Documentation

Julius Edqvist, Filip Michelsonas, Alexander Reider, Erik Österling

May 26, 2025

1 PCB

The PCB consists of three Arduinos and three DRV8871 motor controllers connecting to the arm which are all connected via USB to a Raspberry pi 5 for easy navigation and control. The PCB will be powered by an external 12V power source, and the different sensors will be powered by the Arduinos 5V source.

1.1 Pin Layouts

In this chapter the pin layouts on the PCB will be presented and the function aswell as where it leads on the Arduinos.

1.1.1 JST-PH8 2mm and PH2.0 8P

On the PCB there are two white eight-pin connectors, which are the connectors for the mechanical stops and signals for the radial and angular motors. The first pin starts at the bottom, and originally received a digital one when the motor has made a full rotation. However, due to an error when designing the PCB the first cable on the robot was switched with the second cable (D3 on Arduino). This means that the pulse is connected this to the second pin instead. How this changes the functionality is explained further down with the Arduinos.

The second (now the first pin, connected to D5) and third pin (D2) recieve a continuous square wave with the movement of the arm. These waves are phase-shifted to each other with the shift being to the left or right depending on what direction the arm is moving. If one signal is a digital 1 whilst the other signal is rising it is moving forwards and vice versa. To summarize, the first pin now recieves a continuous square wave, the second pin receives a pulse whenever the motor has made a full rotation and the third pin receives a phase shifted square wave compared to the first pin.

The fourth pin is 5 V power to the signals mentioned above with the fifth and sixth pin being ground. Pin seven (D4) is an input for the mechanical stop, which will output a 1 when it is reached using the IR emitter HOA-1871-031. The eighth pin is a 5 V power connector for the IR emitter. One of these connectors is for the r-axis and the other is for the angular axis.

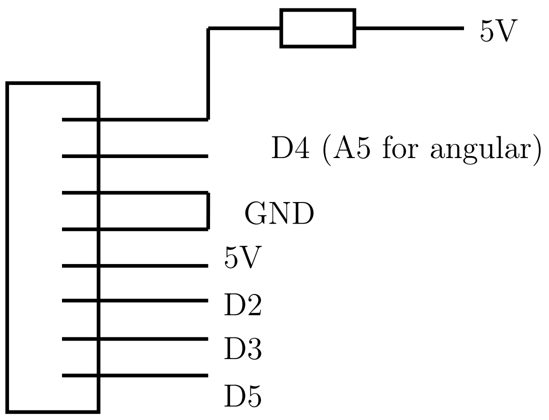

Figure 1: JST-PH8 2mm connections to Arduino

One of these connectors, specifically to the motor driving the r-axis was replaced with a PH2.0 8P connector due to loose cables.

1.1.2 8-pin Red Connector

The large red eight-pin connector is similar to the white, but with the IR emitter seperated into a smaller four-pin white connector. The four first pins are ground, and the fifth pin recieves 5 V power. Pin six and seven have the same function as the second and third pin on the 8-pin white connectors. The eighth pin is similar to the first pin on the white connector. Due to an error when designing the PCB an external pin width converter is used instead of an official connector. This connector only controls the z-axis.

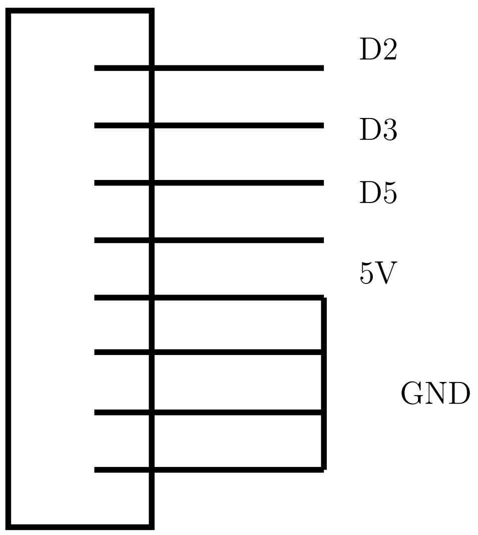

Figure 2: z-axis connections to Arduino

1.1.3 2-pin Red Connector

This connector leads to the brake, the first pin starting from the bottom is connected straight to 12 V whilst the second pin is connected to a IRLB8721 transistor which is controlled by one of the Arduinos. When the Arduino sends a digital 1 to the transistor, ground will open and voltage will pass through the brake.

1.1.4 PH-4A 2mm

This connector is attached to the same HOA-1871-031 as mentioned above. The first pin starting from the top connects to 5 V power with a 180Ω resistance. The second and third pins connect to ground and are connected to each other on the main board. The fourth pin sends the signal to the Arduino. The signal could either be a digital 0 when nothing is blocking the sensor or a 1 when the arm has reached the maximum distance.

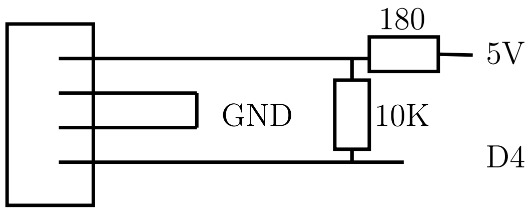

Figure 3: PH-4A connections to Arduino

1.1.5 Screw Terminals

The terminals are connected to the motor controllers and give 12 V power to the motors. More about the controllers is mentioned below.

1.2 DRV-8871 Motor Controllers

The motor controllers are connected to the Arduino which will control how the motors move. When one pin is on and the other off the motor will spin in one direction. When the opposite pin is on and the other off the motor will spin in the other direction. When both pins are on or off the motor will not spin.

1.3 Arduino Nanos

The Arduinos are connected differently depending on which axis it is connected to and there is one Arduino per axis. Pin D2 and D3 are INT pins, which are needed to use attachInterrupt on the received signals. As mentioned above, two cables switched positions on the connector due to the pulse needing an attachinterrupt. This causes one of the square waves to connect to D2 and the other to connect to D5 whilst the pulse is connected to D3. Pins D7 and D6 were connected to the motor controllers, however these pins could not deliver a PWM signal which is needed to contol the speed of the robot. These pins were later connected to D9 and D10, and switched to inputs.

The Arduino for the angular motor could not read the mechanical stop signal from the motor on the D4 pin. This pin was later connected to the A5 pin to solve the issue, meaning A5 reads the signal instead. Pin D13 is connected to the LEDs.

The z-axis has a separate brake, which is connected to pin D8 on the Arduino. This pin is connected to the transistor which will allow current to flow if the Arduino sends a digital 1.

2 User interface

The user interface consists of a frontend and a backend. Useful information about them will be provided below.

2.1 Frontend

Button	**Function**	**More information**
Rectangular chamber buttons	Used to select a slot to pick from or deposit to.	If something is to be deposited into the process chamber, "P" is clicked.
Clear	Deletes all log entries	The log has a maximum amount of 2000 entries and should be cleared periodically. The 2000 entry limit can be changed in `Control.vue`.
CLEAR	Clear the next action from the queue.	If the wrong chamber button is pressed this is useful.
STOP	Stops the moving robot.	Can be pressed during any type of movement to completely stop the robot.
RESET	Resets the position of every axis to 0.	If the robot malfunctions or moves in unexpected ways it can be beneficial to reset it.
RESUME	Press to resume movement after pressing STOP.	Also works for resuming if stopped during RESET.
RUN NEXT	The movement command currently queued up is executed.	**Important**: do not initiate a new movement command before the robot has finished the previous. If this is done the robot must be stopped and reset.
LOG POS	The position of all axis is logged.	Good to use for adding new positions.

Table 1: Interface buttons and their corresponding functions.

Choosing only one location so that `Next action` is displaying [Location] will make the robot pick up a sample from said location and then move to the 0-position with said sample.

Choosing the same location twice so that `Next action` is displaying [Location => Location] will make the robot deposit the sample it is currently holding in the chosen location.

Choosing two different locations will result in the robot picking up a sample from the first location and depositing it in the second location, after which it returns to 0.

2.2 Backend

The Backend consists of all the server side code and communication between the user interface and the Arduinos.

2.2.1 Startup

The project is managed by systemd and can thus be managed by using the cli `systemctl`. The main commands used are

- `systemctl start backend`
- `systemctl stop backend`
- `systemctl restart backend`

The same commands are used to start/stop/restart the frontend, the only difference is that "backend" is exchanged for "frontend".

2.2.2 Add positions

To add a position that the robotic arm should be able to move to one needs to add an entry in the file `/backend/routes/Arduino.js` in the `position_reference` variable. The command should be either in the form of an array or a single string where:

- Array: The array should consist of several sub-arrays where each sub-array will be a new command. The sub-array should consist of two elements where the first indicate which position the engine should move to and the second element represents the id of the Arduino.
- String: If the command is in the form of a single string, then this command will be broadcast to all available Arduinos.

3 Arduino code

Each Arduino runs a separate program that is uploaded from the Raspberry Pi. Useful information for communicating with the Arduino and editing the code is provided below.

3.1 Input signals

List of signals found in table 2.

Input signals are case insensitive. An input signal that does not match any of these formats will likely result in a crash, because the program treats all messages except these four known messages as if they were numbers.

3.2 Output signals

List of signals found in table 3.

3.3 Control system paramteters

List of parameters found in table 4.

This list of parameters does not include the various arbitrary numbers used in the error detection. Said numbers are written into the code in each program.

Message	Function
STOP	The Arduino's missionIndex is set to 0. The motor will recieve no signal, and stops. The Arduino reports "stopped".
RESET	The Arduino's missionIndex is set to 2, and it attempts to find its starting position. The motor moves backwards until it recieves feedback on a particular pin, and then sets its position, its missionIndex, and many other variables to 0. The Arduino reports "done" when finished.
Any integer	The Arduino's missionIndex is set to 1, and its target location is set to the value of the input integer. It uses a control loop to move the motor to the target location. Once the target location is reached, the missionIndex is set to 0 and the Arduino reports "done".
Resume	The Arduino's missionIndex is set to whatever it was before it was stopped or stopped because of an error. It resumes whatever mission it had previously.
Request_pos	The Arduino sends a message on the form "Current position: " and the Arduinos current locationNumber.

Table 2: Input signals for the Arduino. Case insensitive.

3.4 Notable differences between Arduinos

For the R motor, the move mission is be considered to be completed if the motor is stuck but is close enough to its target location. This was added because this usually occurs when the motor is stuck against the sample, it is often able to pick it up in this case.

For the Z motor, a voltage is applied to the LockPin to turn off the lock when the motor is meant to move. When the motor is not meant to be moving, no voltage is applied to the lock pin, turning the lock on.

For the Z motor, the move mission is considered to be complete as soon as the motor has overshot, not when it is standing still and within some set margins.

Message	**Function**
done	Sent when the Arduino successfully completes a move mission (index 1) or a reset mission (index 2).
error : veryStuck	Error message - the Arduino has encountered an error and has stopped, but can resume its mission if the resume command is recieved. The Arduino suspects that the error is caused by it getting stuck on an obstacle.
error : takesTooLong	Error message - the Arduino has encountered an error and has stopped, but can resume its mission if the resume command is recieved. The Arduino has no idea what caused the error. It is likely caused by loose cables. It is likely that cables should be adjusted and the whole system restarted.
error : varvInterruptSpam	Error message - the Arduino has encountered an error and has stopped, and should not resume its mission until it has been reset. The Arduino has recieved far too many signals on the lap interrupt feedback, a problem that is likely caused by loose cables. Cables should be adjusted and the whole system restarted.
Current position: *number*	Returned when the Arduino recieves "request_pos" (case insensitive) from the backend. Contains the Arduinos current position.

Table 3: Output signals from the Arduino.

Variable name	**Function**
P	Proportional term in the control loop. Currently set to be very high, but its effect is capped by the generalSpeedFactor.
I	Integrating term in the control loop. Currently set to 0, but the code to make it work still exists to make it easy to change.
D	Differentiating term in the control loop. Currently positive, in order to resist friction, and because it for some reason works well.
antistuckGrowthRate	Rate of growth for the custom antistuck term, measured in change per second.
antistuckMaxSpeed	The speed which, if the motor is moving fewer units per second, the antistuck bonus will increase.
antistuckMaxDist	The distance which, if the motor is further away from the target location than this, the antistuck bonus is always set to 0.
generalSpeedFactor	The motors maximum speed. A factor applied to the motor signal, after the motor signal has already been clamped between 0 and 1.
forwardsMargin	The distance by which the motor is allowed to overshoot, and still be considered to have completed a move mission.
backwardsMargin	The distance by which the motor is allowed to undershoot, and still be considered to have completed a move mission.
loopsPerLongagoPositionUpdate	How often the variables longagoPositionOne, Two and Three are updated. Affects how long the Arduino needs to be still in order to be considered to be stuck.

Table 4: Parameters in the different Arduino programs.