Scriba Robot – Technical Design

This document aims to describe the technical design of the Scriba robot. It includes mechanical design, electronics and code.

Summary

1 Design Overview

1.1 Drive unit

1.1.1 Overall design

The traction and steering motor are assembled in a “drive unit” placed at the front of the robot.

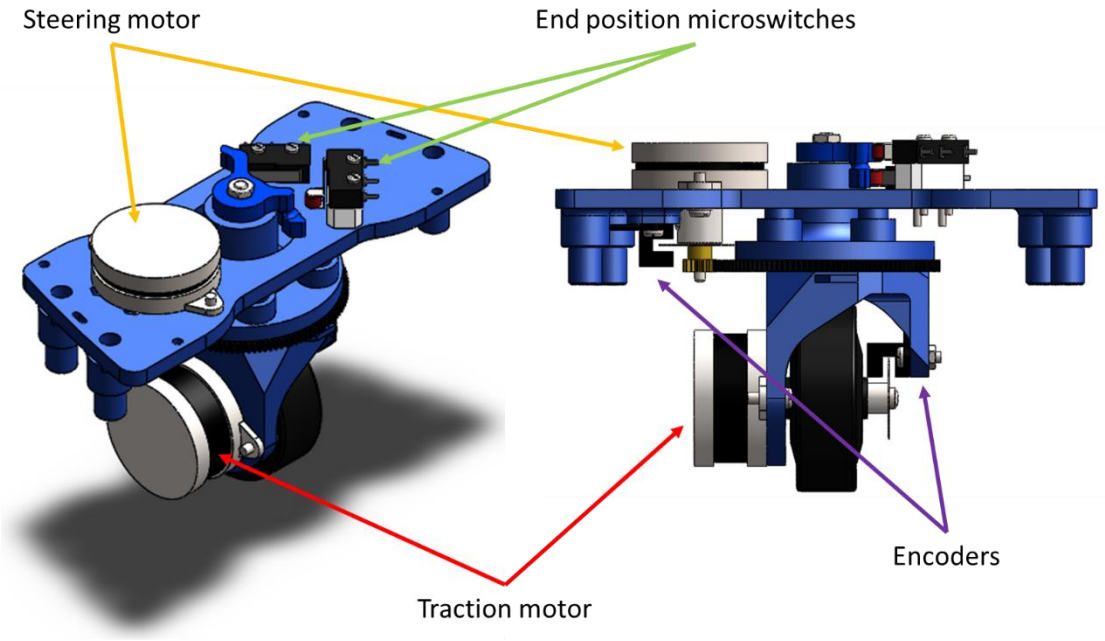

Figure 1: Drive unit

The driving wheel is powered by a stepper motor (14HR08-0654S) with a step angle resolution of 0.9 deg. In addition an encoder counts the rotation angle with a resolution of 1.8deg.

The driving wheel is steered by a stepper motor (14HR05-0504S) with a step angle resolution of 0.9 deg. In addition an encoder counts the rotation angle with a resolution of 1.8deg. The system is designed to rotate the driving wheel +/-90°.

Taking in account gears and microstepping configuration (r_{ms}), the relation between steering motor step and wheel angle (φ) is:

$$\varphi = steps * angle\ by\ step * r_{ms} * \left(\frac{z_{mot}}{z_{gear}}\right)$$

$$\varphi = steps * r_{ms} * 0.9 * \left(\frac{16}{120}\right) = steps * r_{ms} * 0.12$$

To recap the possible results:

Microstepping configuration	Wheel angle (φ) for 1 step in degrees
Full step	0.12°
1/2 step	0.06°
1/4 step	0.03°
1/8 step	0.015°
1/16 step	0.0075°

Two microswitches are used to detect the end position for calibration purposes: the starting angle of the wheel is unknown and a calibration is required before driving the robot. The microswitches are placed on slotted holes and can be adjusted.

1.1.2 Electronics

The stepper motors are controlled by an Arduino Nano and two A4988 modules. All those elements and connections for encoders, microswitches, and so on are integrated into a PCB (named "Steering Board").

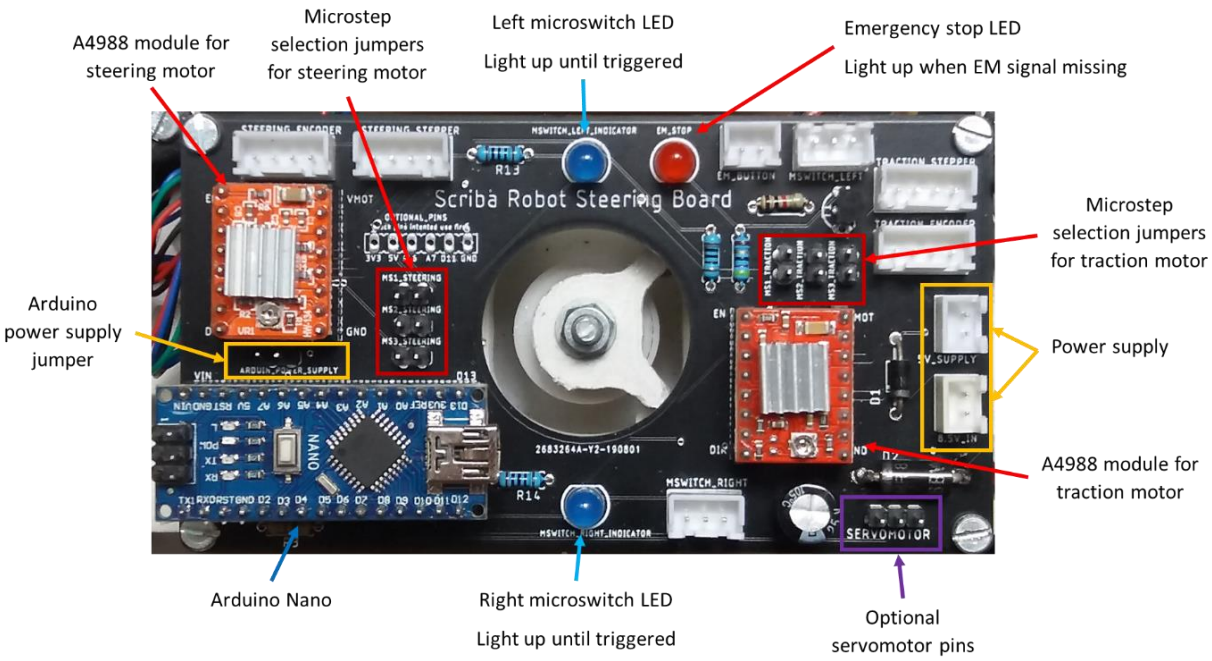

Figure 2: Steering Board

The board needs 5V and 8.5V power supplies. The 5V can be supplied by the Arduino if the Arduino power supply jumper is connected.

1.1.2.1 Microstepping selection

Stepper microstepping can be configured by the use of jumpers on the board. Connecting a jumper sets the microstep pin to high.

The truth table for the MS pins logic is the following:

MS1	MS2	MS3	Microstep Resolution
Low	Low	Low	**Full step**
High	Low	Low	**Half step**
Low	High	Low	**Quarter step**
High	High	Low	**Eighth step**
High	High	High	**Sixteenth step**

1.1.2.2 Emergency stop

An emergency stop feature prevents the motors from running without a 5V input on the emergency stop connector. The emergency stop red LED lights when the signal is absent.

1.1.3 Program & Control

The unit is driven by an Arduino Nano. The Arduino is connected to the main computer of the robot, the Odroid-XU4 by USB.

The program on the Arduino connects to a serial node of the ROS main program.

1.2 Printing system

1.2.1 Overall design

The printing system is assembled on a rotating support (rotating around the robot kinematic center) to correct the alignment if needed.

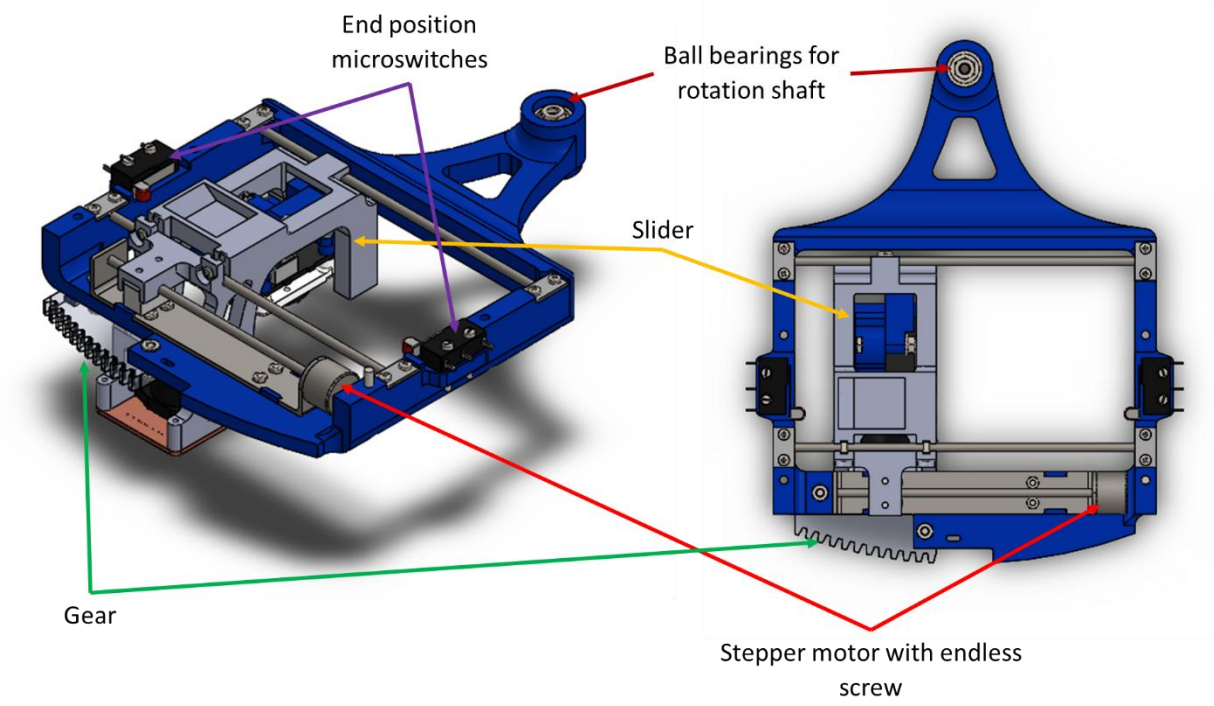

Figure 3: Printing system

The printhead is placed on a "slider", giving the system an 83mm sweeping length. The slider is powered by a stepper motor (DC 4-9V Drive Stepper Motor Screw With Nut Slider 2 Phases 4 Wires) with a resolution of 0.18°/step. The endless screw has a resolution of 0.5mm/turn making the overall system to have a resolution of 0.025mm/step.

Taking in account the microstepping configuration:

Microstepping configuration	Slider displacement for 1 step
Full step	0.025mm
1/2 step	0,0125mm
1/4 step	0,00625mm
1/8 step	0,003125mm
1/16 step	0,0015625mm

Two microswitches are used to detect the end position for calibration purposes. The starting position of the stepper motor is indeed unknown and needs to be calibrated.

1.2.2 Printhead

The printhead used by the robot is an inkjet printhead HP 51604A. It has been chosen for its ease of use and low price. Hacked inkjet printhead with easy controls are difficult to find. The HP 51604A (and the HP C6602) have a good documentation available online.

Most information on how to control the printhead came from Inkjet Printer For the RepRap by Amberish Jaipuria (https://reprap.org/mediawiki/images/1/1a/Inkjet.pdf), itself using as a primary source Inkjet Applications by Gilliland, M.

The printhead can only print in one color (black) with no possibility of half-toning. The printhead features 12 nozzles spanning over 3.2mm.

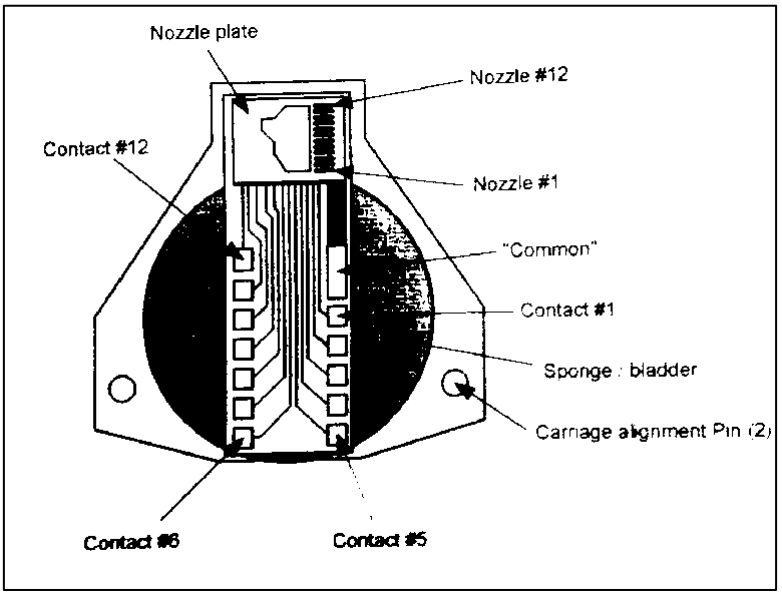

Figure 4: HP 51604A Printhead contacts & nozzles

Each nozzle has a 65Ω resistor and fires when a pulse is put across the resistor. The total energy released during the pulse has to be around 40µ Joules.

Taking in account the following equation:

$$Pulse\ width = \frac{resistance * energy}{voltage^2} = \frac{0.65 * 0.00004}{voltage^2}$$

Gilliand M. calculated the optimal pulse widths and voltages and provides the following table:

Voltage (VDC)	Pulse width (µs)
20	6.5
21	6
22	5.5
23	5
24	4.5

Those pulse width are optimal. A too short pulse time can result in low droplet quality or clogged nozzles. A too long pulse time can burn the nozzle resistor.

It's also recommended to wait at least 800µs before firing again the same nozzle. Two adjacent nozzles cannot be fired simultaneously and only two nozzles on the whole printhead can be fired simultaneously. Moreover it is recommended to wait 0.5µs between each nozzle pair being fired.

A PCB ("Printhead Contact Board") allows for a stable and easy connection between the nozzle contacts and the wires from the controller.

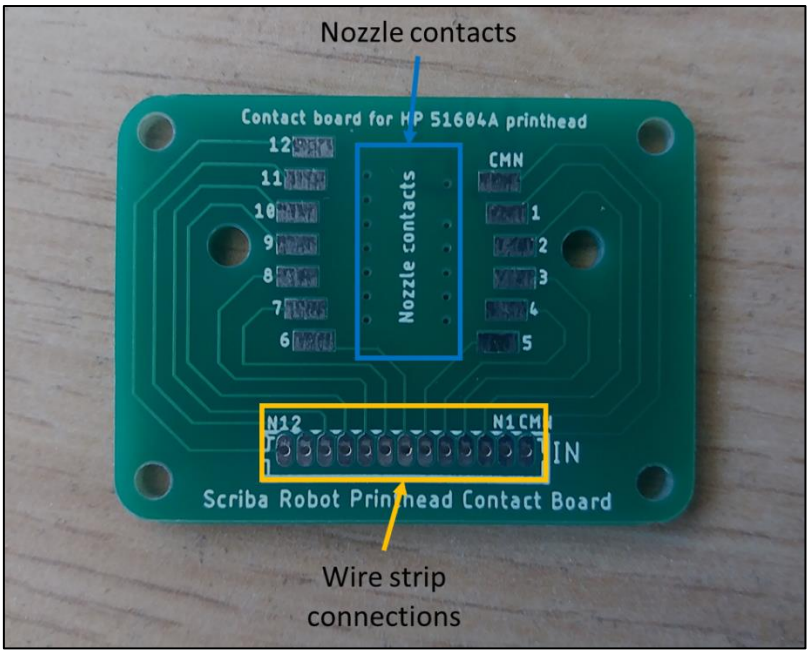

Figure 5: Printhead Contact Board

Wires for contacts are soldered and guided through the holes of the PCB.

1.2.3 Printing slider

The printhead is placed on a slider, allowing the printing pass to be wider. The slider includes the printhead and its contact board, as well as a camera for drift and misalignment detection.

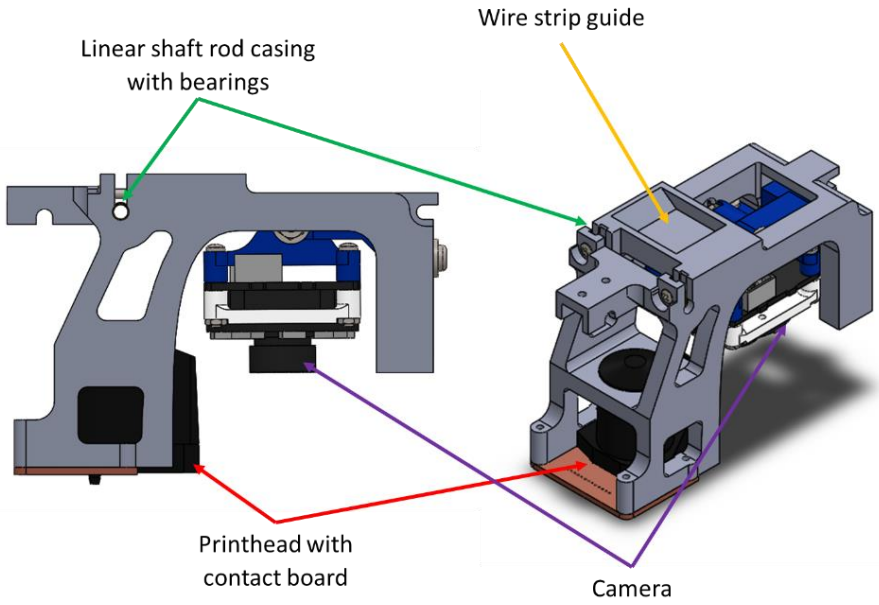

Figure 6: Printing slider

The camera height and angle can be adjusted by two screws. The camera is an ELP USB camera with a 2.1mm objective (ELP-USBFHD01M-L21). Objective value was chosen using a focal length calculator.

1.2.4 Alignment & Correction

The printing system needs to correct the eventual robot misalignment and drift from the intended position to ensure a good print quality. This error correction can be split in three items

- Forward/backward drift ($e_{\vec{x}}$)
- Lateral drift ($e_{\vec{y}}$)
- Alignment (λ)

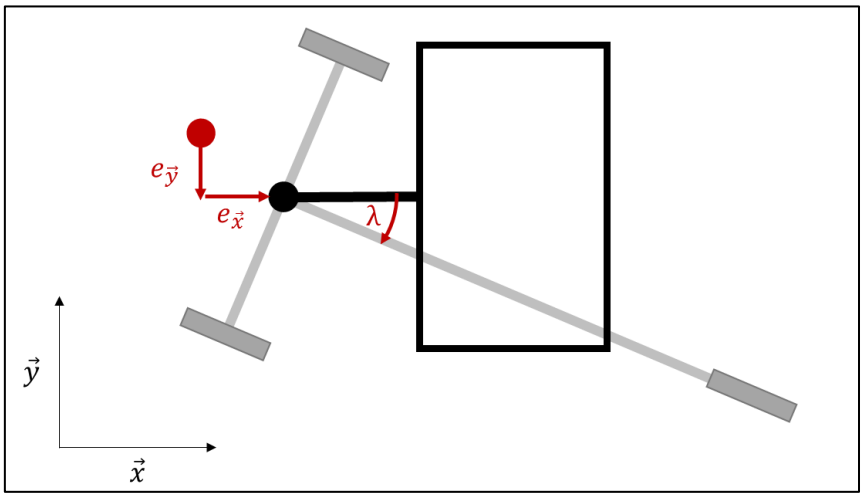

Figure 7: Printing system pose error

1.2.4.1 Forward/backward drift

There is no actuator for correcting an error in the printing direction (robot driving direction). The system relies instead on “shiftting” the nozzles used. A least 1 nozzle on each end of the printhead needs to be reserved for this operation and therefore reduces the overall nozzle span.

1.2.4.2 Lateral drift

Lateral drift is simply corrected by adding an offset on the slider starting position. A specific length on both sides has to be reserved for this operation and therefore reduces the slider course.

1.2.4.3 Alignment

The printing system can rotate slightly around the robot kinematic center to correct its alignment. A servomotor is used to control the rotation. The gear ratio between the servomotor and the printing system is 18/244.

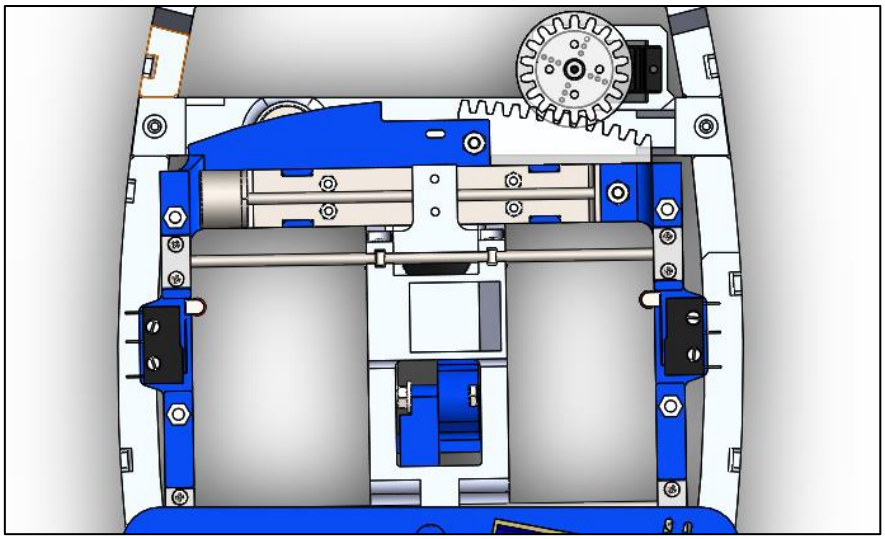

Figure 8: Printing system gears & servomotor, top view

1.2.5 Electronics

The printhead nozzles are controlled by two ULN2803 Darlington arrays to interface the Arduino outputs with the 20V power supply. For each nozzle a pull-down 1kΩ resistor is added to ensure a good control of the firing pulse time.

The stepper motor for the slider is controlled by an Arduino Nano and an A4988 module.

All those elements, as well as the 20V power converter and the connections for the end position microswitches are integrated into a PCB (named “Printhead Control Board”).

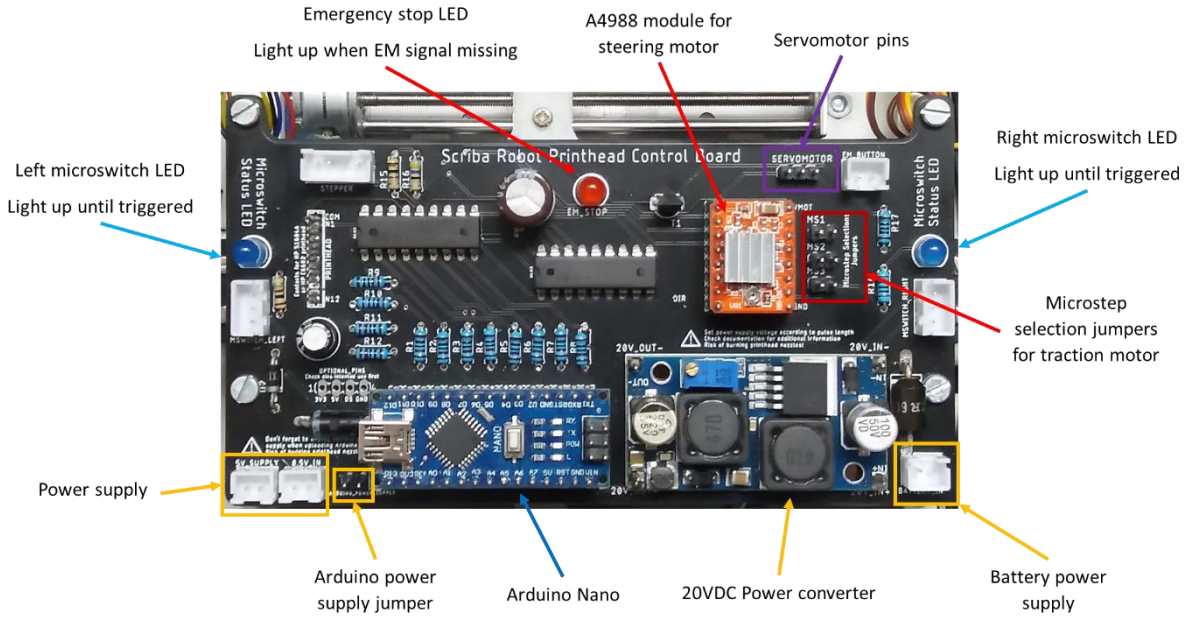

Figure 9: Printhead Control Board

The board needs 5V and 8.5V power supplies as well as a connection to the battery. The 5V can be supplied by the Arduino if the Arduino power supply jumper is connected.

1.2.5.1 Microstepping selection

Stepper microstepping can be configured by the use of jumpers on the board. Connecting a jumper sets the microstep pin to high.

The truth table for the MS pins logic is the following:

MS1	MS2	MS3	Microstep Resolution
Low	Low	Low	**Full step**
High	Low	Low	**Half step**
Low	High	Low	**Quarter step**
High	High	Low	**Eighth step**
High	High	High	**Sixteenth step**

1.2.5.2 Emergency stop

An emergency stop feature prevents the motors from running without a 5V input on the emergency stop connector. The emergency stop red LED lights when the signal is absent.

1.2.6 Program & Control

The unit is driven by an Arduino Nano. The Arduino is connected to the main computer of the robot, the Odroid-XU4 by USB.

The program on the Arduino connects to a serial node of the ROS main program.

1.3 Localization system

Scriba robot uses two cameras for its localization. They are placed at the front and rear of the robot on a rotating support for easy adjustment. A LED ring provides good and adjustable lighting conditions.

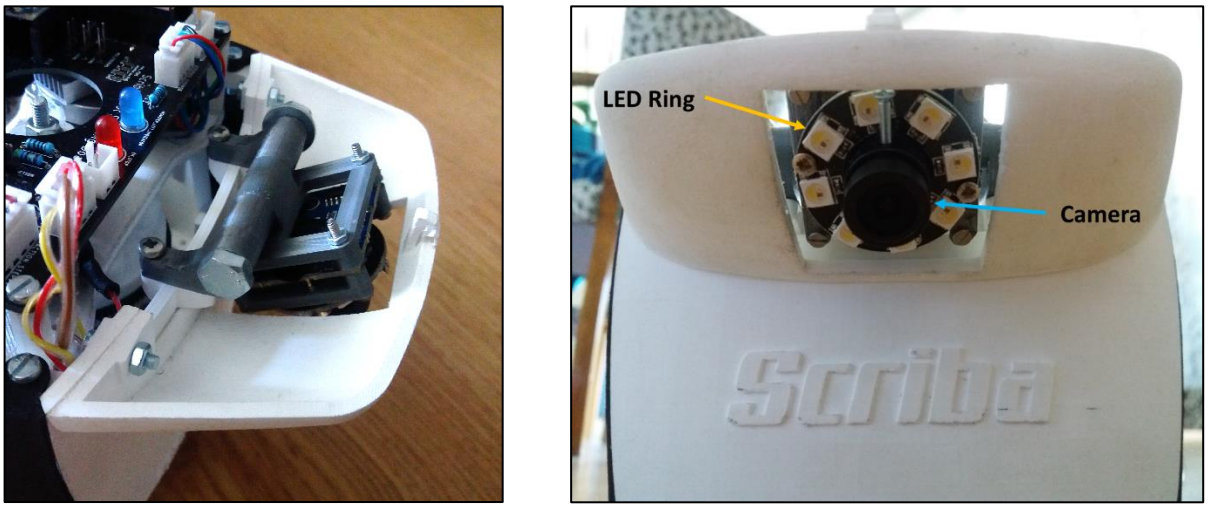

Figure 10: Camera on its support with LED ring

The cameras used are ELP USB cameras with a 2.8mm objective (ELP-USBFHD01M-L28). Objective value was chosen using a focal length calculator (closest value available).

Object distance	Lens focus average (mm)
50	2.85
70	2.89
120	2.975

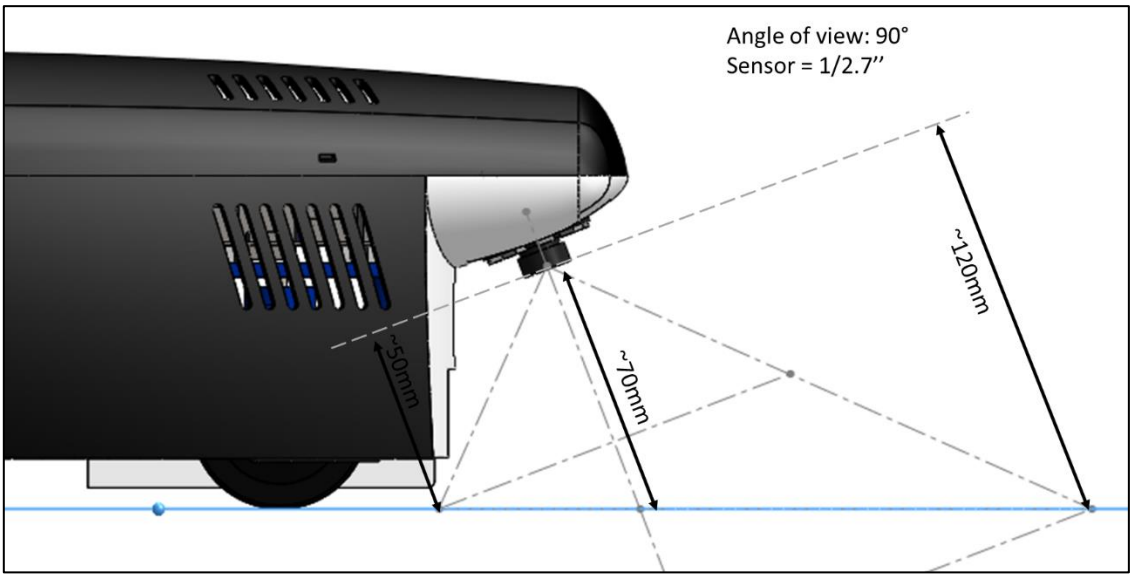

Figure 11: Camera field of view and object distances

1.4 Power distribution

Three different voltages are used by the robot components: 5V, 8.5V and ~20V. Power is provided by a 3S battery (Black Lithium 11.1V 1500mAh) and then converted and distributed to the different elements.

The battery and converters are placed at the rear of the robot to keep the majority of the weight as close as possible to the kinematic center to minimize drift.

Figure 12: Battery case and connection

A first board connects to the battery and distributes the power to the power converters and battery level display. The output power from the power converters (5V and 8.5V) is then supplied to another PCB ("named PowerOut" or "Power Distribution Board") with screw terminals for easy distribution to other elements. This board also includes a 1000µF capacitor on each voltage to smooth current spikes.

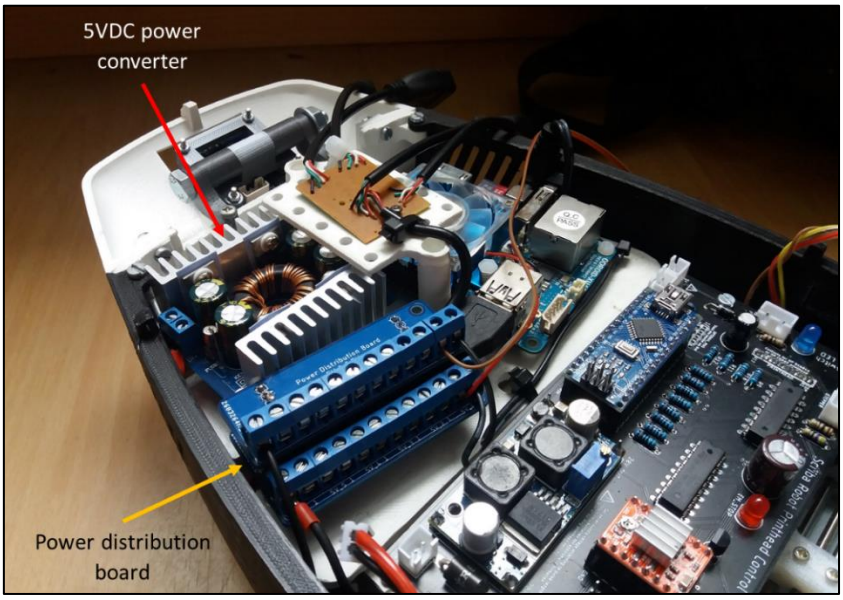

Figure 13: Power distribution board on the robot

The battery board will be replaced by a LVD (Low Voltage Disconnect) board with a relay/transistor in the future.

2 Robot Model

Before implementing any navigation feature, the robot needs to be modeled. The model is not only useful for visualization purposes but also essential for frame transforms and other calculations.

For a ROS implementation two models are needed:

- Kinematic model: needed for odometry and move command.
- CAD model (URDF): base on the kinematic model, needed for visualization and frame transforms.

2.1 Kinematics

The kinematics equations of the robot are needed to implement move command and odometry.

All kinematic equations are described in the document “Kinematics_Scriba”.

2.2 URDF

ROS uses URDF (**U**nified **R**obot **D**escription **F**ormat) files to describe robot models. Those models are described using a kinematic logic, with “links” (parts) connected through “joints” of different types (rotation, translation …).

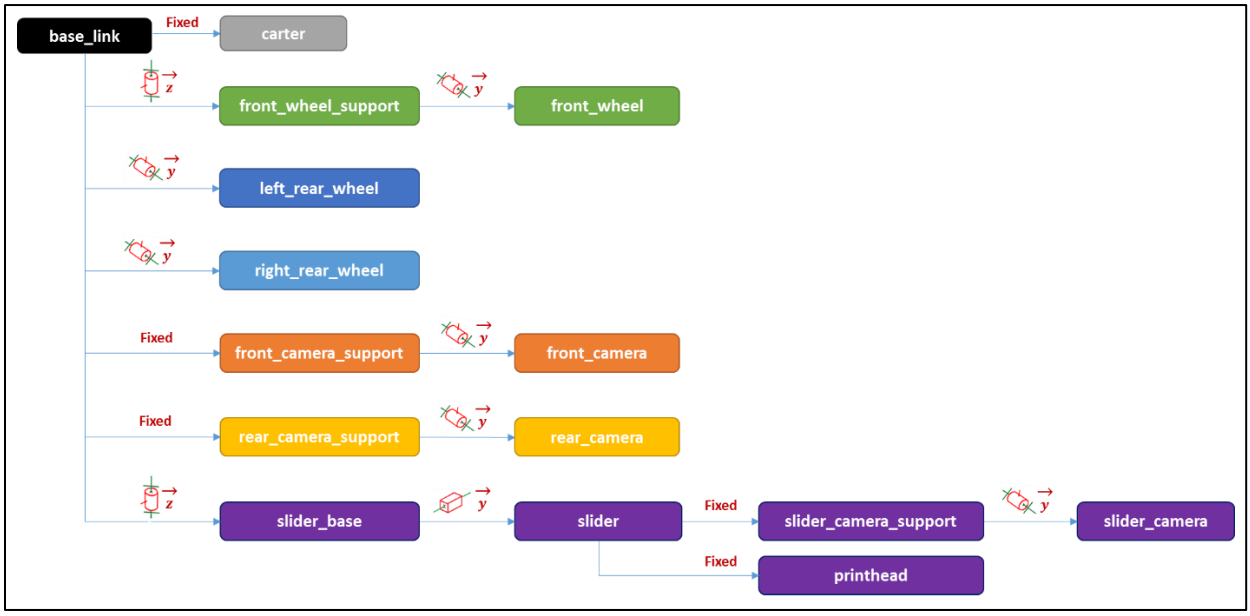

Figure 14: Scriba robot simplified kinematic diagram

The URDF file and other files needed for visualization are included in the ROS package “scriba_description”.

3 Navigation

3.1 Calibration

An accurate model is essential for odometry and navigation commands. The model therefore needs to be calibrated to ensure an optimal navigation.

The three variables to calibrate are:

- Front wheel angle offset
- Front wheel radius
- Robot's length (distance between the kinematic center and the front wheel)

All values are dependent on each other and as such it is possible to get a better calibration by repeating the operations a second time.

3.1.1 Front wheel angle offset

Before calibrating the front wheel angle offset it's necessary to run a script to put the wheel at position zero.

The robot main program is launched first using the bringup launch file:

```
roslaunch scriba scriba_bringup.launch
```

And the following script is run on another terminal (0 as wheel offset):

```
rosrun scriba scriba_front_wheel_calibration.py
```

The robot can then be placed on a straight and visible line (on a floor pattern for example) and driven in a straight line using the teleop keyboard.

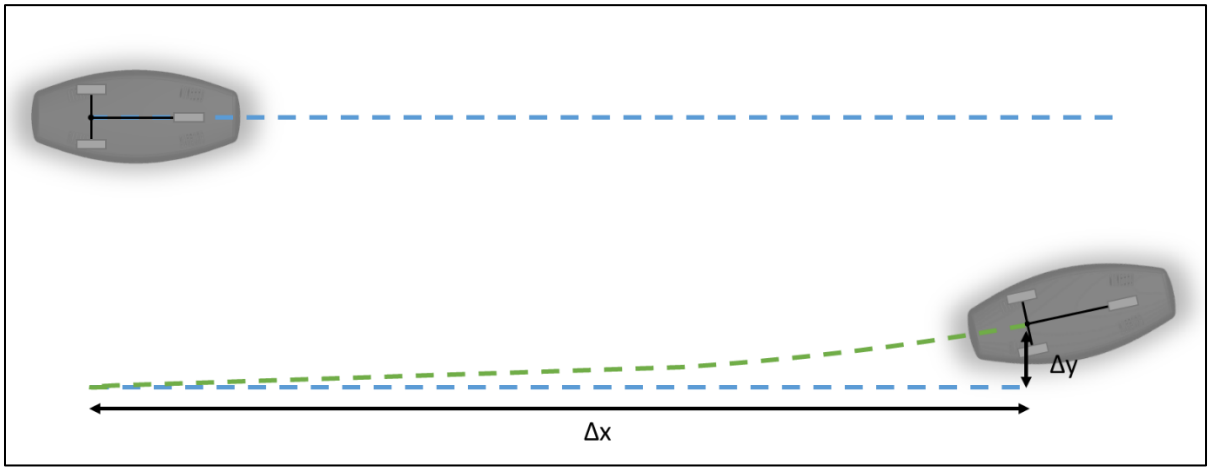

Figure 15: Difference between odometry and real value due to front wheel angle offset

By using the following equation (from the document "Kinematic_Scriba") the front wheel angle can be calculated and corrected by an offset:

$$\varphi = \arctan\left(L * \frac{2 * \arctan\left(\frac{\Delta y}{\Delta x}\right)}{\widehat{d_{wr}}}\right)$$

$\widehat{d_{wr}}$ being the reported x-displacement in the odometry in this situation.

3.1.2 Front wheel radius

Once the front wheel angle calibrated the wheel radius can be calibrated as well.

The robot main program is relaunched first using the bringup launch file:

```
roslaunch scriba scriba_bringup.launch
```

And the following script is run on another terminal (with the previous calculated offset as wheel angle offset):

```
rosrun scriba scriba_front_wheel_calibration.py
```

The robot can then be placed on a straight and visible line (on a floor pattern for example) and driven in a straight line using the teleop keyboard.

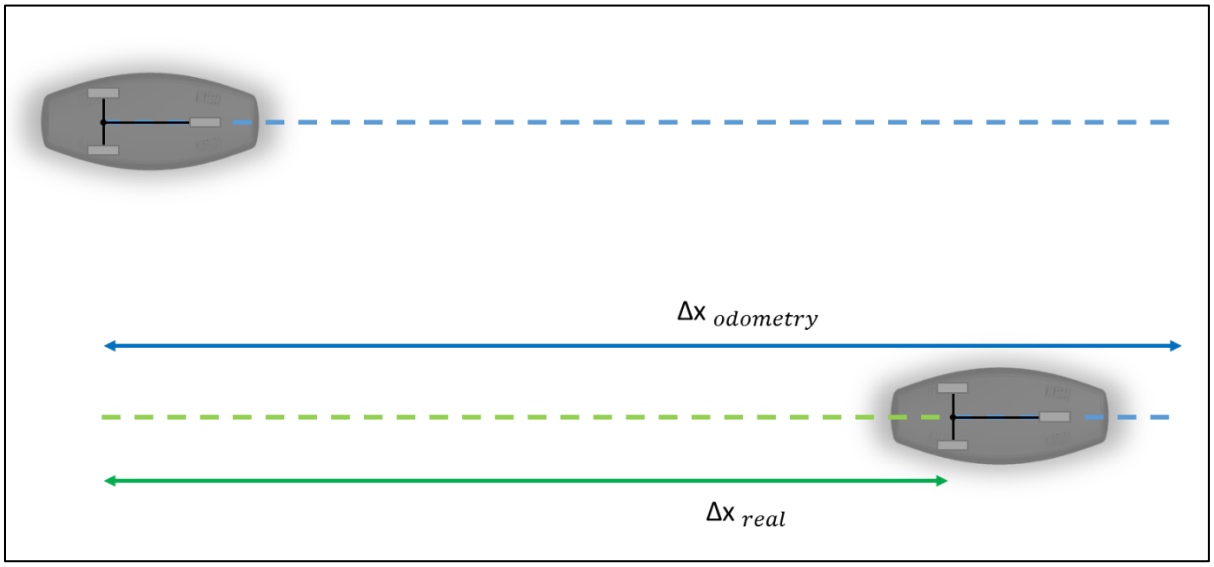

Figure 16: Difference between odometry and real displacement due to front wheel radius inaccuracy

The calibrated wheel radius is given by the following formula:

$$radius_{wheel} = radius_{wheel} * \frac{\Delta x_{odometry}}{\Delta x_{real}}$$

$\Delta x_{odometry}$ being the reported x-displacement in the odometry.

Δx_{real} being the measured x-displacement

3.1.3 Robot's length

At last the robot's length value needs to be calibrated. Length refers here to the distance between the front wheel and the robot's kinematic center.

The robot main program is relaunched first using the bringup launch file:

```
roslaunch scriba scriba_bringup.launch
```

And the following script is run on another terminal (with the previous calculated offset as wheel angle offset):

```
rosrun scriba scriba_front_wheel_calibration.py
```

The robot can then be placed on a straight and visible line (on a floor pattern for example) and driven in circle using the teleop keyboard (maximum rotation: front wheel angle of $\pi/2$ to rotate around the kinematic center).

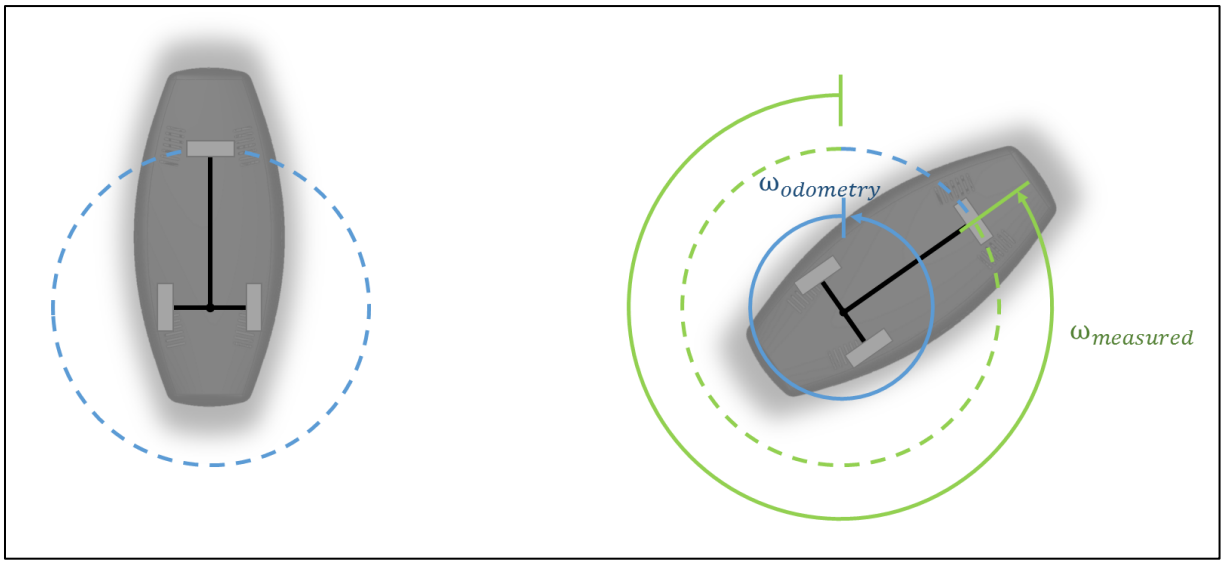

Figure 17: Difference between odometry and real angle due to robot length inaccuracy

The calibrated length is given by the following formula:

$$L = L * \frac{\omega_{odometry}}{\omega_{real}}$$

$\omega_{odometry}$ being the reported z-angle in the odometry (be careful, angles are reported as quaternion in ROS).

ω_{real} being the measured angle.

3.2 Odometry

The odometry (or "dead reckoning") refers to the estimation of a robot position through its motion sensors data. The odometry is a relative position estimation (relative to the starting position) and suffers from the accumulation of sensor uncertainties, preventing it from being efficient over a long time period. Odometry is often use in combination with an absolute position estimation (such as a camera using external element as reference points).

In Scriba, the odometry is computed by the main computer with data from the Arduino of the drive unit. The node "scriba_odometry_broadcaster" in the "scriba" package subscribes to the topic /data_odom to receive the odometry data.

The odometry data is transmitted on the topic in a custom message type "odom_data" (from the "scriba_msgs" package) including:

- Time stamp
- Front wheel angle (at the beginning of the sample)

- Front wheel angle (at the end of the sample)
- Front wheel traveled steps
- Sample duration

The position is estimated using this data and the previous position.

The odometry data currently uses only stepper motor commands. Encoders are wired but not yet used due to the limited memory available in the Arduino Nano of the drive unit. They will be added if the odometry quality needs to be improved.

3.3 Localization

The localization refers to the absolute position estimation (using external references) used to correct the odometry position estimation.

WIP – Algorithm is not finished yet.

3.4 Move commands

Two drive modes are used by the robot:

- Standard mode, used when turning and moving to the next printing position.
- Printing mode, used when printing: low speed and steps command.

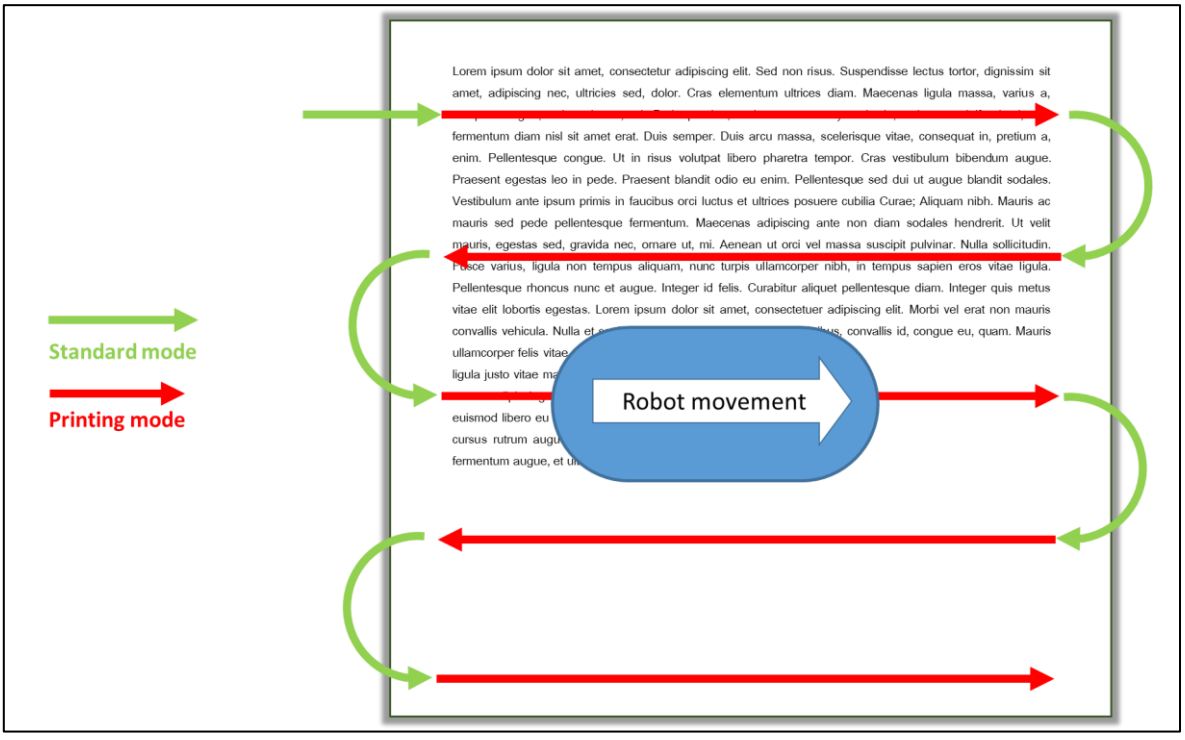

Figure 18: Robot driving modes

WIP – Printing mode not implemented yet.

Table of figures