Figure 14: Reed sensor mounted on window frame in lower position, with other cable leading to upper sensor

2. Pack everything into nice enclosure
3. Make the blind move up and down automatically based on time of the day

Time left to submit report was getting shorter and shorter, so I decided to focus on completing goal number 1 and 2 first. I left goal 3 for later, because it would take quite some time to solder all the pins of RPi and to write script.

I have decided that control signal could be generated either by detecting claps using sound sensor or by pressing a simple button. Clapping to open blinds sounded fancier so I began working on that, but unfortunately sound detection unit I had was defective, so the button was the only viable option left.

For that reason I had to modify my state machine from version 1 (figure 9) to include new logic for controlling movement. Updated state diagram is shown on figure 15.

- **State 0**: (initial) System waits for button press, after which it transitions state 1. If top limit switch is triggered, transition to state 2.
- **State 1**: Blind moves up until button press or it triggers top limit switch after which it transitions to state 2
- **State 2**: System waits for button press, after which it transitions state 3. If bottom limit switch is triggered, transition to state 0.
- **State 3**: Blind moves down until button press or it triggers bottom limit switch after which it transitions to state 0
- Serial commands should be able to transition system from any state into next logical state

Implementation of new logic wasn't too difficult, but as I found out, It was also crucial to only change state once after button gets pressed and ignore it for remaining time until it it released - this is what variable `SS` is used for. It generates single pulse once button is pushed and then resets to default value. For that purpose, I used `W-15-102C` that I originally intended to use as a limit switch (it is also shown in figure 16).

3.5.1 State machine code V2

```
#include <Arduino.h>
#include <AccelStepper.h>
#include <math.h>

// Stepeprs
#define ENABLE 8
int maxSpeed = 70;  // S/s
int speedRampTime = 3000;
AccelStepper M_L(AccelStepper::DRIVER, 2, 5);  // STEP = 5, DIR = 2
```

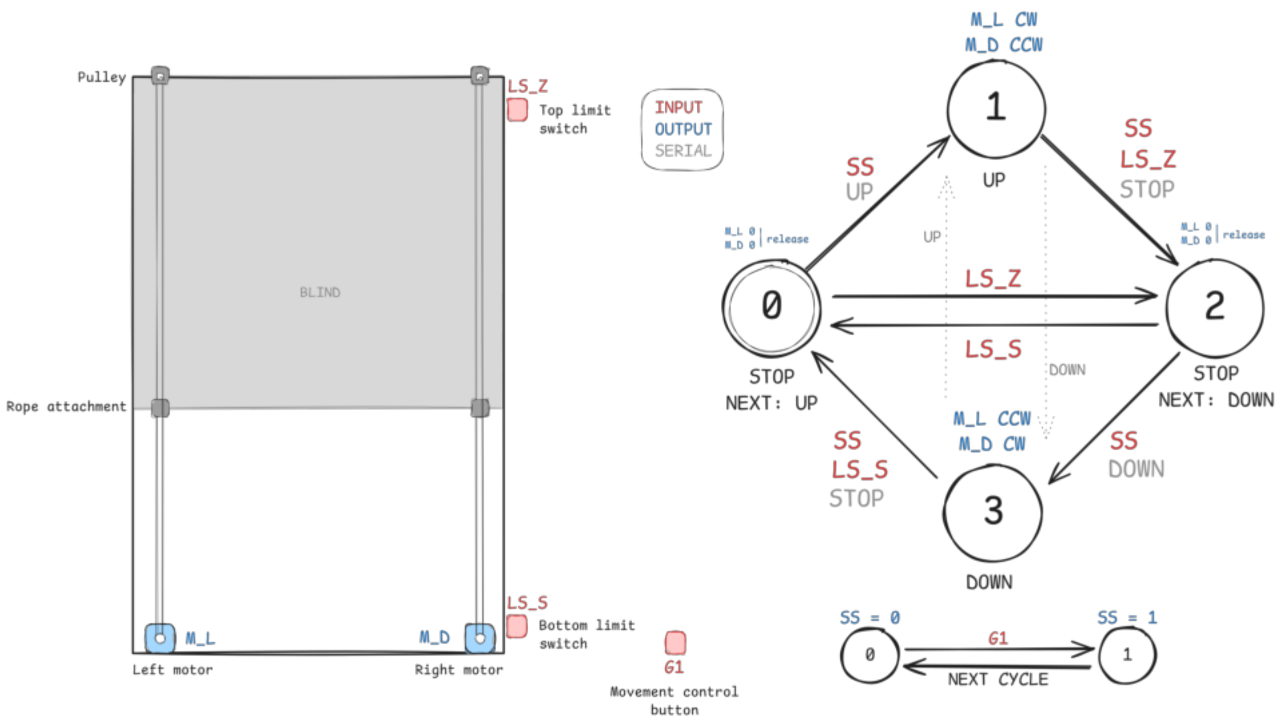

Figure 15: Second version of state machine lets user control blind using a single button

```
AccelStepper M_D(AccelStepper::DRIVER, 3, 6);  // STEP = 6, DIR = 3

// Gumbi
#define G1 9      // X_lim
#define LS_S 10   // Y_lim
#define LS_Z 11   // Z_lim

// Program variables
String serialCommand = "";
unsigned long millisPrev = 0;
unsigned long t = 0;
int speed = 0;
bool SwitchState = false;
bool preveriousG1state = true;

typedef enum {
    STOP_UP,
    UP,
    STOP_DOWN,
    DOWN,
    NONE
} State;

typedef struct
{
    State now = State::STOP_UP;
    State prev = State::STOP_UP;
    State next = State::NONE;
} StateMachine;

StateMachine Window;

int speedRamp(double milis, double transitionTime, double finalSpeed) {
  return (int)round(finalSpeed * (1 - exp(-pow((milis / (transitionTime / 2)), 2.0))));
}

```

```
void setup() {
  Serial.begin(9600);

  pinMode(LS_Z, INPUT_PULLUP);  // LOW when swithch is activated
  pinMode(LS_S, INPUT_PULLUP);
  pinMode(G1, INPUT_PULLUP);
  pinMode(ENABLE, OUTPUT);

  M_L.setMaxSpeed(maxSpeed);  // Steps per second
  M_D.setMaxSpeed(maxSpeed);
}

void loop() {
  // Debug
  if (Window.prev != Window.now) {
    Serial.print(">State:"); //works great with Teleplot VS code extension
    Serial.println(Window.now);
    millisPrev = millis();
}
Window.prev = Window.now;

bool G1State = digitalRead(G1);
SwitchState = (preveriousG1state && !G1State); //detect falling edge
preveriousG1state = G1State;

  // GET SERIAL COMMANDS
  if (Serial.available() != 0) {
    serialCommand = Serial.readString();
    serialCommand.toUpperCase();
    serialCommand.trim();
  } else
    serialCommand = "";

  // STATE TRANSITION LOGIC
  Window.next = Window.now;

  switch (Window.now) {
    case State::STOP_UP:
      if (!digitalRead(LS_Z))
        Window.next = State::STOP_DOWN;
      else if (SwitchState || serialCommand == "UP")
        Window.next = State::UP;
      break;

    case State::STOP_DOWN:
      if (!digitalRead(LS_S))
        Window.next = State::STOP_UP;
      else if (SwitchState || serialCommand == "DOWN")
        Window.next = State::DOWN;
      break;


    case State::UP:
      if (SwitchState || !digitalRead(LS_Z) || serialCommand == "STOP")
        Window.next = State::STOP_DOWN;
      else if (serialCommand == "DOWN")
        Window.next = State::DOWN;
      break;

    case State::DOWN:
      if (SwitchState || !digitalRead(LS_S) || serialCommand == "STOP")
        Window.next = State::STOP_UP;
      else if (serialCommand == "UP")
        Window.next = State::UP;
    break;
}

Window.now = Window.next; // This has to be done this way so switch() variable doesnt change

  // IO
  switch (Window.now) {
    case State::STOP_UP:
    case State::STOP_DOWN:
      M_L.setSpeed(0);
      M_D.setSpeed(0);

      digitalWrite(ENABLE, HIGH);  // Release stepper control
      break;

    case State::UP:
    case State::DOWN:
      t = millis() - millisPrev;
      speed = speedRamp((double)t, speedRampTime, maxSpeed) + 100;  //Slowly increase speed using S-curve

      digitalWrite(ENABLE, LOW);

      M_L.setSpeed(Window.now == State::UP ? speed : -speed);
      M_D.setSpeed(Window.now == State::UP ? -speed : speed);

      M_L.runSpeed();
      M_D.runSpeed();
      break;
  }
  delay(1);
}
```

I had also cleaned up the code and made state transitioning logic clearer and more organized. I also added short delay at end of loop because it solved weird issue of system sometimes transitioning 2 states at once and also, we don't really need to resample state of system that often. After that, it really worked like it was supposed to!

3.6 Finishing up

Final thing to do was to wrap up the project and make a nice enclosure for all the electronics to fit in. First, I did not need a USB cable to debug Arduino anymore. But once unplugged, I realized that CNC shield does not also supply power to Arduino board with its 12 V power supply. To solve that I soldered 2 wires to bottom of Arduino's 12 V power jack and plugged those together with 12 V power supply for CNC shield (see figure 16). Now a single power cord can power both devices simultaneously.

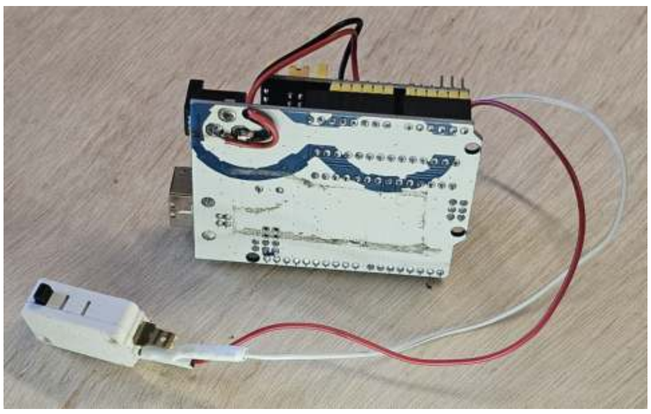

Figure 16: Red and black wires are connected form CNC shield's power supply to Arduino's 12 V power jack

Next, I laser cut a wooden box with hinged lid. To avoid spending too much time on designing the box, I found a great box generator called `boxes.py`, that generates `.svg` files for us using predefined templates and given dimensions [8]. Aside from raw python library, they also have online generator and Inkscape plugin. I have modified generated files to have more air vents, and a hole on the back to pass the cables trough. After making the box, I realized this could be a bit of fire hazard. To mitigate risk, I poured fine concrete into bottom of the box to help disperse the heat in case Arduino begins to overheat. I also tried to ignite wood, from which the box is made, using soldering iron. It did not even start to smoke, even after leaving it on direct heat exposure for quite some time, so I think I'm safe. Finished enclosure is shown on figure 17. I planned for the hinged lid of the box to serve as button - you press on the lid and blind starts to move. Realization was really simple - after embedding all the components into the box and connecting pins, I glued `G1` switch to front end of the box (figure 18) so it's button keeps the lid open for just a few millimeters. This approach has proven to work really well!

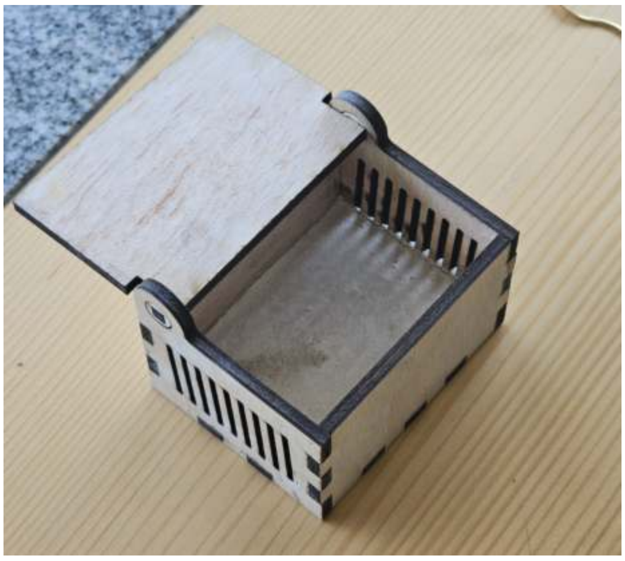
Figure 17: Finished enclosure

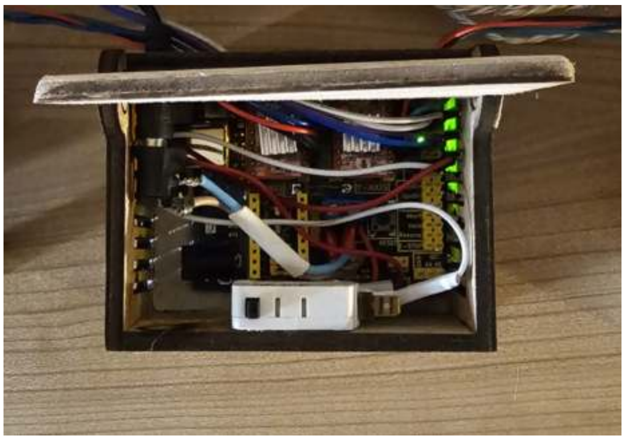
Figure 18: Finished enclosure with Arduino and `G1` switch installed

To hide cables, I just put back all trinkets and images that were on the shelf before. On figure 19 you can see, how everything turned out in the end. And with that, the project was finished! Wiring diagram of final state of project is shown on figure ??.

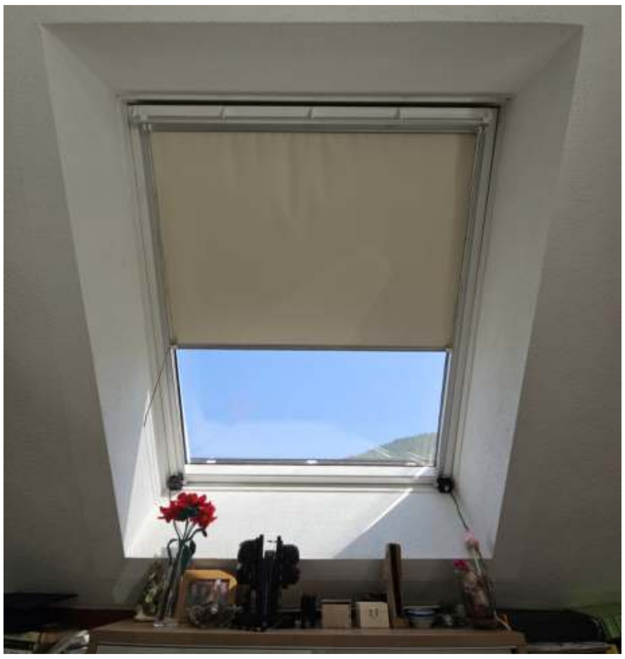
Figure 19: Finished project

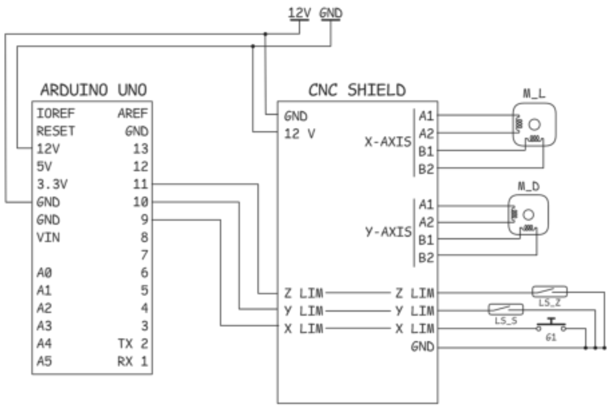

Figure 20: Finished project wiring diagram

4 Analysis

I think that the project turned out great as a proof of concept. It successfully moves the blind up and down, with the added ability to partially open it. However, a few drawbacks arose from the choice of stepper motors, with the reasoning behind this choice detailed in section 2.1. The most significant issue is the noise they produce. While stepper vibrations are typically tolerable in applications like CNC machines, the glass in the window amplifies them considerably. This is closely related to another issue - I had to run the motor at a very low speed to generate enough torque to move the blind. As a result, a full move of blind takes about one minute of constant noise.

4.1 Future plans

When I will have time to do so, I'm going to improve the project by:

- Replacing stepper motors with DC motors, which will result in faster and smoother motion with less noise
- Improving position of limit switches, so that the blind will close all the way
- I will replace Arduino with raspberry pi pico w, which will enable time related functionality

4.2 Final thoughts

The project was quite fun and useful. My parents are also satisfied with how it turned out, although they are a bit concerned by the noise produced by motors. In past few days of testing, we had seen a lot more sunlight reach into that room, which means that the project really succeeded in what it was intended to do.

References

[1] MotionKing, *17hs2408 stepper motor datasheet*, Accessed: 2025-06-18. [Online]. Available: `https://datasheet4u.com/pdf-down/1/7/H/17HS2408-MotionKing.pdf`.

[2] Kollmorgen Support Network, *Pull-in and pull-out torque*, Accessed: 2025-06-18. [Online]. Available: `https://www.kollmorgen.com/en-us/developer-network/pull-and-pull-out-torque`.

[3] Keyestudio. "Ks0151 keyestudio cnc shield v2." Accessed: 2025-06-19. (n.d.), [Online]. Available: `https://wiki.keyestudio.com/Ks0151_keyestudio_CNC_Shield_V2`.

[4] Road Bike Basics. "Landscape image." Accessed: 2025-06-19. (2020), [Online]. Available: `https://roadbikebasics.com/wp-content/uploads/2020/12/Landscape_small_compressed.jpg`.

[5] PlatformIO. "Platformio: The most loved ide solution for microsoft visual studio code." Accessed: 2025-06-19. (), [Online]. Available: `https://platformio.org`.

[6] Cene Trček. "Project's github repository." Accessed: 2025-06-19. (), [Online]. Available: `https://github.com/CTJoriginal/Mechatronic_Actuators`.

[7] ALLEGRO. "A4988 driver datasheet." Accessed: 2025-06-19. (), [Online]. Available: `https://www.allegromicro.com/-/media/files/datasheets/a4988-datasheet.pdf`.

[8] Bamberg Hackerspace. "Universal box – hackerspace bamberg." Accessed: 2025-06-19. (), [Online]. Available: `https://boxes.hackerspace-bamberg.de/UniversalBox?language=en`.