Moritz Larndorfer moritz.larndorfer@gmail.com

AMRAS Build Manual

Figure 1 The finished build

Table of Contents

Introduction

AMRAS is the abbreviation for the German long form “Anti-Mitbewohner-Raketen-Abwehr-System” which translates to Anti-Roommate-Rocket-Defence-System.

It uses a combination of a Raspberry Pi, a camera, some servos, trafos and rubbing alcohol to propel rockets into anyone’s face who dares to enter its vicinity

This guide will show you how to build and start it, have fun!

I got inspired when reddit flushed this video onto my frontpage and I thought “that’s kinda cool, but I have a 3D printer and a Raspberry Pi, I could do that better”.

My code is based on Adrian Rosebrock’s tutorial who used a similar setup to track faces. While he used a PID controller for moving the servos, I had to fall back to a simpler movement, but this will hopefully soon be fixed.

Also thank you to Florian Geißegger, my roommate who ironically helped me with the soldering and had some general input.

Parts List

These are not affiliate links, just the parts I used. It should also work with slightly different hardware, but then you would need to adapt the 3D models for custom mounting holes.

- A Raspberry Pi 3B (a newer version should also work, but this is what I have tested with
- Power Supply
- Camera
- Servo Controller
- 4x Transformer/Spark generator
- 2x Servo
- 4-Band Relais
- Some Cables
- 3x Ball Bearing
- 4x Mini Magnets for keeping the lid shut
- Syringes to craft rockets out of
- 3x Switches
- Micro USB Cable to provide power for the Raspberry Pi (or USB-C if using a newer model)
- SD Card with minimum 8GB for the operating system
- Superglue
- Bunch of Screws (I used Wood Screws)
  Raspi: 4x 2,0x10mm
  Kamera: 4x 2,0x10mm
  Main Body Fillings: 24x 2,9x13mm
  Arms: 4x 2,9x13mm
  Servo Shield: 4x 2,0x10mm
  Relais: 1x 2,0x10mm
- All printed 3D parts

Tools

- Screwdrivers
- Soldering Iron and Tin
- Tweezers
- Tongs
- Wire Stripper

Assembly

Software 1

Installing Raspberry Pi OS

Install Raspberry Pi OS on the SD card and boot your Raspberry Pi.

Installing the necessary python packages

Open the terminal on your raspberry pi and enter following commands:

sudo pip3 install opencv-python

sudo pip3 install smbus

sudo pip3 install imutils

sudo pip3 install "camera[array]"

sudo pip3 install adafruit_servokit

sudo pip3 install adafruit_motor

sudo pip3 install numpy

sudo pip3 install matplotlib

Configuring the Raspberry Pi

Enter '*sudo raspi-config*' to open the Raspberry Pi Software Configuration Tool, go to 'Advanced' and enable Camera and I2C (and SSH, if you want to control AMRAS remotely).

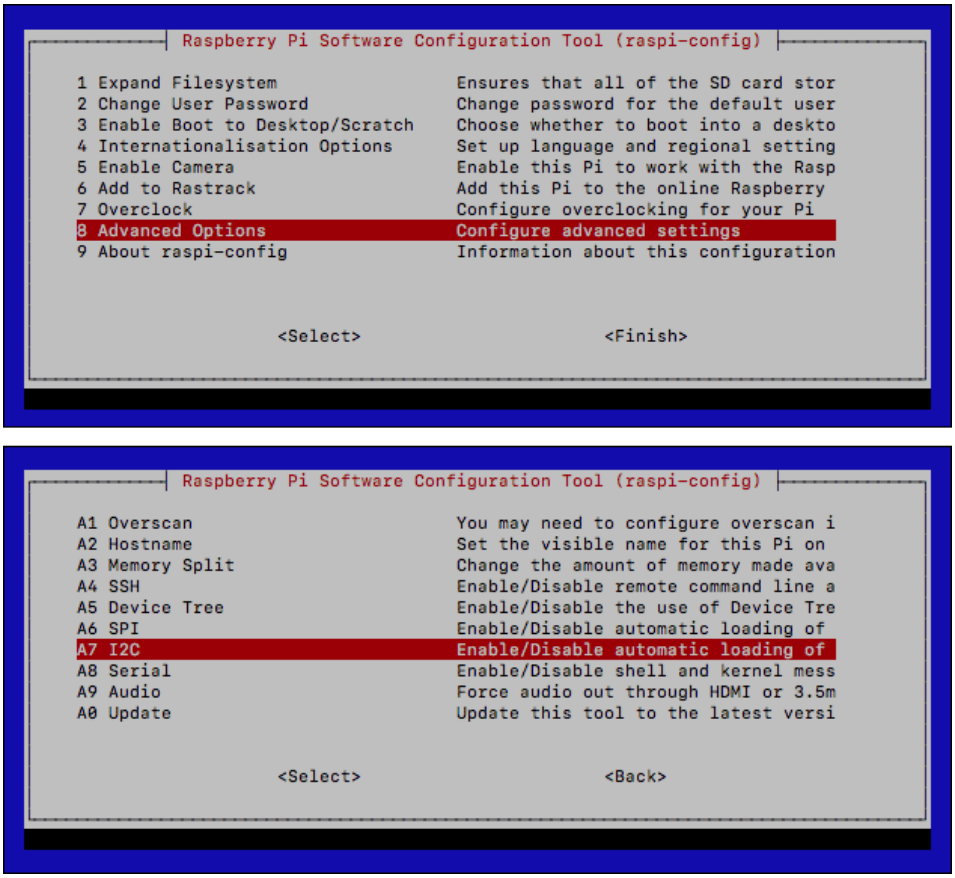

Figure 2 Raspberry Pi Software Configuration Tool

Hardware

Dry Running

I recommend connecting all parts according to the circuit diagram first, just so you don't end up having any missing/broken parts which will need to be swapped (this can be really annoying, trust me).

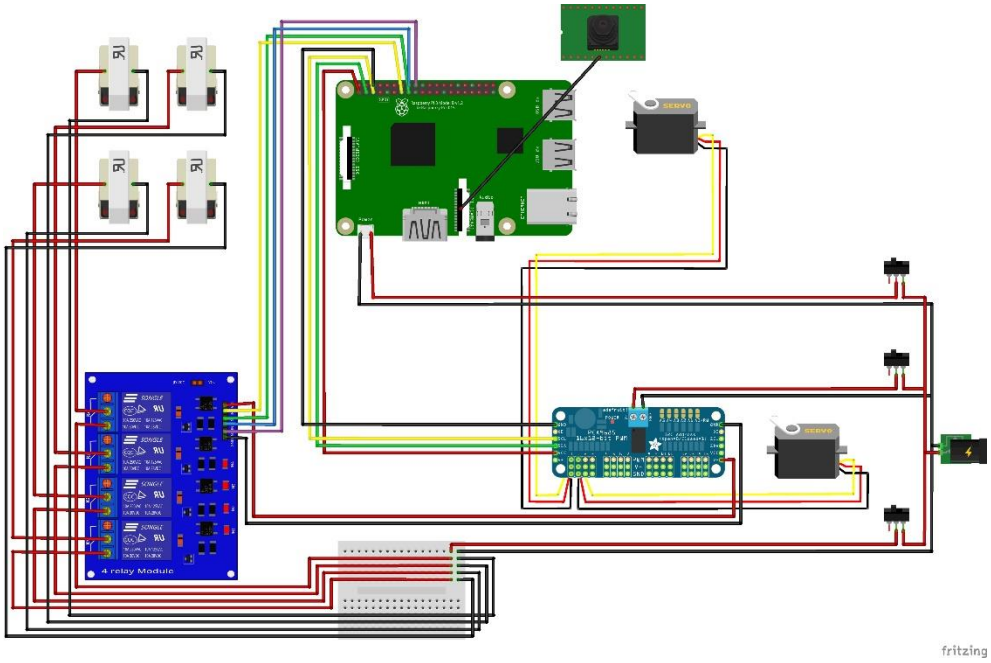

Figure 3 Circuit Diagram

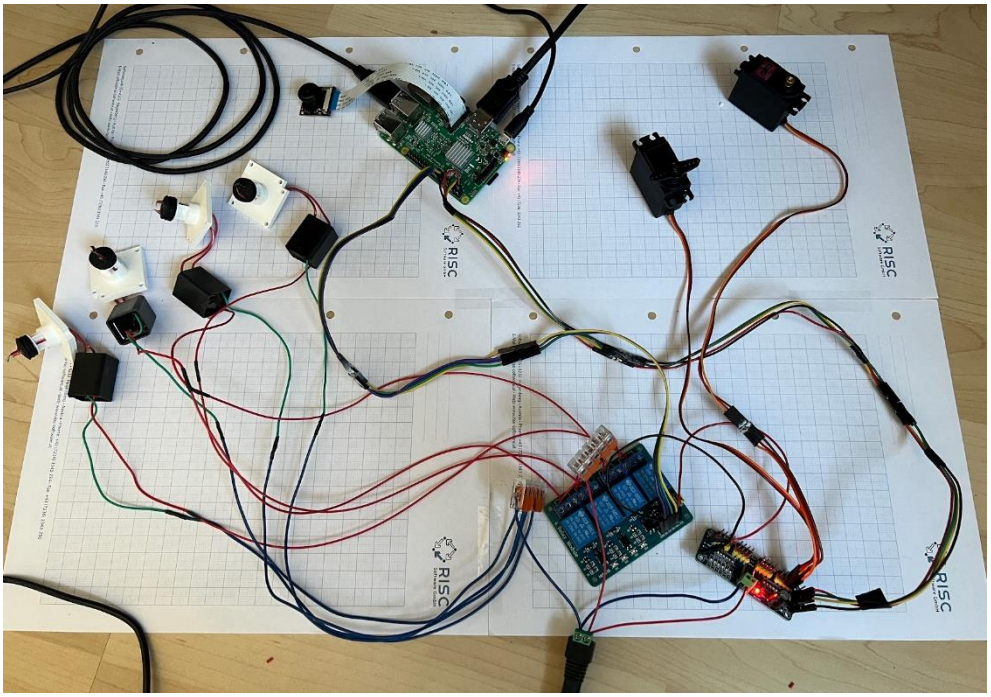

Figure 4 This is just ~90% correct, I forgot to take a new picture after making some adjustments

After making sure everything is connected properly, run test_servos.py to check functionality.

Try not to move the servos afterwards, as this would move the home/default position of AMRAS.

Crafting the Projectiles

Use the rocket fin template provided with the other files to cut fins and the tip out of rubber or similar. Cut off the tip of the syringes and glue the hole shut. Glue on the parts to create the projectiles.

Figure 5 The finished projectiles

To shoot them, you will need to put in a drop of rubbing alcohol and put them into their “missile bays”. The evaporated alcohol will propel them.

Alternatively, you can also just use the provided 3D missile files and print them.

Gluing Magnets

Start by gluing the magnets for the lid in place, making sure to have the proper polarity.

Figure 6 Lid and Main Body with the Magnets

Installing the Camera

Screw in the Camera, the ribbon cable should fit into the recess below it.

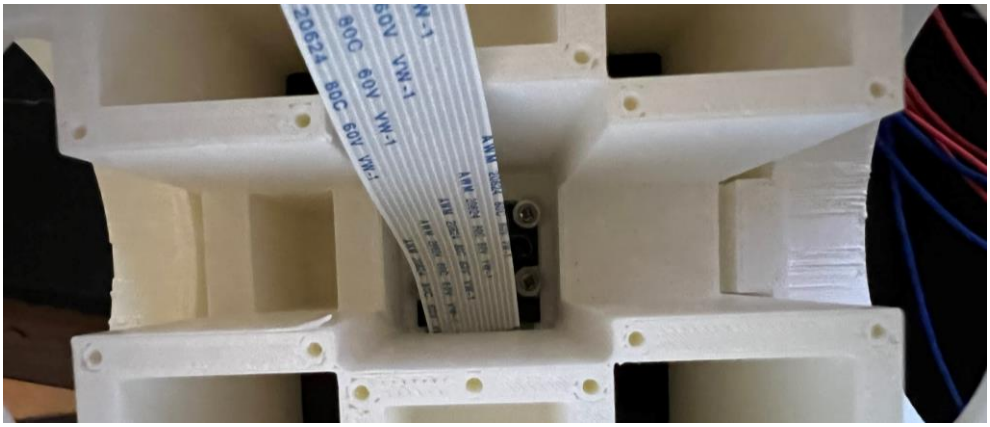

Figure 7 The Camera screwed in place

Preparing and installing the transformers

Poke two holes through the end pieces of the syringes and passthrough the output cables of the transformers through both the mount and the syringe end piece. The endpiece needs to be cut off to be slotted into the cross-holes.

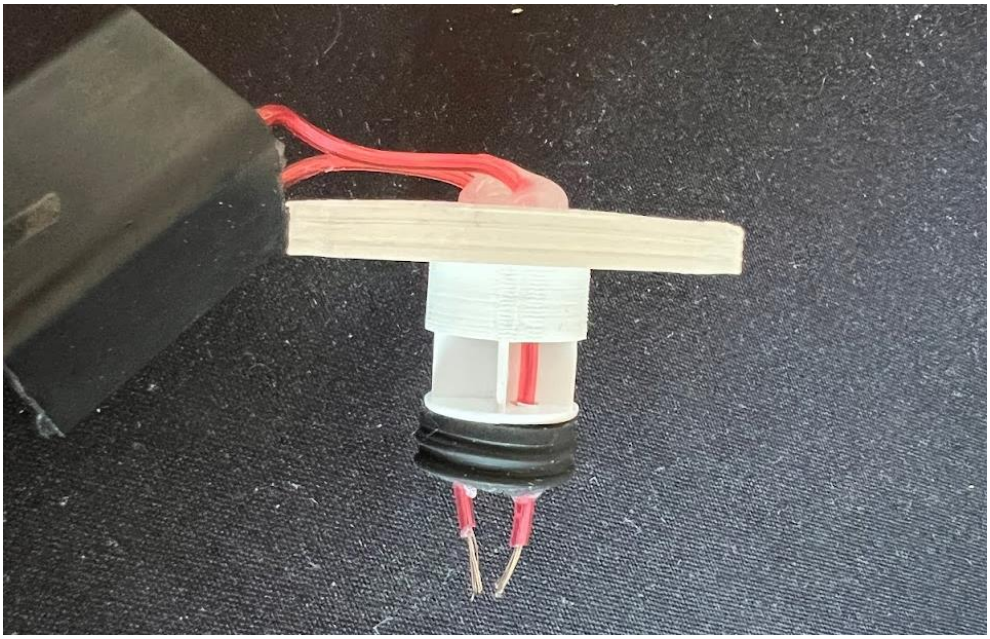

Figure 8 Installation of transformer, syringe endpiece and mount

After making sure you have enough cable length to mount the transformers in the main body, glue the cables in place. Also the two output cables should almost touch, to generate an arc for igniting the evaporated alcohol.

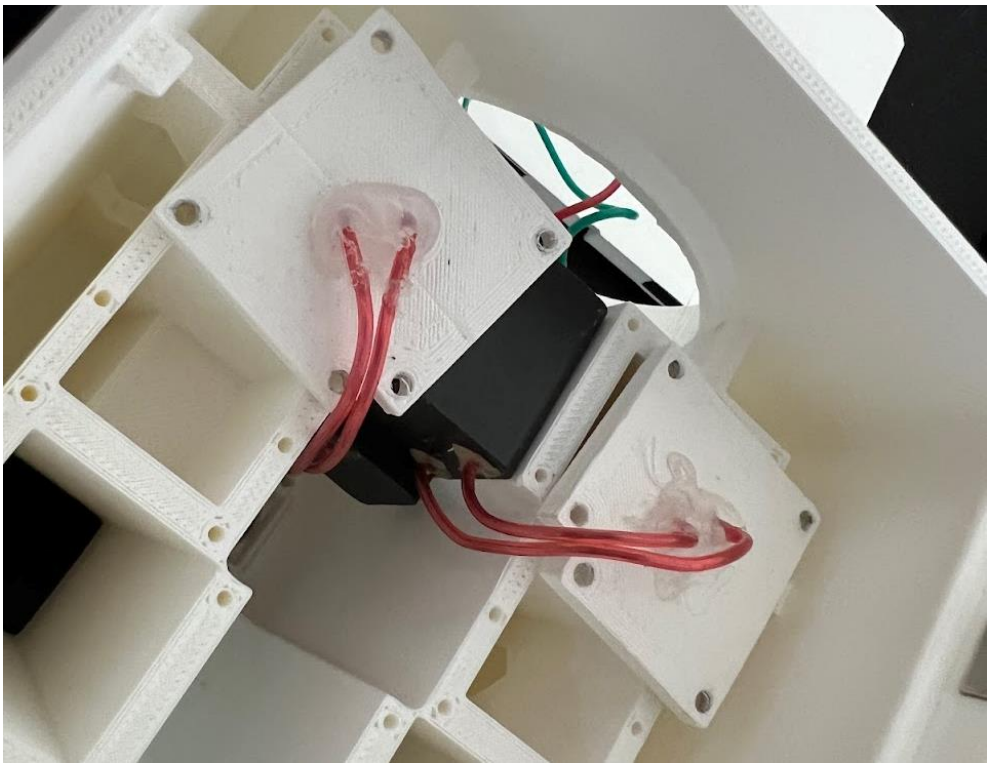

Figure 9 Checking the cable length

The input cables of the transformers will need to be extended for about 30cm, to be able to reach through the arms, I also recommend marking them by colour, to be able to differentiate them later.

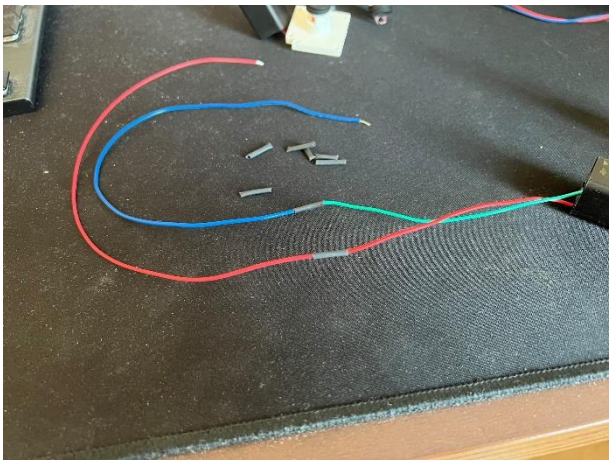

Figure 10 Cable Extension

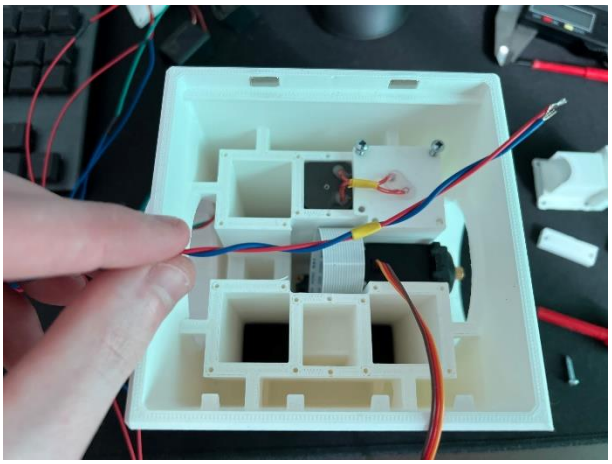

Figure 11 Colour-Coding

Now it is time to mount all the transformers and servo Y, while also putting the cables in the right places.

The slot for the connecters between servo and arm you probably need to slightly trim the plastic; I didn't get the perfect form into Fusion.

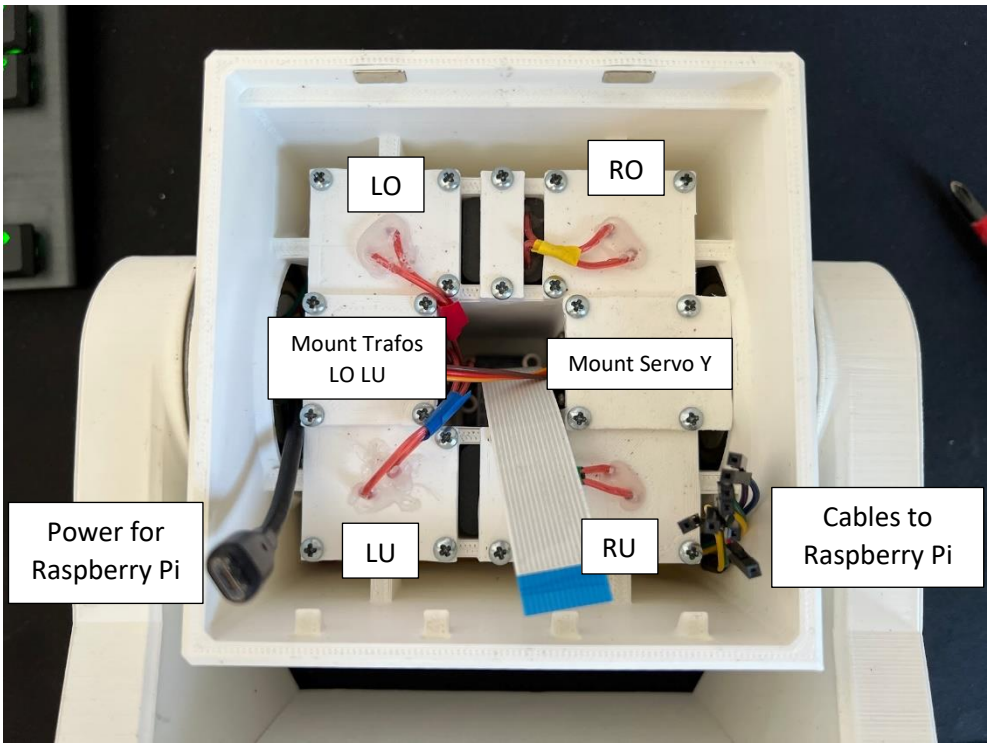

Figure 12 Main Body all parts installed

Mounting the Raspberry Pi

Screw the Raspberry Pi into the provided space on the lid, making sure to pay attention to the orientation.

Figure 13 The mounted Raspberry Pi

Plug in the cables and close the lid for now.

Adding the arms

Put the ball bearings into place and add the arms, while passing the cables through.

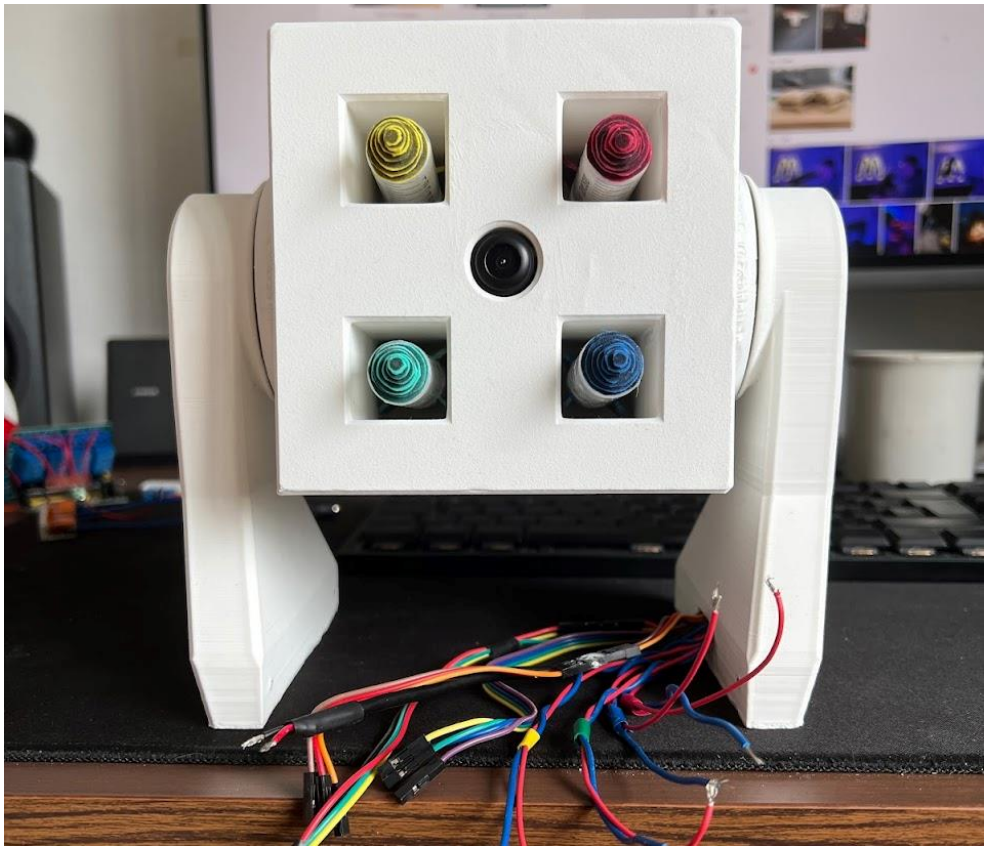

Figure 14 Arms and ball bearings added

Wiring the Base Mount

This is by far the most tedious task, fyi.

Start by feeding all the cables into the base mount and screwing base mount and the arms together.

Then tuck away all the cables, which will go to the relays, and install the servo controller and the adapter for the power supply.

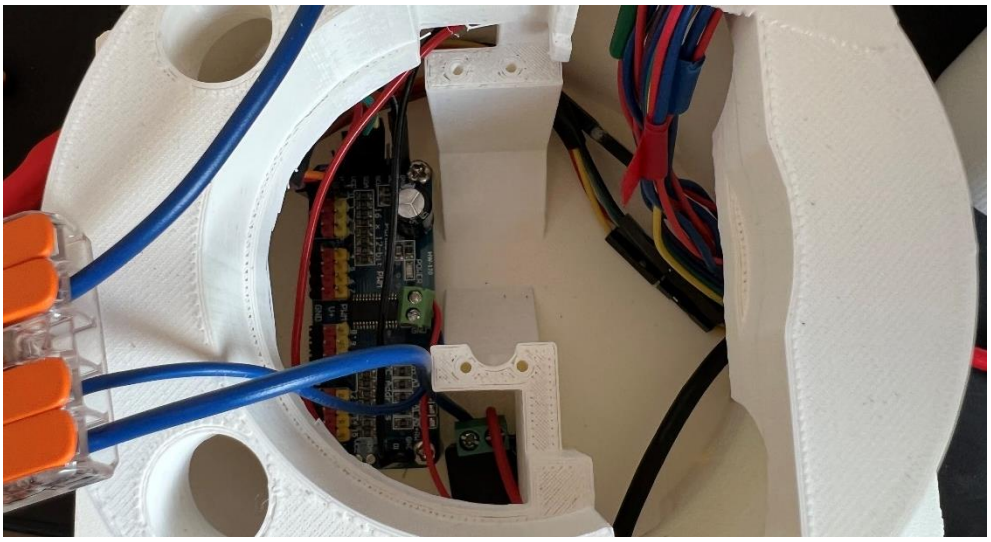

Figure 15 Servo Controller installed

Next install the relays and make sure to connect all the wires which will need to be connected. It is a very tight fit, you will need to bend the pins down slightly, to be able to shove the relays into its place.

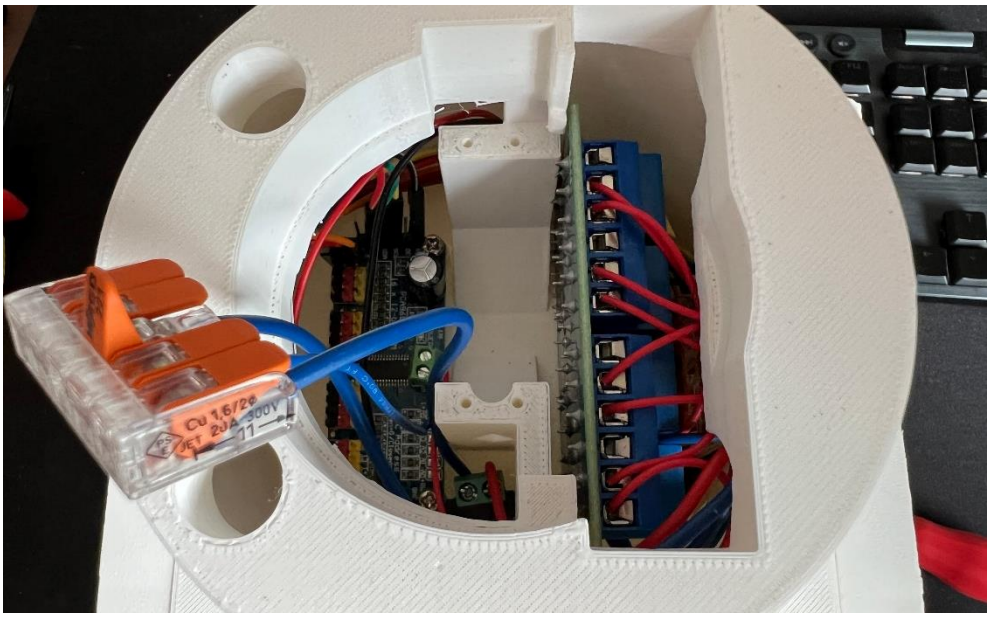

Figure 16 Relais installed

Solder the switches, to allow the three different stages (Raspberry Pi, Relais and Servos, Trafos) to be turned on separately.

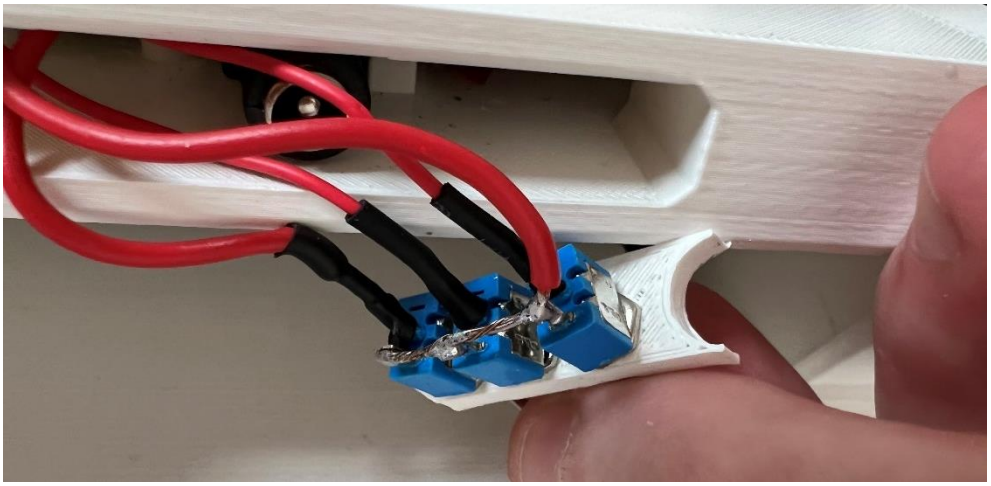

Figure 17 The switches wired and soldered

Put in the power panel, gluing is optional, but it should also hold up without.

Figure 18 Power Panel in place

Lastly put in Servo X and connect all the remaining cables. Tuck everything away so it will not interfere with the ball bearing in the base. Again, the connector between servo and base will need to be trimmed slightly.

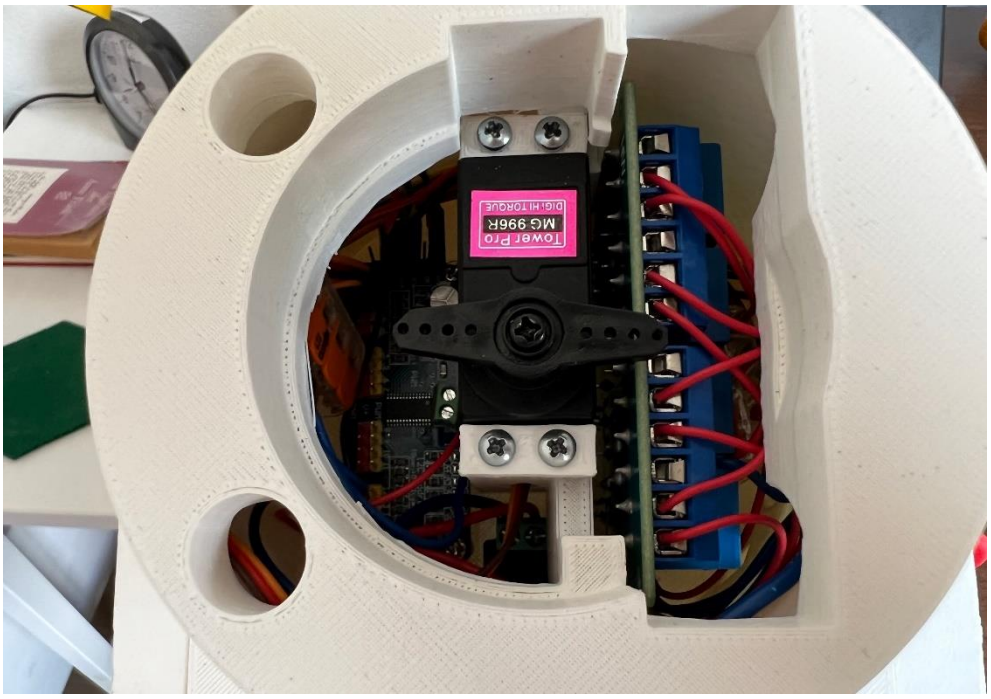

Figure 19 Servo X installed

After putting the remaining ball bearing onto the base you can slot in the finished construction and plug the cable in. I also put some of those felt gliders you usually use for desks or chairs between base mount and base to help with stability and I recommend anti-slip pads for under the base.

Figure 20 Finished AMRAS

Software 2

Running AMRAS

If everything is connected properly, it should now be as easy as running init_amras.py.

For calibrating I recommend plugging in the Raspberry Pi into a monitor with mouse and keyboard, afterwards it is easily possible to just run it using SSH.

You can alter the mid_offset (line 23 in init_amras.py) variable for calibration in case the camera is not mounted perfectly flush/centred.
Also, you might want to play around bit with the translation multiplier (line 21), but I decided it is good enough for me, also getting the PID controller to work is the goal anyways.

Command Line Arguments

It is possible to provide some arguments when running the program, to change the console output, the movement implementation...

Arg	Type	Required	Default	Description
-c, --cascade	String	True	-	Path to the Haar Cascade
-a, --armed	Bool	False	False	Option to arm the turret
-p, --pid	Bool	False	False	Option to use PID over more basic movement (currently not working)
-v, --verbose	Bool	False	False	Option to show console output
-I, --image	Bool	False	False	Option to draw image
-u, --undistort	Bool	False	False	Option to undistort image (need to calculate parameters first, also this makes the response time longer)

Examples:

python3 init_amras.py -c frontalface.xml -a True

Starts AMRAS without showing any output, tracking faces, and shooting when on target.

python3 init_amras.py -c ball.xml -i True -v True

Starts AMRAS while showing various values I the console, also enables camera feed and will track balls, recommended for calibrating the offset.

Included Haar Cascades

Currently there are two different Haar Cascades included in the repository, to detect different things.

Filename	Description
Frontalface.xml	Detects faces, provided by Adrian Rosebrock's tutorial
Ball.xml	Detects balls, can be used to fine-tune the servos without needing to move your face in front frantically.

If you want to create your own you can follow this guide using Matlab, but there are also other ways to create these.

Calibrate Camera

To calculate the camera matrix to undistort the camera run ‘camera_calibration.py’. I recommend not doing this on the Rapsberry Pi, as this can take a while.

This is only necessary if you are not using the same camera module as me.

For this to work you need to take a few pictures of a chess board with varying distances and angles and out them in the ‘Images’ folder in the same directory as the script. For better detection I recommend increasing the contrast and brightness and turning colour saturation to 0, to create a mostly black and white image, where the pattern is easily recognizable.

If the rest is extremely washed out it does matter, only the checkerboard is important.

If you do not have an actual chess board, you can print out the pattern on a piece of paper, if this is not the full 8x8 field you will need to change the pattern size in line 15.

Figure 21 Chessboard off center and away

Figure 22 Chessboard close and heavily distorted

This script returns a camera matrix and distortion coefficients, which will need to be updated in ‘init_haar.py’ in line 35.

To verify the result, you can run ‘test_calibration.py’ which will show all the files in the folder undistorted and time it.