Arm exoskeleton supporting two degrees-of-freedom made with LEGO and Raspberry Pi

Corentin Serrie, Barnabas Homola

June 26, 2020

Abstract

This report presents the construction and usage of an arm exoskeleton which was developed as part of the special course *Designing and implementing an arm exoskeleton prototype for rehabilitation of stroke patients* at the Technical University of Denmark (DTU). The presented final version of the exoskeleton is a result of a rapid prototyping process and still considered as a proof-of-concept which can be used as a base for future enhancements and research projects. It has two degrees-of-freedom (DOF): it allows the user to bend the elbow and allows longitudinal wrist rotation. The exoskeleton is built by using LEGO Technic bricks and EV3 sensors and actuators. The system is controlled by a Raspberry Pi single-board computer extended with a BrickPi board. It is capable of wireless communication to send sensor values and take actuation commands. As part of the project, the system was succesfully connected to a Unity program to control a simple game and nudge the user by using the motors in order to initiate given arm movements to achieve a better score. The Unity integration enables the exoskeleton to be used in virtual or augmented reality (VR/AR) environments.

1 Arm exoskeleton background and introduction

This design and development project was carried out as part of the ReHyb project, whose purpose is to develop a hybrid upper-limb exoskeleton that uses its own sensing and novel actuation capabilities for generating the digital twin of the user. [1] The aim of the project presented in this report was to create a system which can serve as base or potential probe for studies connected to the ReHyb project. A lightweight, cheap and easy to use arm exoskeleton system with Unity [2] integration can be used for prototyping AR/VR systems for stroke

patient rehabilitation.

There are other existent kits for serving similar purposes. The most notable system is the EduExo Robotic Exoskeleton Kit [3]. Even though this kit provides great functionalities and well-documented introduction to exoskeleton robotics, it is limited. It only has one degree-of-freedom at the elbow joint with a weak electromotor actuator. Furthermore it consists of multiple, non-integrated parts which means that the exoskeleton can only be used supported by a computer and is not ready to be worn by patients due to the limited structure and the absence of energy supply. Nevertheless it is an intuitive, quick and innovative way of exploring the possibilities of such system. The LEGO exoskeleton in question is also trying to aim for similar goals.

The goals of the development process were the following:

- Make a modular, extendable physical arm exoskeleton which supports two degree-of-freedom
- The system should be ready to be worn by patients or test participants meaning that the exoskeleton should have its own control unit and energy supply integrated
- The system should be able to connect and establish bi-directional communication with Unity

With a system which has the above mentioned features, research experiments and studies can be conducted to learn more about arm movement rehabilitation with the usage of serious gaming and AR/VR technologies.

1.1 Why LEGO Technic + EV3?

The physical prototype presented in this report is built of LEGO Technic bricks and using LEGO Mindstorms Evolution 3 (EV3) sensors and actuators. Multiple points are supporting this choice of using these building blocks:

- The parts are easy to access. These elements are widely accessible, ready to order for anyone, there is no need to go through complicated ordering processes and look up the specific retailers.
- The product becomes scalable. Since the building blocks are easy to access, there are no boundaries for scaling the product. With the correct manual and information, it is easy to quickly build multiple exoskeletons or ship the parts.
- The exoskeleton is modifiable this way. In case of a need for change, the product can be easily modified. These use-cases can be new modifications and features for new iterations but also new emerging requirements e.g. the patient's arm length or width.

With the LEGO Technic bricks, it is easy and fast to accommodate these changes without ordering or recreating different parts.

- The EV3 sensors are also easy to access, versatile, robust, relatively cheap and advanced enough for prototyping compared to their price.
- By using LEGO elements in the product, the exoskeleton can exploit the well-known positive effect of LEGO on children. For possible later use-cases with child patients, a product like this is more likely to be accepted than a traditional, stiff solution.
- Lastly, during the development of the exoskeleton, an unforeseen advantage has also risen. The ability to continue working and iterating despite the restrictions due to the COVID-19 pandemic in Denmark was also partly possible due to the easily accessible components. Without access to labs and special equipment, the development would have been much slower and cumbersome.

2 Physical device

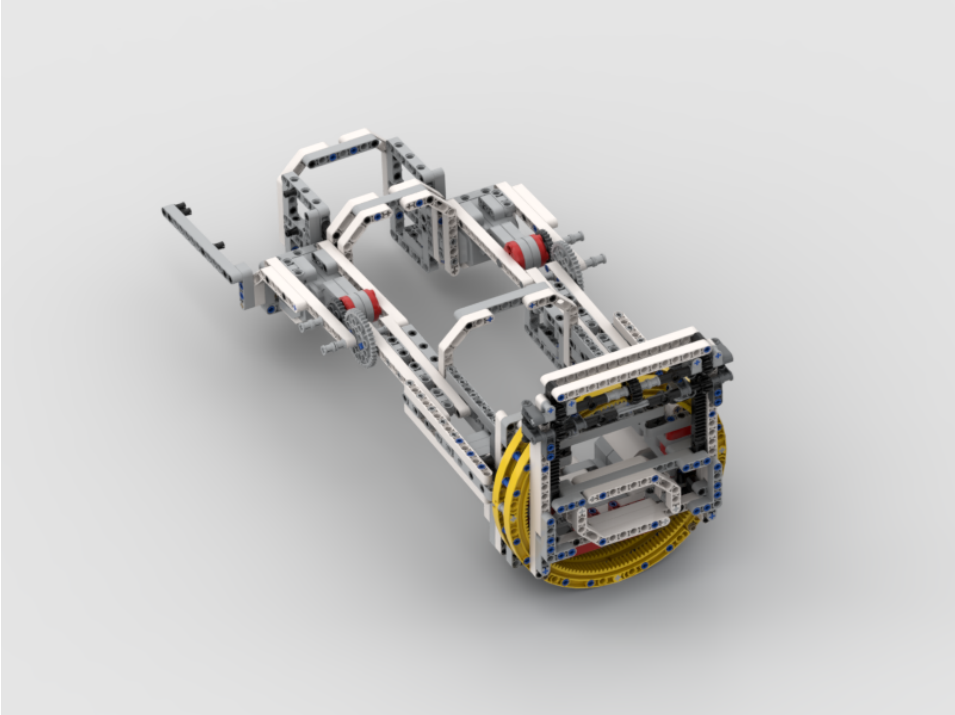


Figure 1: 3D rendering of the LEGO Exoskeleton made using BrickLink Studio

2.1 Manual

Available in the Appendix Section C is a manual describing how to assemble the exoskeleton step by step. It was made using the software BrickLink Studio [4], and if any step remains

unclear, a 3D model [5] can be opened with BrickLink Studio, to allow for a more detailed inspection of the model and all its parts. A rendering of the final model can be seen in Fig. 1.

2.2 Feature overview

As discussed previously, this exoskeleton has two degrees-of-freedom, one for the elbow, and one for the wrist. The mechanics for the elbow are pretty straightforward: since the axis to rotate around is available on both sides of the elbow, it is really easy to create a hinge joint on both sides that will allow for this rotation. Two motors were used, one on each side, to improve the strength of the input that the exoskeleton can give. One can also experiment with the gear ratios to either make it easier to move the exoskeleton, at the expense of weaker nudges, or the other way around. The current gear ratio was deemed satisfactory with the strength of the LEGO motors, but by simply exchanging the two gears on each side, it would make for weaker nudges, but less resistance to movement.

As for the wrist rotation, this was harder to implement. The main issue is that the axis of this rotation is inside the whole forearm, the hand, and even the arm and shoulder when the arm is extended straight. Thus a wheel with internal gear had to be used, sustained by four smaller gears at its periphery, to allow the user to slide his arm inside, while still being able to rotate the wrist part with the motor. This section only has one motor, and it is enough to give strong nudges to rotate your wrist. Once again, one could experiment with the gear used between the motor and the inside gears, to either make the nudges stronger and slower, or weaker and faster, while making rotating the wrist without the help of the motor either harder or easier, but the current ratio was deemed satisfactory.

The final interesting part is the wrist locker, at the front of the exoskeleton. It is basically a pincer with a locking mechanism that can be easily locked or unlocked one handed. Therefore the user can equip the entire exoskeleton by himself, using his free arm to lock everything in place. The locking mechanism can also be adapted to different wrist sizes, which makes the exoskeleton one size fits all.

As for the non LEGO features, there are three velcro bands used to strap the exoskeleton to the user's arm. Again, these work towards the effort of making the exoskeleton one size fits all. There are also some foam pads attached on the points of contact to the user's arm, especially on the wrist lock mechanism, as being constricted by LEGO plastic can be uncomfortable. Finally, as a result of the overall design, the exoskeleton is compatible with both arms, although the control unit needs to be replaced to the other side (See Section 3.1).

3 Electronics and software

3.1 Control unit

The exoskeleton is controlled by a controller unit consisting of a Raspberry Pi single-board computer and a BrickPi board [6] which enables the former to communicate with the EV3 sensors and motors. For this version of the product, an older, Model B+ revision 1.2 Raspberry Pi and a BrickPi3 v1.1 were used. The BrickPi3 set comes with an acrylic casing with LEGO interface making it possible to easily connect the control unit physically to the LEGO exoskeleton. Since these boards are not capable of WiFi communication, an additional nano USB wireless network adapter (Edimax EW-7811UN) was connected to one of the USB ports of the Raspberry Pi. An important factor was to preserve the advantage of the exoskeleton being a "one-piece", independent device. This means that everything is included in one single object, there are no wires running to or from the device and every part of the system is located on the patient's arm. In order to achieve this, the control unit is mounted on the outer side of the upper arm part of the exoskeleton. The side depends if the exoskeleton is used on the left or the right arm, yet the LEGO structure makes it possible to remove and attach it again easily to the opposite side.

The energy is also provided by batteries, located directly on the exoskeleton. The control unit needs 8 standard AA batteries which are stored in a battery rack. The rack is attached to the opposite side of the upper arm using velcro strips. (See the control unit and its placement in Fig. 2) It is important to mention that the motor power is highly dependent on the state of charge of the batteries, therefore in order to achieve the best results the batteries need to be fully charged.

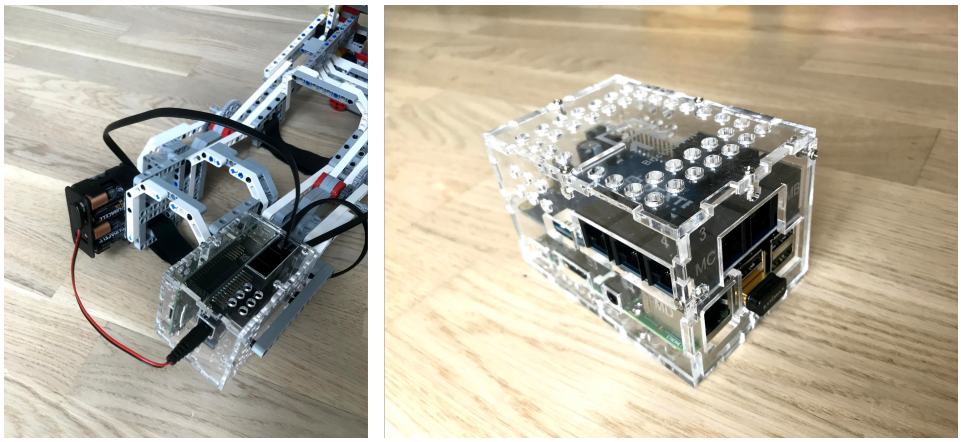


Figure 2: Left: Control unit and battery placement on the upper arm part of the exoskeleton, Right: Control unit (Raspberry Pi, BrickPi, USB WiFi adapter)

3.2 Sensors and actuators

In this version of the exoskeleton, the actuators and the sensors are identical. The LEGO Mindstorms EV3 Large Servo Motor is capable of moving and sending sensor data at the same time. This is made possible by the built-in tachometer which is capable of distinguishing one degree change in the motor rotation state. In total, there are three Large Servo Motors used in the exoskeleton: two at the elbow joint located on the two sides of the elbow and one built in under the arm in the wrist structure responsible for tracking and moving the circle around the wrist.

According to the official LEGO data sheet [7] the Large Servo Motor has the following attributes:

- 160-170 RPM
- Running torque of 20 N/cm (approximately 30 oz/in.)
- Stall torque of 40 N/cm (approximately 60 oz/in.)

In order to learn how precise the sensor information is, local logs were created on the Raspberry Pi. In these logs, the values of the sensors are kept alongside the corresponding timestamp with accuracy less than a millisecond. The python program which is operating the exoskeleton has a main loop where the sensor information are being read. In order to lower the Raspberry Pi CPU load and be aligned with the Unity program (see Section 3.3), 20 ms of system sleep was introduced in every loop. This of course reduces the accuracy of the read data but makes the wireless communication more seamless and syncs better with the Unity program. In Table 1 and Table 2 snippets of these logs can be seen. As it shows, without the 20 ms sleep the sensor value jumps are lower (more accurate) than with the sleep. On average the system created 20 logs per second with the 20 ms sleep and approximately 44 per second without. These values are also influenced by the speed of the the rotation when these logs were created.

value_elbow	timestamp
-153.5	00:13:31.843555
-155.5	00:13:31.865957
-157.0	00:13:31.883427
-158.5	00:13:31.900321
-160.0	00:13:31.917286
-161.0	00:13:31.934888
-162.5	00:13:31.951016
-164.0	00:13:31.967853
-165.0	00:13:31.987161
-166.0	00:13:32.003223
-167.0	00:13:32.022513
-169.0	00:13:32.050820
-170.0	00:13:32.066712
-170.5	00:13:32.082789
-172.0	00:13:32.114453
-173.0	00:13:32.135674

Table 1: Snippet of log values from EV3 Large Servo Motor without sleep in the main python loop. The column "value_elbow" shows the angle read from the motor

value_elbow	timestamp
-51.0	00:11:38.974136
-47.0	00:11:39.050367
-43.0	00:11:39.125527
-40.0	00:11:39.190281
-37.0	00:11:39.267482
-34.0	00:11:39.323987
-32.0	00:11:39.374381
-29.5	00:11:39.430927
-26.0	00:11:39.496651
-23.0	00:11:39.554433
-20.0	00:11:39.626116
-17.0	00:11:39.684907
-14.0	00:11:39.741422
-11.0	00:11:39.790880
-8.0	00:11:39.840402
-4.5	00:11:39.905485

Table 2: Snippet of log values from EV3 Large Servo Motor with 20 ms of sleep in the main python loop. The column "value_elbow" shows the angle read from the motor

3.3 Software

Dexter Industries, the creator of BrickPi3 provides a modified Raspbian operating system to run on the Raspberry Pi which contains the firmware needed to communicate with the EV3

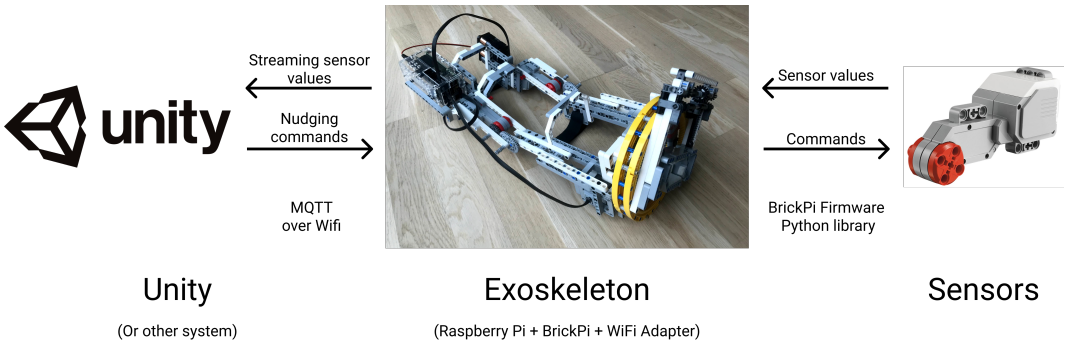


Figure 3: The software architecture of the system

sensors and motors. The Raspberry Pi is running a Python 3 program using the `brickpi3` library by BrickPi which provides API to the different EV3 components. [8] This program is responsible for gathering sensor data, creating logs, controlling actuators and communicating with other systems as indicated in Fig. 3.

The exoskeleton is capable of streaming its sensors values to another system and is ready to take commands to "nudge" the wearer to carry out given arm movements. To achieve this nudging torque the motors are moved 100 degree to the given direction on the wrist or the arm joint. These movements can be resisted by a healthy user (or by the gravity in case of the elbow joint), but they are strong enough to send signal to the user regarding the direction of the required movement. (A video of the exoskeleton working and initiating nudging movements can be seen in the appendices Section A)

3.4 Wireless communication

The exoskeleton was designed to communicate with other systems (Unity, data loggers...). In order to preserve the advantage of the exoskeleton being a "one-piece", independent device, wireless communication needed to be implemented. For this prototype the MQTT protocol is in use to communicate with other devices over WiFi.

3.4.1 MQTT

MQTT is a machine-to-machine (M2M)/"Internet of Things" connectivity protocol. It was designed as an extremely lightweight publish/subscribe messaging transport. [9] The exoskeleton is able to use the MQTT protocol through the `paho-mqtt` python library. [10]

As shown in Fig. 3, there is bi-directional communication taking place between the exoskeleton and the Unity program. The exoskeleton is sending its sensor values while listening to commands coming from the Unity program to initiate torques.

There are three different MQTT topics with messages being frequently transferred between the Unity program and the exoskeleton shown in Table 3.

Topic name	Description	Quality of Service	Message example
motor_value_elbow	Representing the mean angle of the two motors located at the elbow joint	QoS 0	-32.0,12:13:24.945779
motor_value_wrist	Representing the angle of the motor located at the wrist joint	QoS 0	45.0,12:13:25.745779
nudge	Commands for initiating nudging torques	QoS 1	"up"

Table 3: The three MQTT topics used by the system

The two sensor value topic's messages contain the value of the given sensor and the timestamp registered on the Raspberry Pi at the time of reading the data. The third topic is used for sending nudging commands to the exoskeleton and only contains the direction of the intended nudge as a string ("up" and "down" for elbow movements, "left" and "right" for wrist rotations).

The MQTT protocol has three different level of Quality of Service (QoS) [9] from which in this project only the first two are used. The QoS 0 is the fastest, but least secure level since it's only one directional, there is no proof that the message has been received. This level is used for "streaming" the values from the sensors to the Unity program (approximately 25 per second) because it is not substantial issue if a few messages get lost in the way, but on the other hand, speed is the key. The next QoS level (1) makes sure that the message has been received by waiting for a confirmation message from the subscriber. This level is used for the nudging message arriving from the Unity program to the exoskeleton. In this case it is important not to miss any message and since the frequency of these messages are not high (approximately one message per every 10 seconds) the speed doesn't need to be the fastest.

As described before the nudging movements are basically resulting in moving the actuating motors by 100 degree. In order not to influence the sensor values which are being read and used by the Unity program, the streaming of the sensor values are stopped for the short period of time when the nudging movement is happening. This method became relevant after testing the system since it was noticeable that after the nudging movement, the arm is passively moving the exoskeleton back to the starting position resulting a quick back and forth movement in the sensor values.

3.4.2 Latency

The latency between the exoskeleton and the Unity program was measured as the following: The timestamp of reading the sensor data on the control unit (Raspberry Pi) is sent to the Unity program as part of the MQTT message. After this the Unity program registers the timestamp when the given message is consumed.

Since the measurements are really precise (milliseconds) the time setting of the two devices are crucial and needed to be as synchronized as possible. To solve this issue the Unity program is sending the current time of the system as a timestamp on the program startup which is received by the exoskeleton control unit and used to change the time on the

Raspberry Pi immediately. After this, the local logs and the timestamps which are sent to the Unity program is based on this newly set system time.

After this initial step the Raspberry Pi's timestamp is included in every sensor message. The Unity program is creating a log entry once it processes these messages. The log entry contains the timestamp coming from the exoskeleton control unit (the time of reading value of the sensor), the Unity timestamp of processing the message and the angle value of the sensor. After analysing these logs, the mean latency has been calculated as **568 ms**. This number is based on 7343 log entries in total. The minimum recorded lag was 376 ms and the maximum 4 seconds 398 ms, which is clearly an outlier. From the 7343 measurements 459 were above 1 second.

Nonetheless, these lag values could have been influenced by another factor: There is some latency involved in sending the initial time from the Unity program, receiving it, then applying the time settings on the Raspberry Pi. It is possible that this adds a constant value to the calculated latency data.

Furthermore, based on the test runs, the speed is highly dependent on the router and its settings. The values on which the calculation was based were recorded using a private network provided by a TP-LINK AC1750 Wireless Dual Band Gigabit Router with firewall turned off.

4 Integration with Unity

As mentioned before, the exoskeleton is able to communicate with Unity by using the previously described MQTT protocol. The Unity program is using a Nuget package called M2MQTT which is an MQTT client available for all .Net platform. [11]

There won't be any detailed specific implementation here, but the current version of the exoskeleton allows three important things to be done with Unity :

- Calibration, this is a way for the software to know the range of movement of the current user. This is easily done by prompting the user to first extend his arm forward, palm facing down, and then bending his arm upward as much as possible, palm facing their shoulder. This way the software gets to record the two bound of the two rotation that are being recorded, and can interpolate from there the current position of the arm based on the streamed values.

- Recording. The streamed value of the rotations could be used for example to move an object on screen, where the wrist rotation could move the object horizontally while the elbow rotation could move the object vertically.

- Nudging. Using the motors to give some slight nudging could be used to prompt the user to avoid a virtual obstacle for example, or to extend his arm forward when a virtual interactive object is nearby.

5 Discussion and future work

On the exoskeleton itself, there are still some possible improvements available:

- Adding a second layer of gears to interact with the second layer of yellow inner gear. The amount of available gears for the project was limited, but it should be simple enough to add this second layer. This would reduce some of the gripping of the wrist rotation. Indeed, since these yellow inside gears are not in one piece, but actually four quarter of a circle connected together, when the transition from one quarter to the next comes in contact with the gears, the gears sometimes grips and the user either has to force the movement or cant do it at all. This has been greatly reduced in the final version presented here, but since the second layer of yellow inside gear is rotated by 45 degrees, having it interact with a second layer of gears connected to the motor would ensure that when a transition comes around a gear on one level, it is not here on the other one, and therefore there is less gripping.

- Reducing the size of the wrist locker. The current wrist locker is actually quite big, and masks a bit of the view of the user's hand. This could be a bigger problem if this exoskeleton is paired with a hand tracking software, therefore improving the visibility of the hand might be an important next step. Achieving this could be problematic, especially considering that the LEGO bricks don't really lend themselves to a robust locking mechanism: the current one took a lot of the prototyping time, and many previous versions were just letting the wrist go free as soon as there was a bit of resistance.

- Adjustable arm length and size. The current version assumes that the user doesn't have too big or little arms, and although it should be suitable for a good portion of the user base (one of the group members has long arms while the other has short ones), it is still probably too long for a child, or not large enough for a really muscular person. While the current version can easily, albeit with a bit of time and pieces, be adapted to most arm sizes and length, having a true one size fits all exoskeleton would probably require more than LEGO bricks, as adding this kind of moving parts would most likely weaken the whole design, and require stronger materials.

As described in the Section 3.4.2, the latency calculation has some flaws and the presented numbers are only best estimations. By working on the software and using better apparatus, more efficient and precise ways of logging and latency calculations could be developed which give a closer and more reliable latency value. Based on tests the calculated average latency (>500 ms) seems excessive, since the game was playable and no substantial lag was perceived. (See video in the appendices Section A)

The BrickPi Python library is lacking sophisticated methods as part of the API. One of these problems is the absence of async functionalities of motor commands. The program

currently cannot run while a motor is being moved. In this short time period the logging and the wireless connection are both put on hold.

As experienced during the tests the MQTT protocol is sometimes too slow for such high frequency data communication. A more specified or lower level protocol or library could be used in the future to ensure higher reliability and better speed.

As part of this project an older version of the Raspberry Pi was used (Model B+ revision 1.2). It is possible that by upgrading to a newer, state-of-the-art model, the system becomes quicker and more reliable.

In the future, muscle activity sensors can also be attached and used with the system through the GPIO pins of the Raspberry Pi. The muscle sensor is sending analog signal which the Raspberry Pi cannot process, yet a cheap analog-to-digital converter (ADC) can transform the signal to digital. By incorporating such sensors the patients could initiate the supporting movements only by flexing muscles as part of the start of the arm movement. A possible sensor is the MyoWare Muscle sensor. [12]

6 Appendices

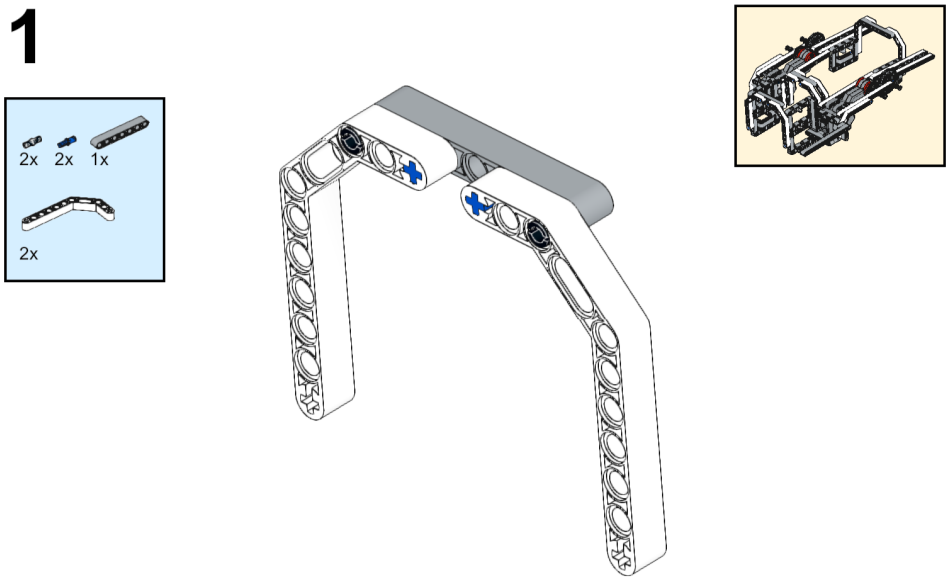
Appendix A Video of the exoskeleton being used with Unity

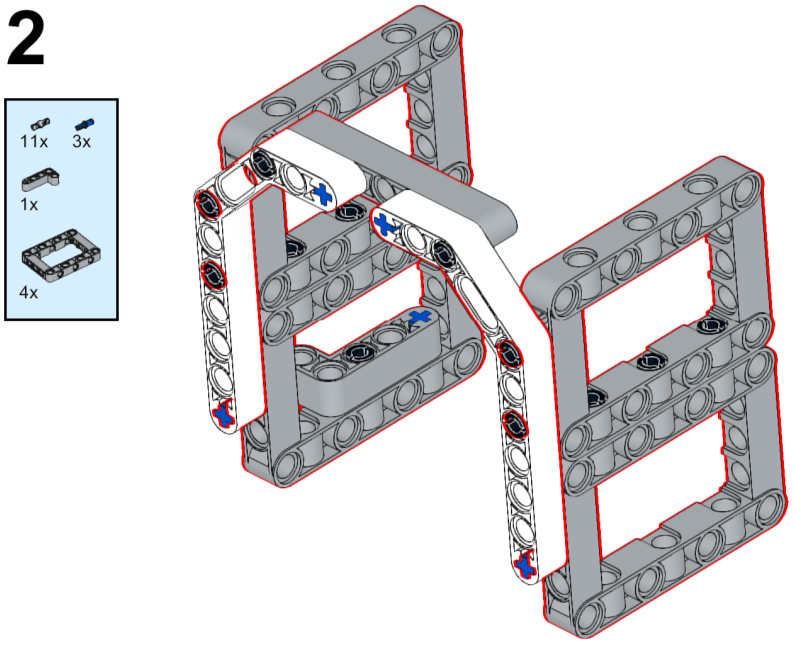
A video of the exoskeleton used connected to Unity can be seen in the following video. It also shows the nudging feature in form of small automated movements at the elbow and wrist join.
Video link

Appendix B Photos and videos of the development process

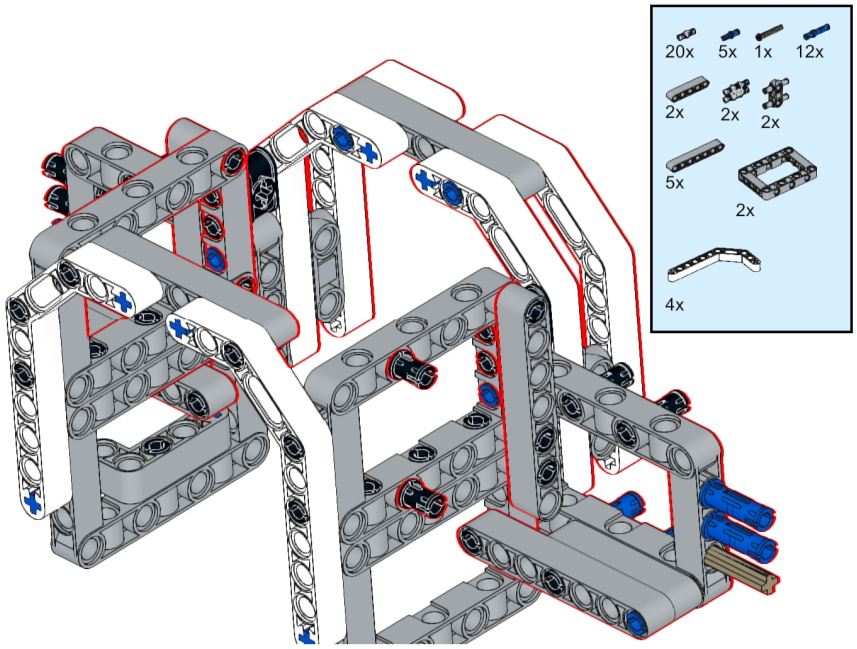
The photos and videos of the system through the iterations can be found here.

Appendix C Construction Manual

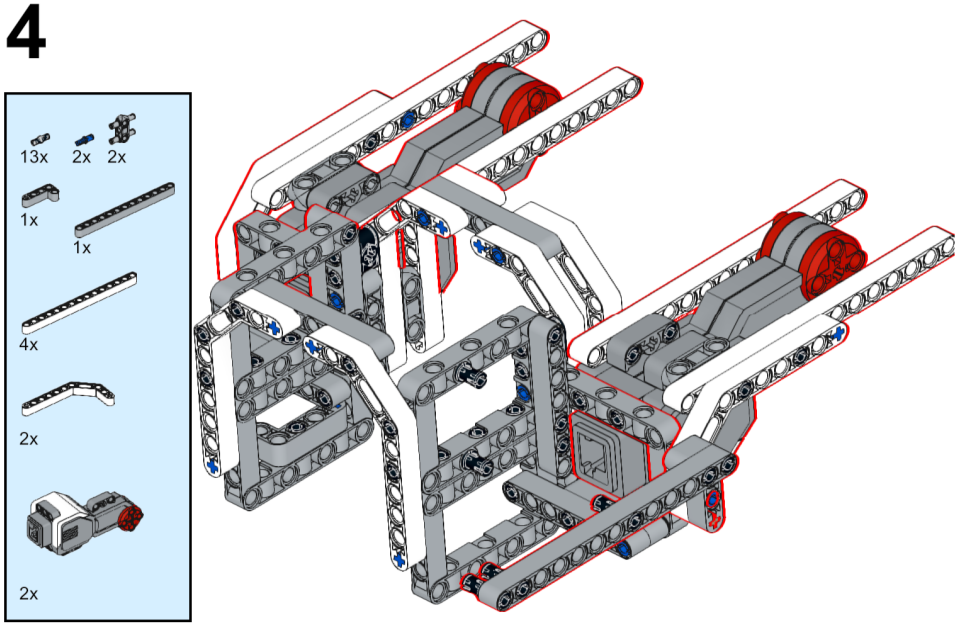




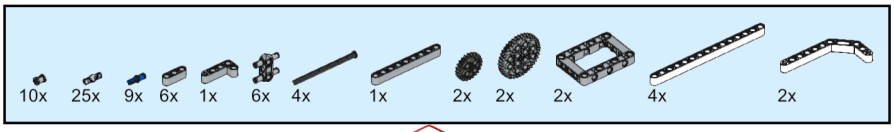
3

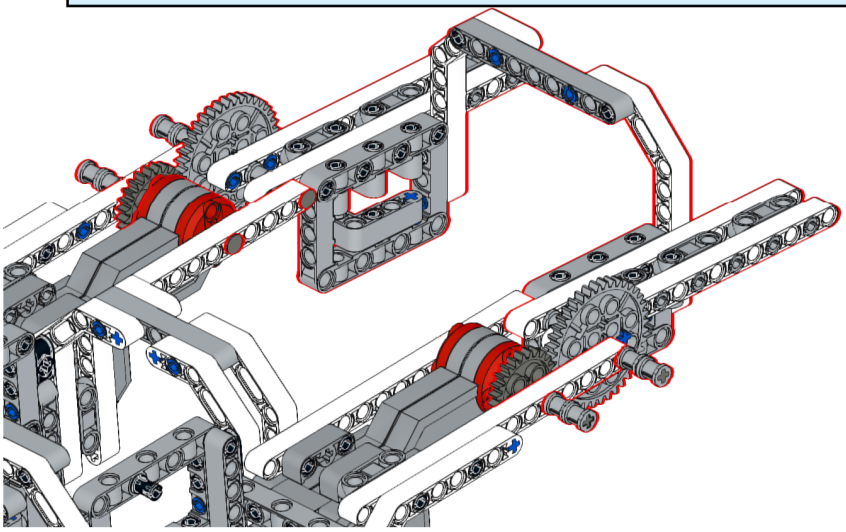


4

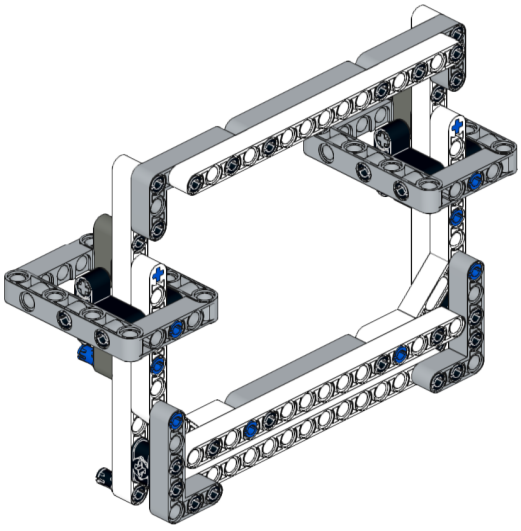


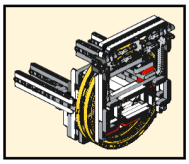
5

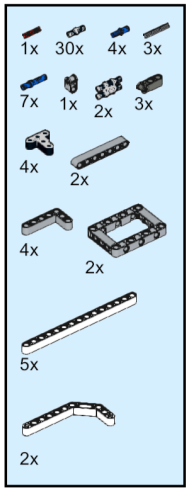

10x
25x
9x
6x
1x
6x
4x
1x
2x
2x
2x
4x
2x



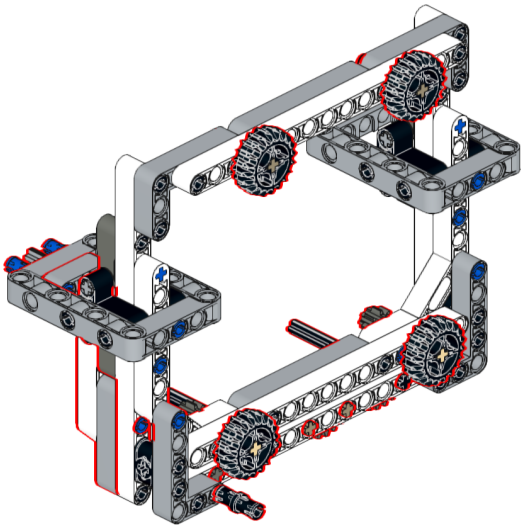
6

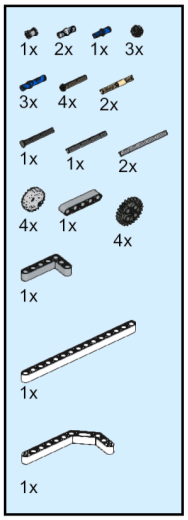




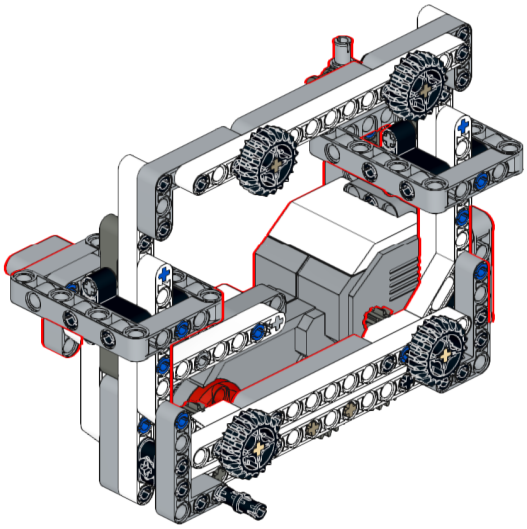

1x
30x
4x
3x
7x
1x
2x
3x
4x
2x
4x
2x
5x
2x

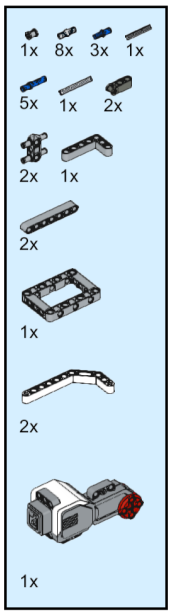
7



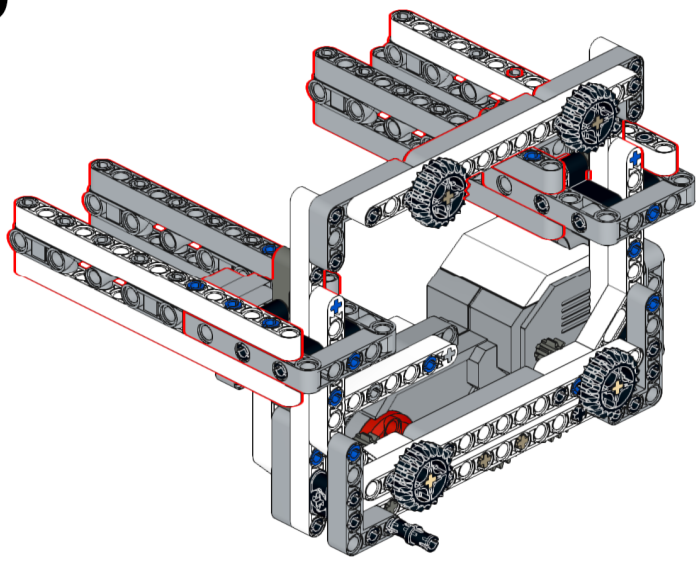

1x
2x
1x
3x
3x
4x
2x
1x
1x
2x
4x
1x
4x
1x
1x
1x

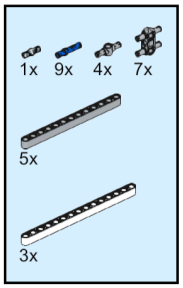
8



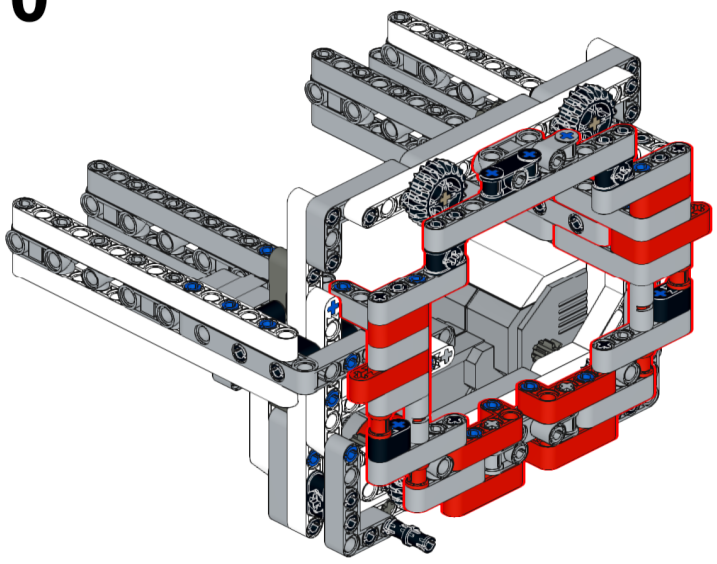

1x
8x
3x
1x
5x
1x
2x
2x
1x
2x
1x
2x
1x

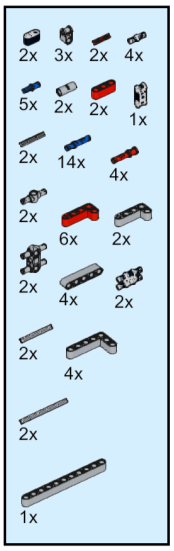
9



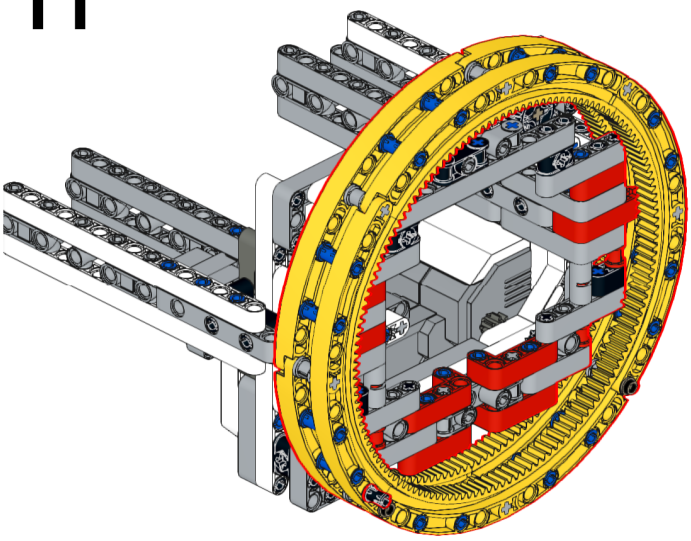

1x
9x
4x
7x
5x
3x

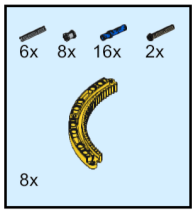
10



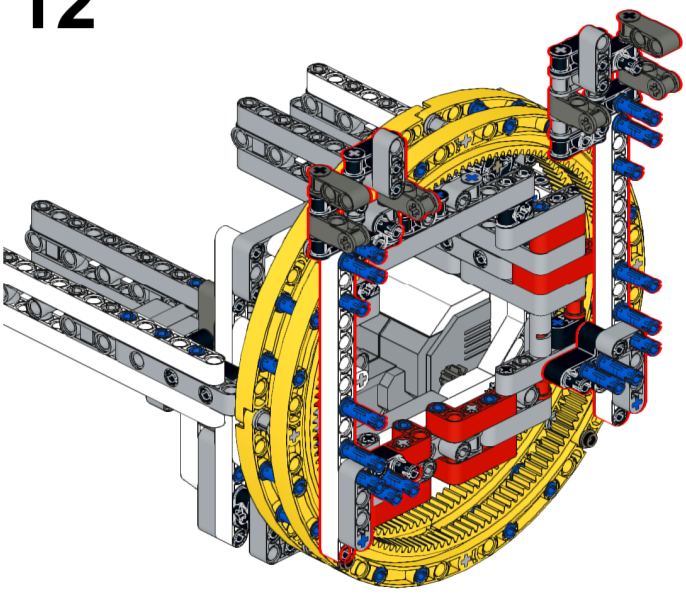

2x
3x
2x
4x
5x
2x
2x
1x
2x
14x
4x
2x
6x
2x
2x
4x
2x
2x
4x
2x
1x

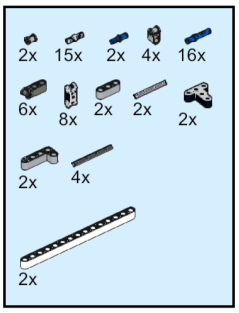
11



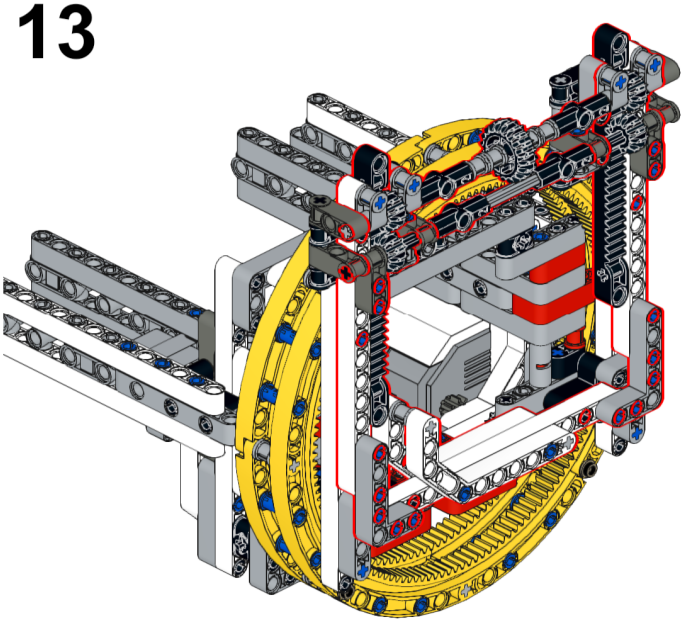


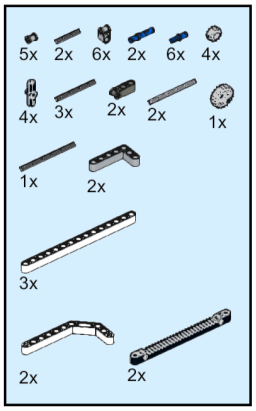
12



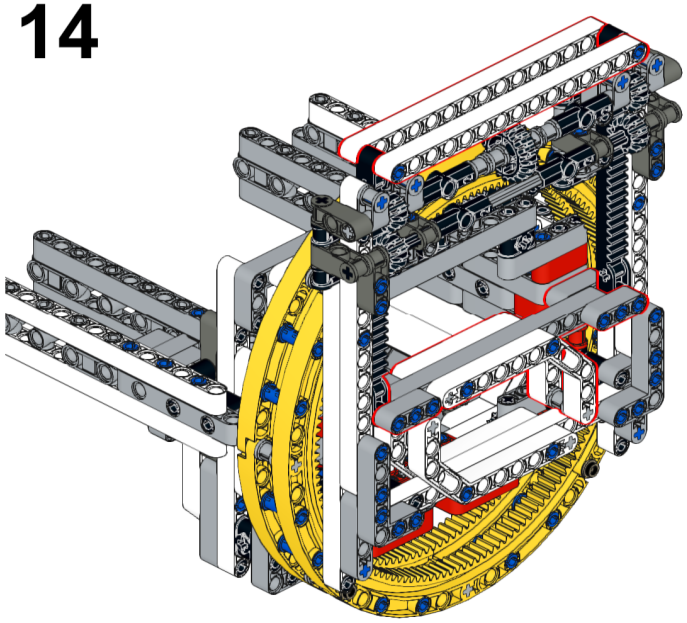


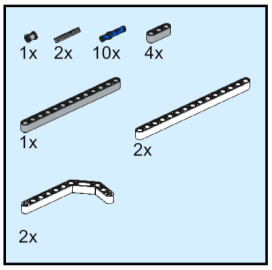
13



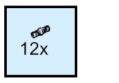


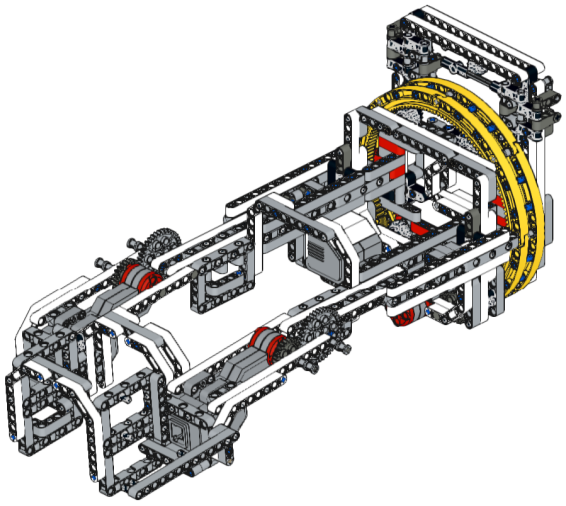
14





15





Appendix D Share of work

Abstract	Barnabas
Exoskeleton background and introduction	Barnabas
Physical device	Corentin
Electronics and software	Barnabas
Integration with Unity	Corentin
Discussion and future work	Corentin and Barnabas

References

[1] "Rehabilitation based on hybrid neuroprosthesis," ReHyb project. [Online]. Available: https://rehyb.eu/

[2] "Unity." [Online]. Available: https://unity.com/

[3] "EduExo - The robotic exoskeleton kit." [Online]. Available: https://www.eduexo.com/

[4] "BrickLink Studio." [Online]. Available: https://www.bricklink.com/v3/studio/download.page

[5] "BrickLink model." [Online]. Available: https://drive.google.com/drive/folders/1SBU3jC-dA5zfZrbNhBTJIHgbJg1PplQ8?usp=sharing

[6] "BrickPi." [Online]. Available: https://www.dexterindustries.com/brickpi/

[7] "LEGO EV3 Large servo motor." [Online]. Available: https://education.lego.com/en-us/products/lego-mindstorms-education-ev3-large-servo-motor/45502

[8] "BrickPi3 firmware and Python drivers." [Online]. Available: https://github.com/DexterInd/BrickPi3

[9] U. Hunkeler, H. L. Truong, and A. Stanford-Clark, "Mqtt-s — a publish/subscribe protocol for wireless sensor networks," in *2008 3rd International Conference on Communication Systems Software and Middleware and Workshops (COMSWARE '08)*, 2008, pp. 791–798.

[10] "paho-mqtt Python library." [Online]. Available: https://pypi.org/project/paho-mqtt/

[11] "M2MQTT Package." [Online]. Available: https://www.nuget.org/packages/M2Mqtt/

[12] "MyoWare Muscle sensor." [Online]. Available: http://www.advancertechnologies.com/p/myoware.html