TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY

PROJECT REPORT
REPLICATING A LEGO STRUCTURE WITH UR5 ROBOT USING AFFORDABLE MACHINE VISION SOLUTION

MEI-56306 Robotics Project Work

2018

Members

Julius Linkinen
Matiss Ziemulis
Tomi Sepponen

Andrei Lobov Supervisor

TABLE OF CONTENTS

FIGURES

1. INTRODUCTION

The goal of this project work was to use a designated UR5 robot arm and a Robotic 2-finger servo gripper to build a LEGO house or other structures based on user-prepared 3D model. The structure can consist of two different sized blocks, each in six different colors. The challenge was to implement a machine vision solution that the robot uses to recognize specific required bricks from the pick-up-platform to pick and place on the build-platform.

The primary goal was to implement above specifications. Additional features, for example a web interface, would be implemented if everything else were to be finished.

This report is going to go through the resources the group had at our disposal the project plan phase and our vision. Next the report goes in to more detail on what was implemented, how it was implemented and why it was done in a specific way. Before presenting the achievements and the conclusion, reader is given a user guide on how to use the project files to replicate and run our implementation and a brief rundown of multiple demonstrations we gave while working on the project.

2. RESOURCES

This section describes the resources that were used for the project. Some resources, for example the robot and the gripper were defined as part of the project. Using Raspberry Pi was suggested by groups supervisor, but the software side solution was decided in the group.

2.1 Universal Robots UR5 Collaborative Robot

The UR5 is lightweight, highly flexible, and collaborative industrial robot arm that lets user automate repetitive and dangerous tasks with payloads of up to 5 kg. The UR5 flexible robot is ideal to optimize low-weight collaborative processes, such as picking, placing, and testing. Promotional image of the robot arm is presented in figure 1.

Figure 1. *Universal Robots UR5 [1].*

With a working radius of up to 850 mm and 6 rotational joints, the UR5 collaborative robot is a great choice for this project. Our implementation uses UR controller version 3.4. However, all major versions 3 starting from 3.4 should work with our program.

2.2 Robotiq 2-Finger 85 Gripper

The Robotiq 2-Finger Gripper as seen in figure 2, has “two articulated fingers that each have two joints (two phalanxes per finger). The Gripper can engage up to five points of contact with an object (two on each of the phalanges plus the palm). The fingers are underactuated, meaning they have fewer motors than the total number of joints. This configuration allows the fingers to automatically adapt to the shape of the object they grip, and it also simplifies the control of the Gripper."   [2]

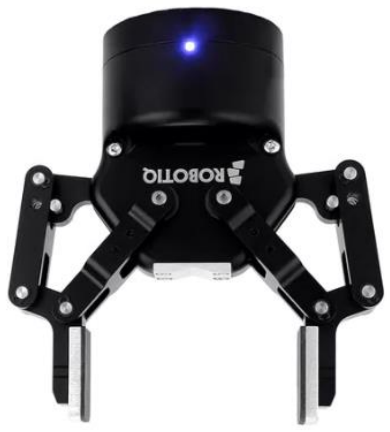


Figure 2. *Robotiq 2-Finger 85 Gripper [2].*

The Gripper comes with either 85 mm or 140 mm opening. The 85 mm option was used in the project.

2.3 Raspberry Pi 3 Model B and Camera Module

Raspberry Pi is a small single-board computer that is developed by the Raspberry Pi Foundation. Their main objective is to provide an affordable and easy to use computer to promote teaching of computer science in schools. The third-generation model B board was released in 2016 and its specifications include [3]:

- Quad Core 1.2GHz Broadcom BCM2837 64bit CPU
- 1GB RAM
- BCM43438 wireless LAN and Bluetooth Low Energy (BLE) on board
- 100 Base Ethernet
- 40-pin extended GPIO
- 4 USB 2 ports
- 4 Pole stereo output and composite video port
- Full size HDMI
- CSI camera port for connecting a Raspberry Pi camera
- DSI display port for connecting a Raspberry Pi touchscreen display
- Micro SD port for loading your operating system and storing data
- Upgraded switched Micro USB power source up to 2.5A.

For machine vision we used the Raspberry Pi Camera Module V2 that is designed to be used together with the board. The Camera Module is equipped with Sony IMX219 8-megapixel sensor that can be used to take high-definition video, as well as still photographs [3]. It's easy and simple to use for beginners and is easy to install. Both the Raspberry Pi and the Camera Module installed can be seen in the figure 3.

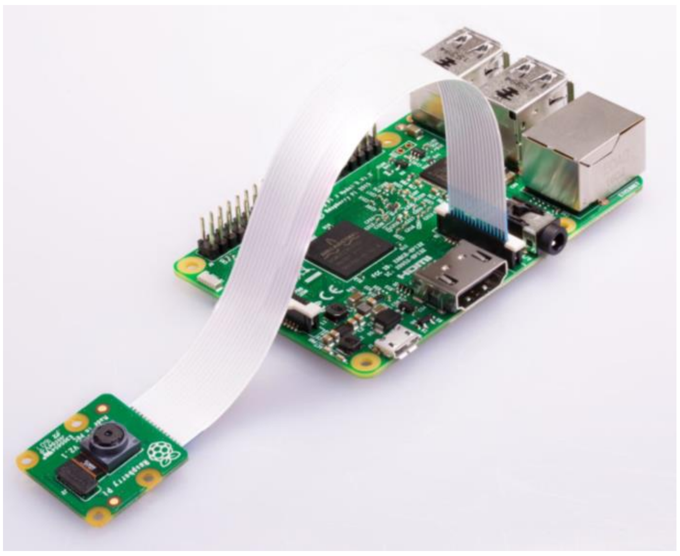


Figure 3. *Camera module attached to a Raspberry Pi 3 [3].*

Other necessary peripherals include the power supply and a charging cable, and the microSD card. For the initial setup a keyboard and a HDMI cable are also required. We used the official Raspberry Pi power supply and a 16GB MicroSD to which Raspbian Stretch operating system was installed. We also ordered a casing for the Pi, but it was not suitable for our needs and we ended up 3D printing our own case which is shown later on.

These components were the only ones that had to be ordered to complete our project, and the total costs were around 100 euros, which is very affordable considering rest of the equipment.

2.4 DUPLO and Lego Digital Designer

LEGO bricks, more precisely DUPLO bricks are used to replicate the model. Two different sizes (2x2 and 2x4) in six different colors (yellow, red, green, blue, black and white) were available to us. The key difference between normal LEGO and DUPLO bricks is that latter are two times the size of the former.

The build platform where the model is built on is a basic green DUPLO baseplate with studs for placing the bricks. The size of the plate does not matter (as long as the model can fit in it and the robot can reach it) as it is calibrated at the start of the build process. The supported bricks and the baseplate can be seen in the figure 4.

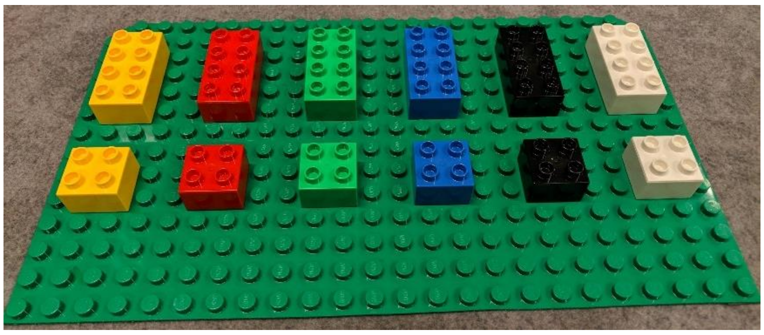


Figure 4. *Available bricks and colors on the baseplate.*

There are multiple CAD software options that enable creating LEGO models, such as Studio 2.0 and Lego Digital Designer (LDD). Any software that has an ability to export the model as and LDraw file can be utilized to run this implementation. We used LDD as it is an official tool and it is easy to use. The user interface can be seen in figure 5.

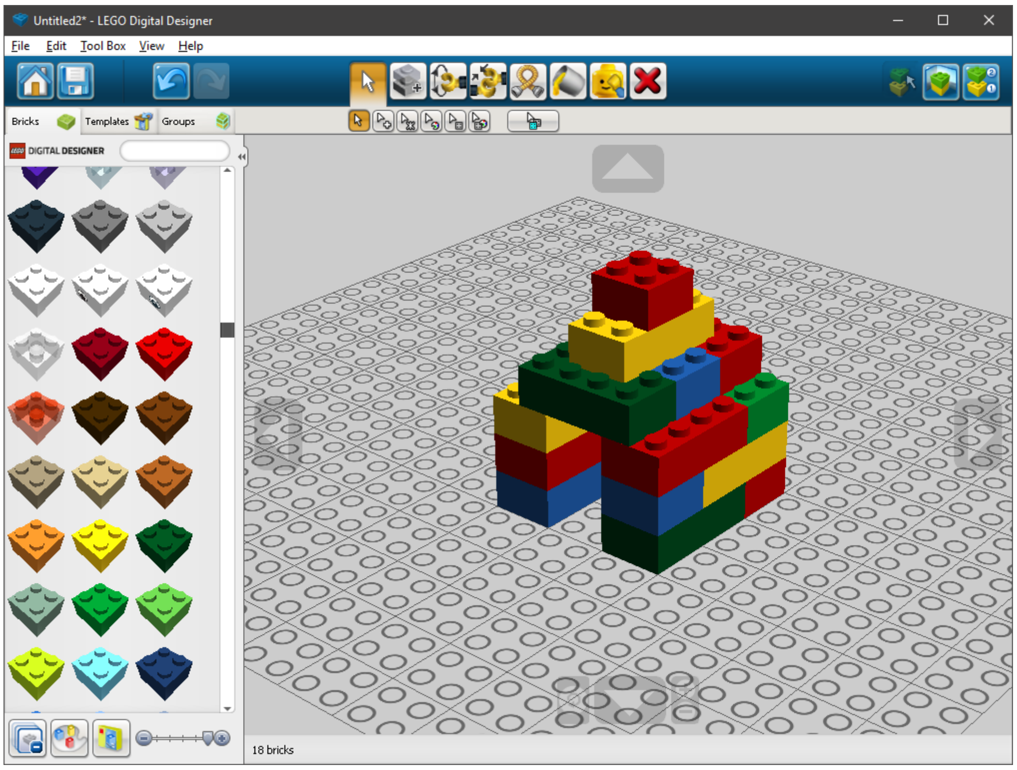


Figure 5. *Lego Digital Designer*.

2.5 Human Resources

The project group consists of three MSc students with associate professor Andrei Lobov acting as the supervisor of the project. The group has a strong background in mechanical engineering with varying skills in programming. The decision to use Python as our language of choice came mainly down to the fact that it is what most were familiar with, it is easy to learn and has wide range of available open source libraries for scientific computation and image processing.

3. PROJECT PLAN AND MANAGEMENT

The objective of the third chapter is to present our project planning process and how everything was managed. This includes the initial planning phase, our vision, evolution of project plan, risk analysis and how everything was communicated.

3.1 Initial Planning Phase

The initial planning phase took place during the very first month from the start of the project work course. During this phase arranged the first meetings with the supervisor on and carried out the inspection of the build platform. This included completing the Universal Robot Academy web course to familiarize ourselves with the UR5 robot.

One group had attempted the same project work a year earlier, and we requested access to their files and project report. After studying their implementation, we came into few conclusions:

- They had accuracy problems when using ROS (Robot Operating System). We opted to take advantage of the URScript API directly and such circumvent any kinematics and calibration inaccuracies ROS might inflict. ROS uses its own kinematics solver and not the one made (also individually calibrated at the factory) for UR5.
- They had a single pick up position, where they fed each brick manually in correct order. We must avoid this.
- Good calibration procedure must be implemented.
- We must resolve problems with re-orientating the bricks.

We created our own vision on how the system should work based on the things stated above. This vision is better presented in chapter 3.2 Initial vision and structure

3.2 Initial Vision and Structure

Our vision is that the robot can replicate the 3D model without any assistance, so user only needs to provide the blocks and the model. System on itself would analyze the model, read the coordinates, look for the necessary bricks in pick-up area, determine the rotation, pick up the block and place it in previously determined place. We would need multiple initial start-up processes, like build platform calibration, camera calibration, that would acquire more attention from the user. We didn’t implement these things in our structure, because they are part of a setup and only request from user to move the robot to certain positions and place blocks in defined points. Deployment diagram (figure 6.) shows what hardware components ("nodes") we need to execute the plan. In the initial stage we got five main modules, UI/monitor, build planner, model designer, robot controller and the machine vision system.

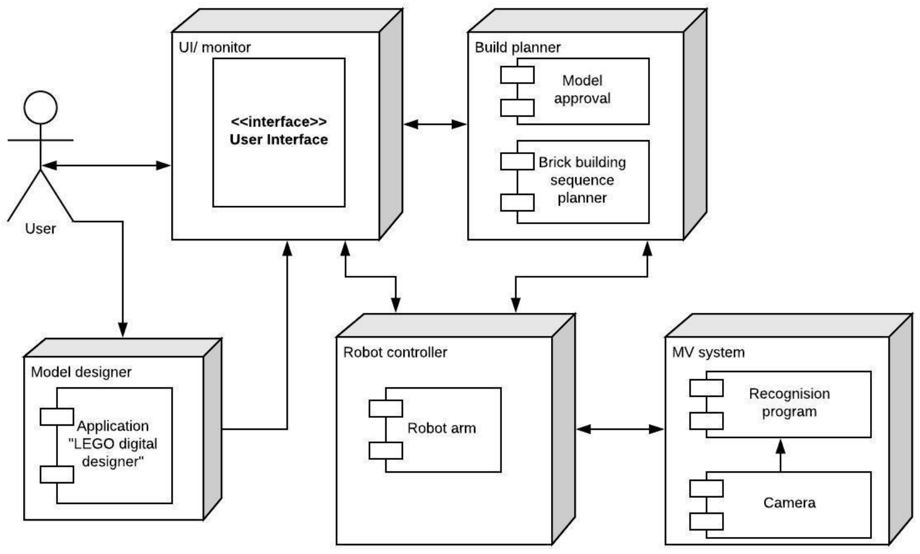


Figure 6. *Deployment diagram.*

In the Sequence diagram (figure 7), we looked at possible command and action order after the setup, and in this we separated robot controller and the robot itself, for more detailed understanding. But of course, while we were working on the project some things changed as we go, but we tried sticking to the initial idea, and kept it as a project development guidance.

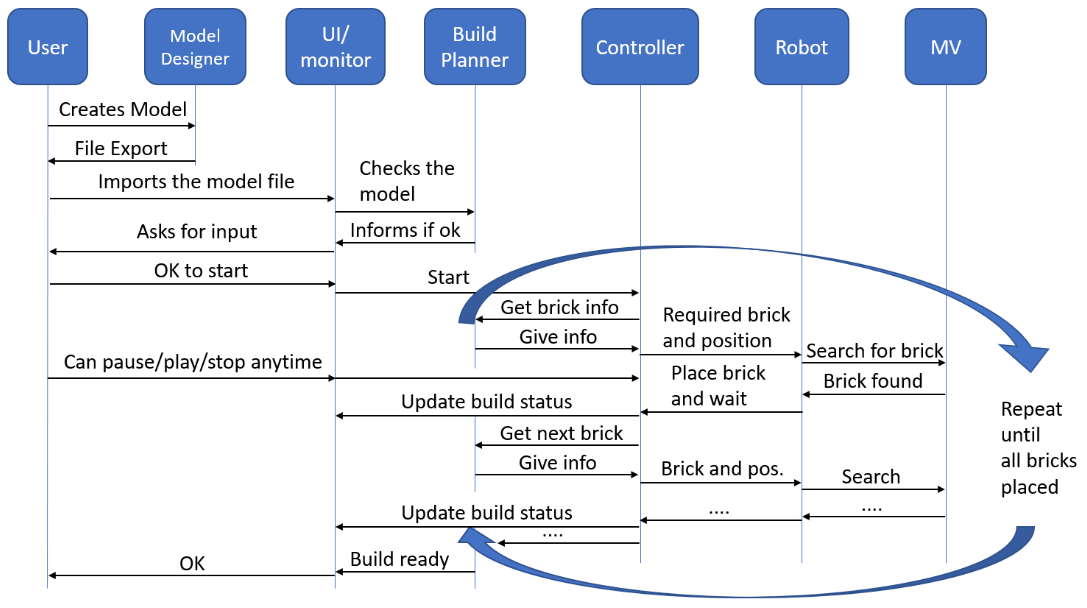
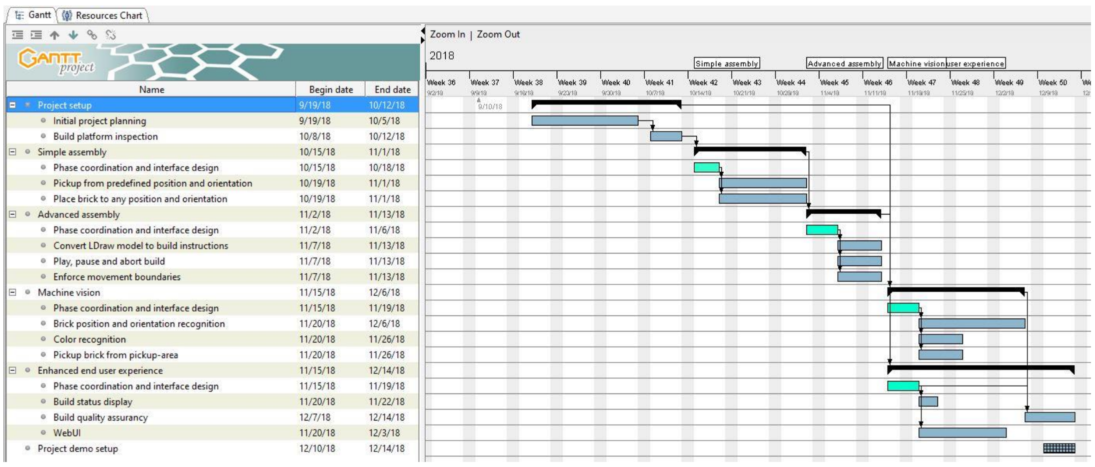


Figure 7. *Sequence diagram.*

3.3 Evolution of the Project Plan

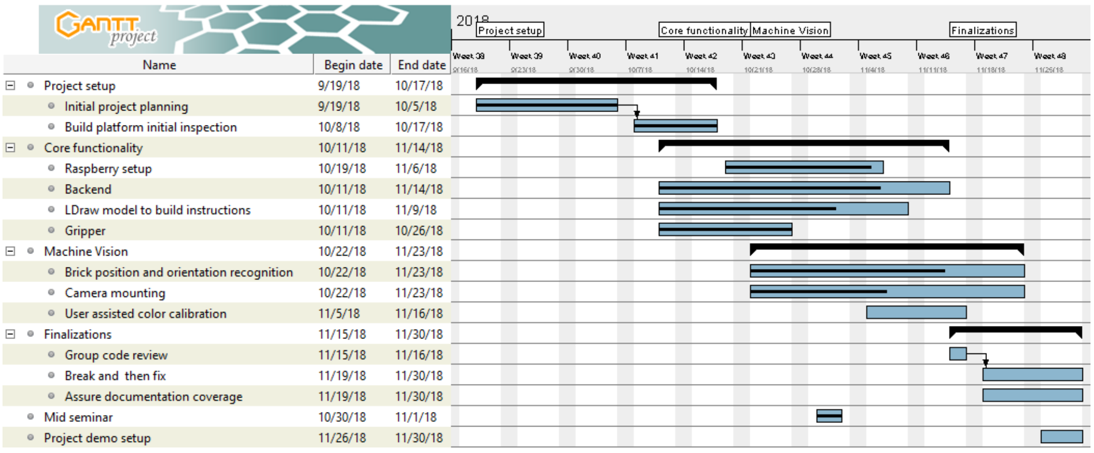
The evolution of the plan began by defining specific modules and features for the system. The initial plan was to develop multiple features at the same time and integrate everything at the end. The Gantt-chart for this so-called integration-based model can be seen in figure 8.



Name	Begin date	End date
Project setup	9/19/18	10/12/18
Initial project planning	9/19/18	10/5/18
Build platform inspection	10/8/18	10/12/18
Simple assembly	10/15/18	11/1/18
Phase coordination and interface design	10/15/18	10/18/18
Pickup from predefined position and orientation	10/19/18	11/1/18
Place brick to any position and orientation	10/19/18	11/1/18
Advanced assembly	11/2/18	11/13/18
Phase coordination and interface design	11/2/18	11/6/18
Convert LDraw model to build instructions	11/7/18	11/13/18
Play, pause and abort build	11/7/18	11/13/18
Enforce movement boundaries	11/7/18	11/13/18
Machine vision	11/15/18	12/6/18
Phase coordination and interface design	11/15/18	11/19/18
Brick position and orientation recognition	11/20/18	12/6/18
Color recognition	11/20/18	11/26/18
Pickup brick from pickup-area	11/20/18	11/26/18
Enhanced end user experience	11/15/18	12/14/18
Phase coordination and interface design	11/15/18	11/19/18
Build status display	11/20/18	11/22/18
Build quality assurancy	12/7/18	12/14/18
WebUI	11/20/18	12/3/18
Project demo setup	12/10/18	12/14/18

Figure 8. *Original project plan.*

After few discussions with our supervisor we came in to a conclusion that this integration-based model involved too many risks, one of which we referred as “integration hell” where we would be unable to successfully integrate every part in to a cohesive system. That would have left us with multiple separate sub-systems and nothing to demonstrate. A new model for our plan was introduced in form of phase centric design. In this model we have multiple phases with their own goals. After completing each phase, we would have something concrete to demonstrate and every other phase after that would add to the existing system. The Gantt-chart for this plan can be seen in figure 9.



Name	Begin date	End date
Project setup	9/19/18	10/17/18
Initial project planning	9/19/18	10/5/18
Build platform initial inspection	10/8/18	10/17/18
Core functionality	10/11/18	11/14/18
Raspberry setup	10/19/18	11/6/18
Backend	10/11/18	11/14/18
LDraw model to build instructions	10/11/18	11/9/18
Gripper	10/11/18	10/26/18
Machine Vision	10/22/18	11/23/18
Brick position and orientation recognition	10/22/18	11/23/18
Camera mounting	10/22/18	11/23/18
User assisted color calibration	11/5/18	11/16/18
Finalizations	11/15/18	11/30/18
Group code review	11/15/18	11/16/18
Break and then fix	11/19/18	11/30/18
Assure documentation coverage	11/19/18	11/30/18
Mid seminar	10/30/18	11/1/18
Project demo setup	11/26/18	11/30/18

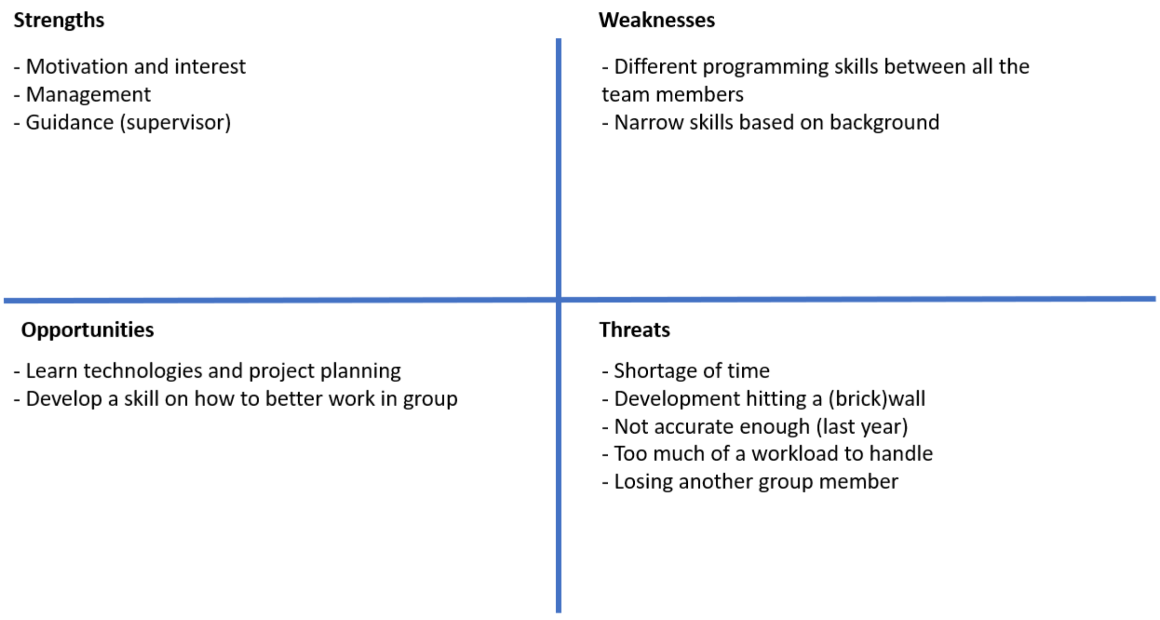
Figure 9. *Evolved Gantt-chart.*

Both Gantt-charts can be more easily examined in the second attachment.

3.4 Risk Analysis

In the first stage of our project, we established a SWOT analysis chart (figure 10), where we evaluated our strengths, weaknesses of our team, opportunities to learn and threats that could delay advancement. Our team’s biggest weakness is based on our levels of skill that would be necessary to finish the project. As this project contains both programming and a little bit of mechanical engineering, mechanical part did not concern us because all of us have some background in this field. Biggest difference was in programming skills, Tomi was confident about his skillset, Julius had experience in programming, but Matiss did not have any real practice.

There were several threats regarding project management. Shortage of time, that’s one of the reasons why we chose to switch our project plan to phase centric. So that at any point we would have something to hand in. Another one was concern of workload too much for us to finish the project, and the problem we encountered in the beginning, that one of our former team member left us in the development stage.



Strengths	**Weaknesses**
- Motivation and interest - Management - Guidance (supervisor)	- Different programming skills between all the team members - Narrow skills based on background
Opportunities	**Threats**
- Learn technologies and project planning - Develop a skill on how to better work in group	- Shortage of time - Development hitting a (brick)wall - Not accurate enough (last year) - Too much of a workload to handle - Losing another group member

Figure 10. *SWOT analysis.*

When we thought about project itself, two things came to our minds, one would be that our project encounters such a problem, that we could not resolve, that would make us start the project from scratch, to avoid the problem. Second, based on the last years teams report, that there could be problem with accuracy and inconsistency issues with robot, gripper and the gripper-fingers.

3.5 Communication and Management

The student group chose to use WhatsApp as the main channel for communication and discussion about the individual progress. The communication between the group and the supervisor was managed using emails and weekly meetings from the very start of the project course until the final week of working.

File sharing was managed using Microsoft OneDrive where every file was uploaded for easy access. Every line of code was uploaded to Gitlab repository for version control. In addition to weekly meetings the files created during each week were uploaded to a weekly update folder for the supervisor to keep track of the progress.

4. IMPLEMENTATION

In this chapter we will describe our final implementation and present the evolution of some of the features such as 3D designed components. The goal is also to explain some of our logic behind the design choices and other possible details.

The following describes in very short what the system is composed of and how they are connected. Same can be seen in figure 11.

1. User creates 3D model in LDD and imports it as LDraw file-format.
2. Host PC runs the code and does all the processing.
3. Raspberry Pi takes pictures for the machine vision and sends them to host PC.
4. Robot controller takes the command scripts sent by host PC and calculates the inverse and forward kinematics for the robot arm. Executes movement.
5. The host PC and the Pi are connected wirelessly through a router while the router is connected by ethernet-cable to the socket connection in robot controller.

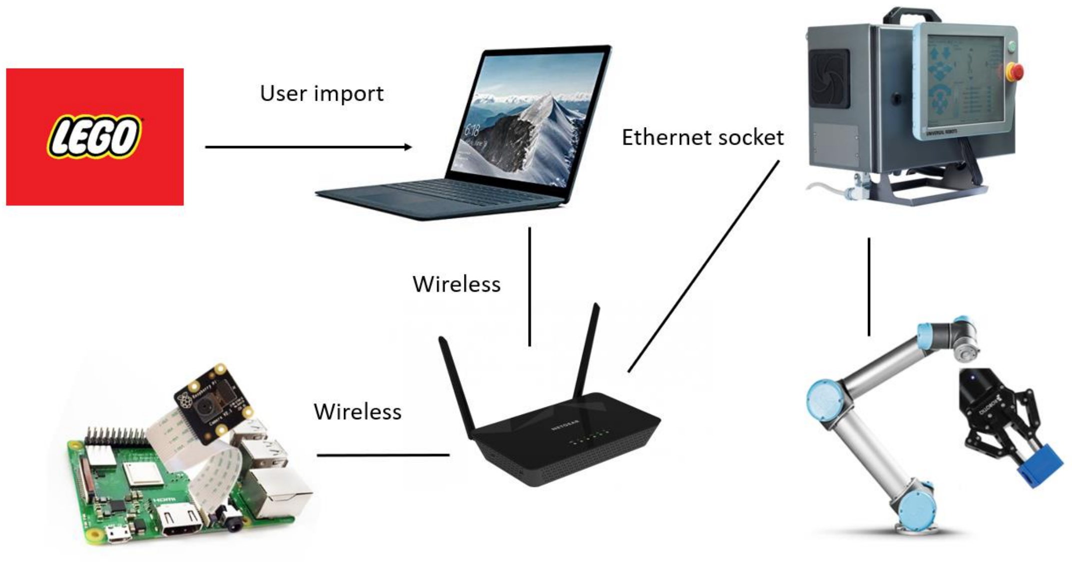


Figure 11. *System and connections.*

4.1 3D Design and Mounting Solutions

Our project required manufacturing of two components, gripper fingers and a mount for attaching Raspberry Pi and camera module to the UR5 robot. Throughout the process we considered different fabrication methods but settled down for 3D printing. Because the final product is durable enough for our application and prototyping is easy trough fast production method.

4.1.1 Gripper Fingers

While modelling and testing gripper fingers we prototyped 4 different models. There were three different ideas on how to grasp the Lego block.

First prototype was a simple idea for picking up the blocks, it seemed like a good solution because the idea was that it would align blocks as needed and the precision of picking up would not be that crucial. But was scratched because the gripper cannot apply the force in outwards direction, as we found out in our first try moving the blocks.

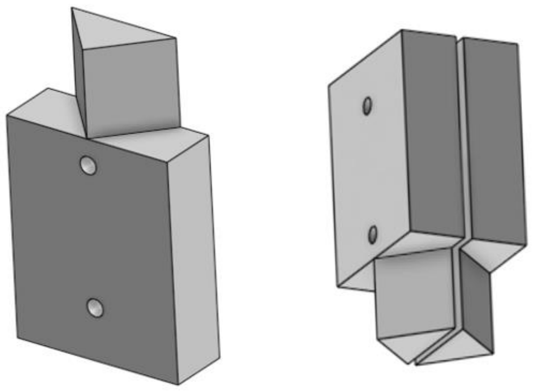


Figure 12. *First finger prototype.*

Second prototype lasted quite a while in our system and we used it in the development and the testing of the system. This utilizes holes in the middle of the studs, but it requires precision of picking up the blocks ± 1 mm, which in the end turned out as one of the reasons to develop new designs. There were two major flaws in this prototype, one was the lack of accuracy in our system and the other was gripper fingers tilting right at the moment the force was applied, so the blocks got pushed out.

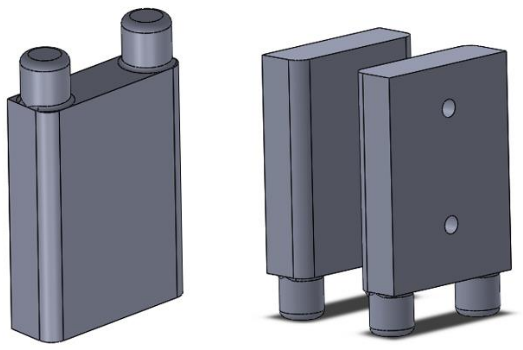


Figure 13. *Second finger prototype.*

After troubleshooting of second prototype, we developed two new versions, one was upgraded version of existing fingers, other was different principle of grasping the blocks.

In upgraded version we changed the diameter and form of fingers, added a little slope to compensate for the play in the gripper itself.

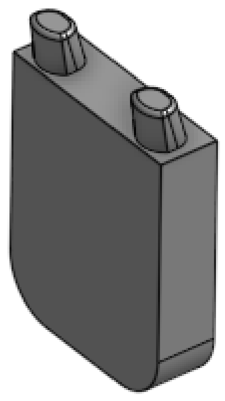


Figure 14. *Alterations on the previous design.*

New version of the gripper, and the final version, we are gripping the Lego bricks from outsides of the studs, that are on top. This allows some inaccuracies of the system; the gripper fingers help to align the blocks for easier pick up and provides enough grip.

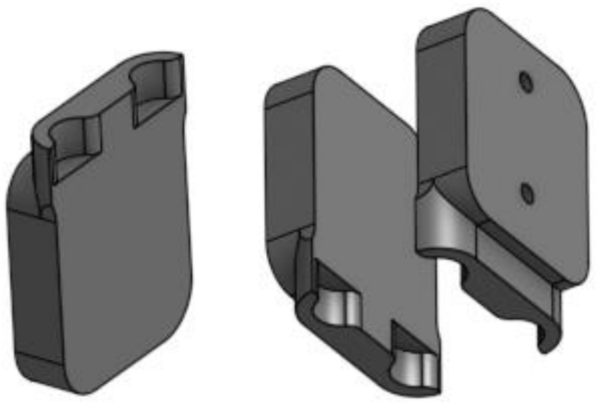


Figure 15. *Final version for the gripper fingers.*

4.1.2 Raspberry Pi Mount and Casing

Main idea behind of the first mount was so that it would be easy to attach and remove from system. Main plan was to use existing geometry to make sure that placement stays the same with each implementation. But as we were missing some pieces of our system we just needed to adapt and use household items (shoelaces, tape, zip ties) to secure it in place, and that made our system unreliable.

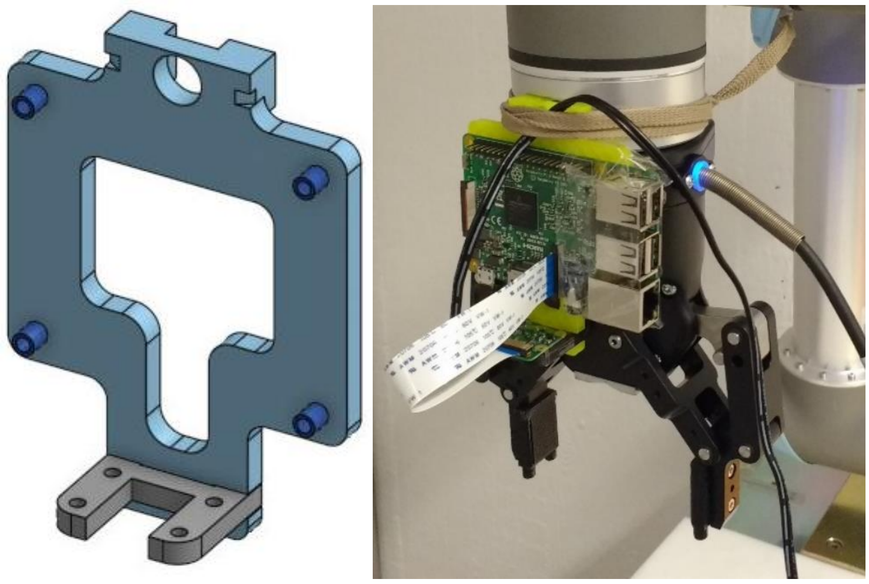


Figure 16. *First mount prototype.*

To make sure that our system does not move after it was installed, next prototype was secured by placing it between robot and gripper. Which made our system dependable, next problem was the tolerances between fitted parts, as there was just a little bit too much of a play between those, it allowed our camera module to move. That did not leave a good impact on machine vision. We added casing for the Pi and secured it in place with screws, so it performed better and looked more serious.

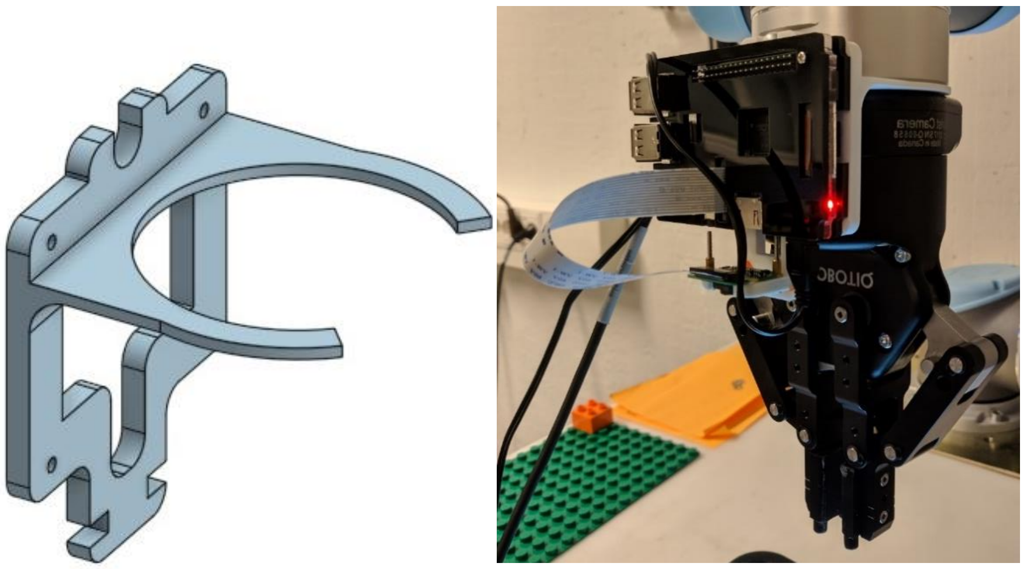


Figure 17. *Second mount prototype with temporary casing.*

In order to insert correctly the gripper, we needed to make some parts of the mount thinner, so it would fit better, and we added a rim in order to strengthen the model. Some other changes we did was lowering the tolerances of fitted pieces. This might be final model for mount, while we have not found out any problems with this. On addition about the things we might do would be gluing both fitted parts in order to avoid any unnecessary movement.

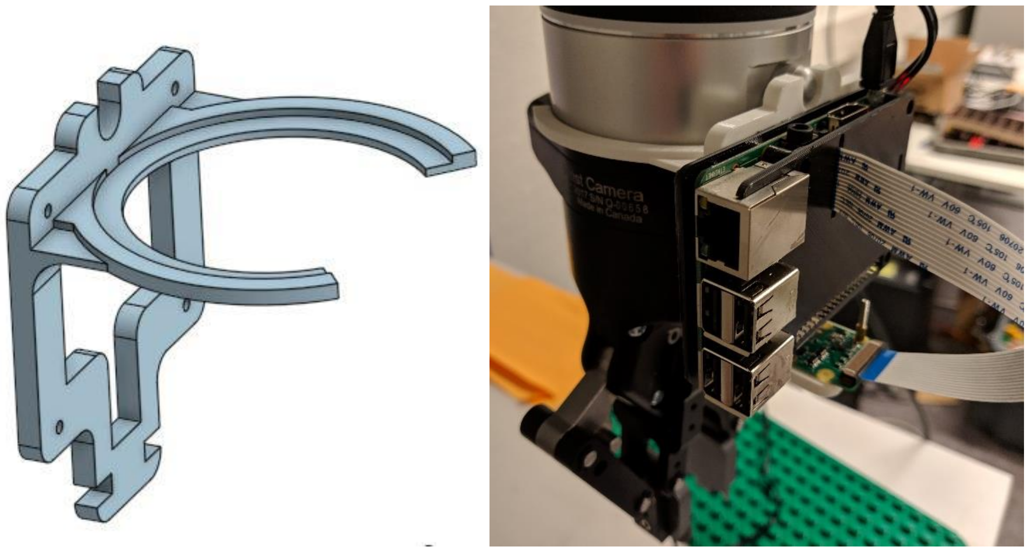


Figure 18. *Final mounting solution with casing.*

Another manufactured part was a casing for Raspberry Pi, which was necessary for better protection and better overall look of the system. We acquired the 3D models from Thingiverse [4] as there are already multiple good models and we saw no value in creating our own take on the case. With small modifications the final parts came out really good in terms of quality and was a great fit on our Raspberry Pi.

4.2 Creating the Build Plan

Our implementation accepts LDraw (.ldr) file format imports and it was chosen as it includes the information for individual bricks each on a separate row of text. It also has no excessive information which makes handling the information much easier. Each row contains the following information:

```
1 <color> x y z a b c d e f g h i <file>
```

- The first value implies the line type. Line type 1 means that it is a brick.
- **<color>** is a number representing the color of the part. As there are multiple color-code options with slight difference (e.g. light green and dark green), our code translates all of them to match a single color.
- **x y z** are the coordinates. It is important to note that this matches the coordinate system used in the software, where x implies the height and y and z create the plane where the bricks are placed on. It uses its own LDraw units (LDU) where 1 LDU = 0.4 mm. In case of DUPLO this also needs to be doubled.

- **a b c d e f g h i** describes a top left 3x3 matrix of a standard 4x4 homogeneous transformation matrix. This represents the rotation and scaling of the part. This is used to recognize if the gripper needs to be rotated to parallel either x- or y-axis.
- **<file>** is the filename of the sub-file referenced and must be a valid LDraw file-name, for example 3001.dat is a 2x4 brick.
  [5]

We take the file and check that it has only supported bricks and process it so that in the end we have each brick in our build plan ordered by the layer they are placed on. The output that is used later for the robot looks like this:

```
<layer> x y <orientation> <color> <size>
```

It is important to note that now the x and y match the coordinate system used on the build platform. The layer is simply calculated by the height and the size is either 2x2 or 2x4.

4.3 DUPLO Localization from Images

In order to pick up DUPLOs their location and orientation in the world must be determined. Our machine vision system looks for **rectangles** of certain colors and extracts 2D poses (x, y and angle) for their center points with respect to TCP. These poses can then be transformed to world coordinates. OpenCV 3 is used for image processing.

Discovering rectangles of certain color is based on removing everything but the target color from the image. Target color is defined as a range of HSV (hue, saturation, value) values. Everything in the range is set to white and rest to black (figure 19 left). The result is a binary image illustrated in. From there on the edges of the white areas are extracted as contours. For each fully closed contour a minimum bounding area rectangle is fitted (figure 19 right). Now that all areas are enclosed by a rectangle the program can sort and filter the results to find the best match. Best match is the one with dimensions closest to the one program is looking for. If there are no matches a new capture is taken, and this is repeated until a match is found.

The detection process:

1. Take a still image.
2. Filter out all but the target HSV range.
3. Create binary image (figure 19 left).
4. Find fully closed contours.
5. Fit minimum area rectangles to contours (figure 19 right).

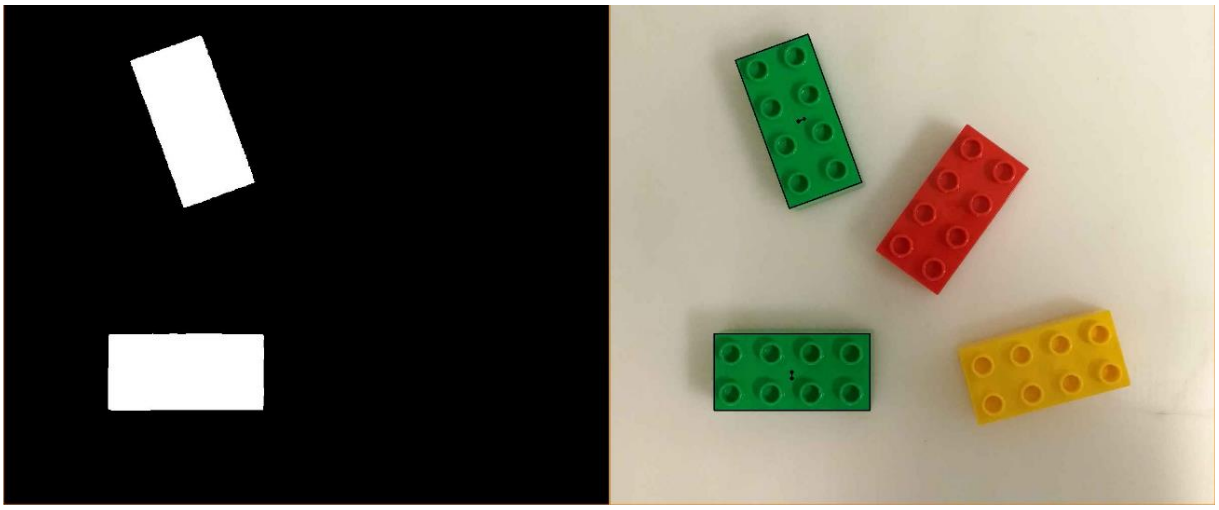


Figure 19. *Mask image and bounding rectangles drawn for green DUPLOs.*

Rectangle’s pose in image is represented as three values: coordinates x and y, and angle theta w.r.t image’s x-axis. Rectangle’s axis is set so that x-axis aligns with the short side of the rectangle. For squares axis selection is indifferent.

Before returning a match results its pose w.r.t image’s origin is converted to a pose w.r.t TCP. This, more useful pose, can be used to move TCP above the match. TCP’s location within image is determined during camera calibration that can accommodate to minor errors in camera mounting whereas using fixed offset values would not. Calibration is discussed in a later chapter.

Certain assumptions are made for the imaging environment. The camera’s plane is assumed to be parallel to the table in which DUPLOs are laid on. All objects of target color are assumed to be DUPLOs that are in a position from which they can be picked up (e.g. not upside down). This is due to the fact the system doesn’t differentiate between objects using anything else but the color information.

For accurate localization the imaging process is repeated until following correction movements would be smaller than a pre-defined value. This circumvents the problem of bad estimates that are worse the farther away the objects are from the image center.

For consistent captures the camera’s various automatic modes are disabled. By default, the camera does auto white balancing and varies exposure time between captures. These were turned off and fixed values were provided for white balancing. Especially auto white balancing might cause differences in background to change the tint of a capture enough to lead to a detection failure.

4.4 Placing the Bricks

After we had implemented the building feature, we ran in to recurring accuracy issues. After multiple tests we concluded that our build-platform moves ever so slightly after each brick placed. The green build-platform was attached to the base plate with double sided tape but that did not fix our problem. This is when we realized that the base plate itself was not attached to the desk on our robot cell and after removing the excessive plate the accuracy issues were mostly gone.

The robot always automatically tries to use the shortest path available, even when rotating the gripper. When looking for the correct brick to pick up this would ultimately lead up to a problem where the joint limits would cause the program to crash. This problem was fixed by forcing the robot to undo the rotation after picking up the brick and not to use the shortest path.

Initially 2x4 bricks had caused problems with machine vision as the system could not reliably decide the orientation of the brick and that would lead the robot to fail placing the brick correctly. This was fixed so that the machine vision always recognizes the shorter edge of the rectangular shape as the x-axis.

What we also noticed was when placing the brick in such orientation that required the gripper to rotate from the original calibration pose the accuracy would decrease drastically. The quick fix for this can be seen when the robot picks up a certain brick that is going to require the gripper to orientate itself 90 degrees. The rotation is actually performed before picking up the brick, because the gripper fingers allow larger error than the small tolerances between LEGO bricks. This way the gripper is orientated the same way every time a brick is placed.

There is a one good so-called rule of thumb to remember. The robot can successfully place and build same way as a person using two fingers on one hand could. If a person fails to place the brick and the structure collapses, the robot will fail the same operation as well.

4.5 Communication between systems

There are three main systems in play: robot (UR), camera server (Raspberry) and main program (host PC). Between these all communication is implemented through sockets that operate over TCP.

Universal Robot is able to create sockets and also connect to external ones. Every script snippet that is sent to the UR controller must report back its status and optionally some other data. To accomplish this each snippet connects to a socket created by the main program and sends data to it. Connecting to a socket is computationally cheap.

The camera server creates a socket to which the main program sends capture requests. Capture request contains camera parameters and triggers image capture. The image is sent back to the main program through the socket.

4.5.1 Configuration

User configurable values are collected into a file named ***config.yml*** located in the root of the project folder. The file is formatted as YAML, a readable data serialization language. The file contains various values that are loaded on program startup. Values are explained in the user guide chapter.

4.5.2 Calibration Procedure

Platform is calibrated in three phases: color calibration, platform teaching, camera calibration.

Color as are defined as one or two ranges of HSV (hue, saturation, value) values. Color calibration is user assisted. During calibration the user labels bricks from image. Program optimizes (by brute force) minimum width range that encapsulates most of the color in user pointed area. The resulting range is widened slightly to allow some shift of colors during operation.

Platform teaching requires the user to place robot in six different spots, four corners of a build platform and two corners of a pickup area. From that information robot understands where the coordinate origin point is and the rotation of the build platform.

Camera is calibrated by recording an offset from TCP to the camera’s image center and the pixel width of a square DUPLO. During camera calibration the program expects to find a square object of a certain color (specified in config) on the first taught platform corner. Its position is known, and such the offset can be inferred with the help of an image capture. Recorded pixel width is used to convert pixels to millimeters.

4.5.3 Moving the Robot and the Gripper

Robot commanding is implemented using official URScript (version 3.4) interface [6]. Our implementation should be compatible with all major versions 3 starting from 3.4. Class *Robot* implements our remote controlling interface with features:

- Move TCP (L and J movement types)
- Display popup on teach pendant
- Get TCP and joint positions
- Transform a pose
- Convert RPY (roll, pitch, yaw) to rotation vector and vice versa

- Set TCP
- Set teach mode
- Activate gripper
- Apply force along tool z-axis

Implementing above commands is done by sending URScript snippets to the robot to execute. Those snippets can contain from one to many script commands that execute the intended actions and report back possible outputs and status messages. Each snippet has a callback that enables our program to know when sent command has finished. Currently there is no other way to poll, for example using ModBus interface, when script execution has completed.

Gripper is also controlled remotely in the same manner. However, operating the gripper requires a script of 700 lines length. This script was acquired by exporting it from a program created on the teach pendant itself. Length of the script causes a second or two of delay between sending gripper the command and seeing the gripper move.

4.6 Known Issues

Due to the process where the 3D model coordinates are transferred in to the positive quarter, the built model is inverted when compared to the original. This can and will be fixed if there is time. Other features are priority for now.

For some unknown reason the robot controller might shut down causing the teach pendant to display error claiming that connection to the controller was lost. The root for this problem is still not known when returning this report. Restarting the controller fixes the issue.

Even though the gripper and the amount it opens, and closes are set in the configuration file the percentage can differ in reality. This can happen even with no changes between initializations. The root for this problem is still not known when returning this report. Rebooting the robot cell often fixes the problem.

If the pick-up area for the bricks is set in a way that the camera can see the outside of the table, for example the floor and user’s shoes, it might recognize bricks and colors in a place where they do not really exist. This can be avoided by carefully planning the pick-up area.

Due to changes in lightning conditions when moving the robot cell or running the implementation on a completely different cell or a different back-drop the machine vision can fail to recognize some colors. In most cases recalibration of the colors fixes the issue.

One of the issues we encountered with machine vision, was in situation, when two bricks were too close to each other, especially if the blocks are the same color, that program has problem distinguishing individual blocks, sometimes the shadow can take the color of the block, and then in picture it looks bigger than in real life. Solution for this would be looking for specific area of the block in the photo.

As we implemented force mode in our application, in some cases, on placing the blocks, robot exceeded the reach of the joints when we wanted to carry out the movement. We solved this issue by moving the build platform further away from the base, but eventually **we have removed this mode from our program.**

Another problem we encountered with force mode was it works differently depending on how close to robots base it is trying to apply it, even if the value is set in the program, real life values might be smaller, and so not enough to place the blocks**, we solved this issue with removing force mode from implementation.**

While we were adding the condition for the robot to take the shortest way possible, in some cases, while moving it would go over the base and for that movement it needs to extend completely vertical, which is unnatural and unwanted pose in our system, because of the restrictions of wires and the additional Raspberry Pi mount. We solved this issue by positioning the build platform and pick-up area further away from robot, so there would not be a straight line between them that crosses too close to robot.

5. USER GUIDE

This section describes how to get the implementation running so it can be replicated later by anyone with access to the files and required components.

5.1 Robot Cell Setup

The robot should be mounted as shown in figure 20 where the z axis of the robot is orthogonal to the working table. Pickup area and build platform may be anywhere within the reach on the same side of the robot as long as they are not close to the robot's base and the areas are positioned such that the robot's **shortest path from one area to another is not over the robot's base**. The latter requirement is for preventing the robot from moving over its base, which may put cabling and the camera in danger. On the UR teach pendant set payload to 1.07 kg (weight of the gripper and Raspberry).

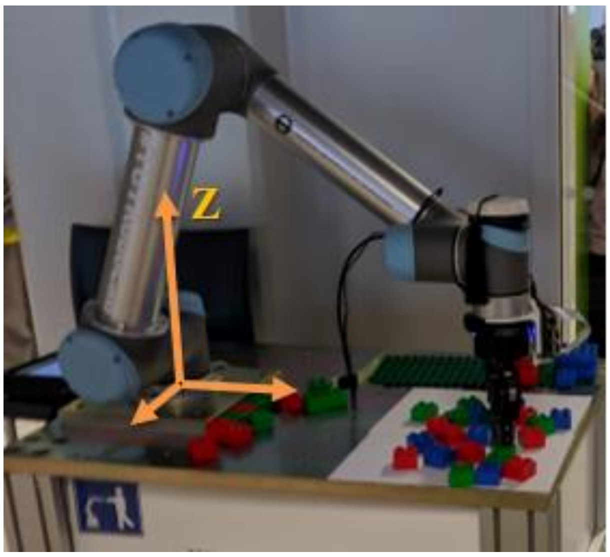
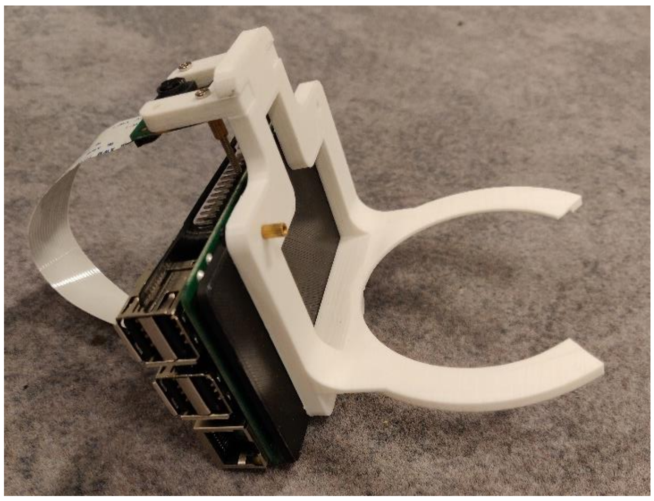
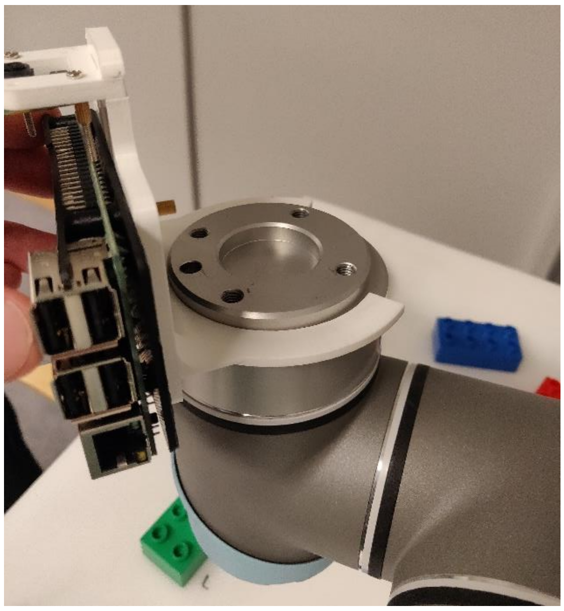


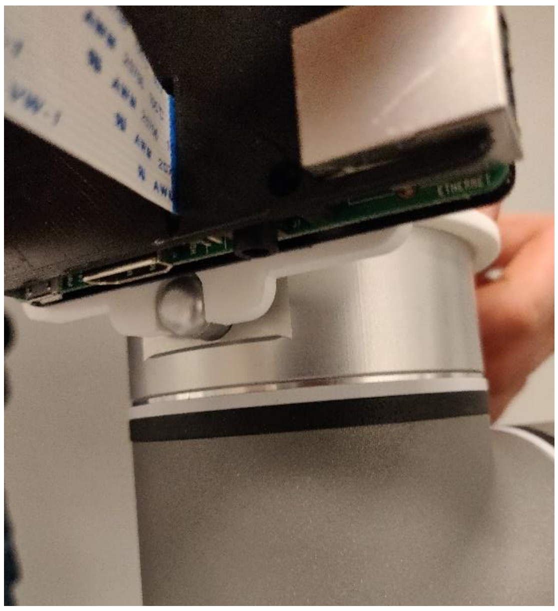
Figure 20. *UR5 mounted on a plane parallel to the working table.*

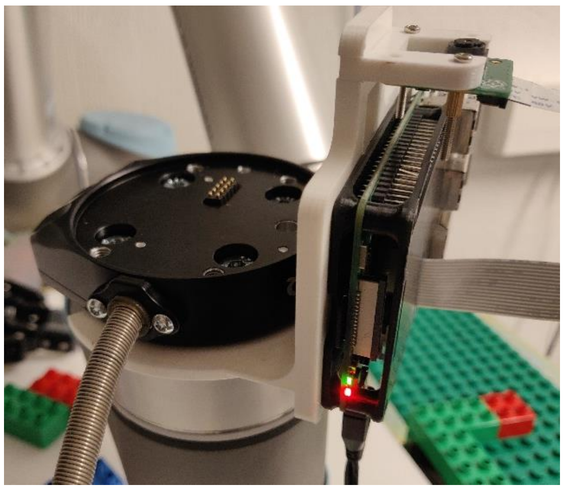
The build platform (rectangular) should be attached to the table in some way to prevent it from sliding out of position during operation. Double sided tape works well.

Print the 3D models and assemble the camera fixture, Raspberry Pi and the camera module (figure 21a). Place the Raspberry mount at the end of the robot arm (figure 21b) so that the tool base plate connector locks the mount in place stopping it from rotating (figure 21c). Tighten the mount in between the arm and the tool using the fitting ring. If the Robotiq gripper has Robotiq camera module installed leave the fitting ring out to make room for the fixture (figure 21d). Finally install the Robotiq gripper and the cables can be left free or attached to the arm itself (figure 21e).


a)


b)


c)


d)

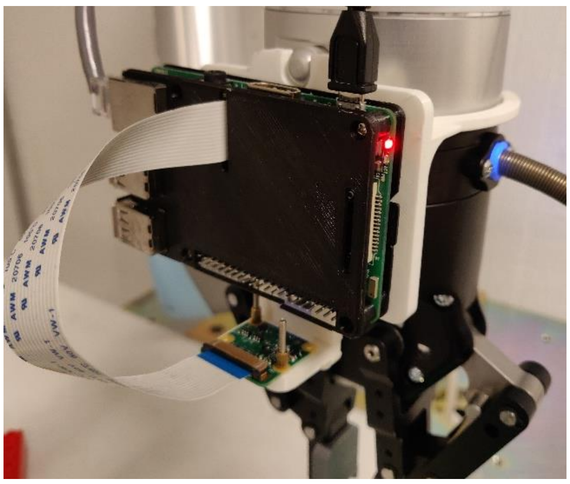

e)

Figure 21. *Installing the mount.*

Finally, a switch or a router is recommended for connecting Raspberry, robot and host PC. Connection from the robot itself is always wired through ethernet and the robot's IP is set according to the networking settings accessible from the teach pendant. Raspberry has Wi-Fi capabilities and by taking advantage of that there is one less cable to worry about.

5.2 Software Setup

Software installation is two-part: Raspberry Pi camera server setup and host PC setup. Both systems run their programs using **Python 2.7**. **UR software version** must be higher or equal to **3.4** and lower than 4.

The file *raspberry_setup.md* describes the steps to install auto-bootable camera server and its Python dependencies on Raspberry Pi and a blank SD-card. On host PC install Python 2.7 and Python package manager PIP. Then Python dependencies can be installed by running command *pip install -r requirements_host.txt* that installs Python packages listed in the text file.

User configurable values are collected into a file named ***config.yml*** located in the root of the project folder. The file is formatted as YAML. The file contains values for

- TCP – Tool center point for gripper and camera.
- Gripper closed amount in various situations.
- Calibration color –color name for the calibration brick.
- Simulation toggle – if true the system is runnable using the official UR5-simulator.
- Travel height – Height above the platform used to travel between locations.
- 2x2 brick dimensions – Ideal dimensions of 2x2 DUPLO brick.
- Calibration data file paths.
- Model name – name of a model file to be built. The file should be in the root folder of the project.
- Camera parameters.
- Gripper definition script file path – Path to a script file used as template for activating the Robotiq gripper.
- **Network configuration** – IP and port values for UR, Raspberry and host PC.

Default values should work in most cases and the user would only need to modify the network parameters. If using router use IP addresses that the router has assigned to the connected devices. Ensure that *config.yml* is identical in both systems before continuing.

5.3 Lego Digital Designer Setup

Download and install Lego Digital Designer (LDD) from official LEGO site. Find the LEGO Company folder in ***/AppData/*** and create folder named ***UserPalettes*** under the LEGO Digital Designer. After that copy the provided ***LegoAssembler.lxf*** file to the location you just created. This is a template that provides only the supported bricks.

5.4 Creating a Model

Create a 3D model using LDD or use the provided ***demo.ldr*** file. Note that after opening LDD, choose the ***Free Build*** – mode. Then click on the ***Filter bricks by boxes*** option on the lower left-hand corner and choose ***LegoAssembler.*** This is the template that includes only the bricks that are supported.

Export your model from LDD as a LDraw (.ldr) file to the project's root folder. Name it as what was defined for the model name in *config.yml*.

As mentioned earlier there is a one good so-called rule of thumb to remember when creating the model. The robot can successfully place and build same way as a person using two fingers on one hand could. If a person fails to place the brick and the structure collapses, the robot will fail the same operation as well. This should be kept in mind when designing the model.

5.5 Running the Program

The program features five subprograms: build a model, teach platform, preview taught platform, calibrate camera and calibrate colors. Python files can be run using command *python name_of_the_file.py*. Make sure you have initialized the robot and the gripper and edited the config file before running the program.

5.5.1 Color Calibration

Text labels are attached to color signatures in a user-assisted manner. Place DUPLOs of various colors under the camera and the run the program. Running python file *do_color_calibration.py* starts the guided color calibration. First the program captures an image from the camera and then the user is prompted to select and label the bricks seen in the image. Labeling process:

1. Using 4 points the user defines quadrilateral area.
2. The points and connecting lines are drawn to the image.
3. User is prompted to give a name to the color within the area.
4. A color signature is saved, along with the given name, to a file named *color_definitions.yml*.

Previous definitions of same name are overwritten.

5.5.2 Teaching the Platform

This subprogram is used to teach where the build platform and the pickup area are. Run the main program *python start_main.py* and select the *teach platform* subprogram. Place 2x2 DUPLOs on all 4 corners of the build platform and follow instructions that are shown on the teach pendant. The teach procedure

1. Select build platform origin according to figure 22
2. Teach it by moving the gripper to grab the 2x2 brick by its studs. See figure 22
3. Teach the diagonal opposite corner keeping the tool orientation same.
4. Teach the last 2 corners in same manner.
5. Teach two corners of the pickup area (ground level).

After the subprogram has finished a file named *platform_calibr.yml* is saved in the root folder. To preview the taught points run the main program and select subprogram *preview taught platform*.

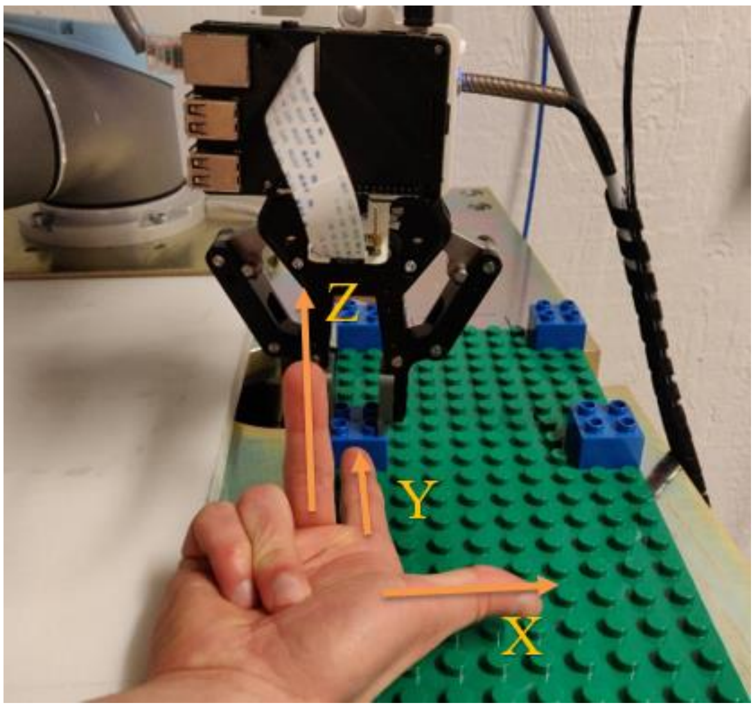


Figure 22. *Right hand rule to find the first corner during build platform teaching.*

Figure 22 demonstrates how to choose starting corner for build platform teaching. Align the gripper such that its y-axis aligns with y-axis of the right-hand coordinate frame. Keep the gripper orientation constant for all following corners. Note that in the figure only part of the dark green platform is used.

5.5.3 Calibrating the Camera

For calibration leave only the brick that was placed on the **first taught corner** (has to the color defined for calibration in config). See figure 23. During calibration the arm takes an image of the brick and uses the knowledge of its taught position to create camera calibration. Calibration file is saved to *camera_calibr.yml*. To start run the main program and select the *calibrate camera* subprogam.

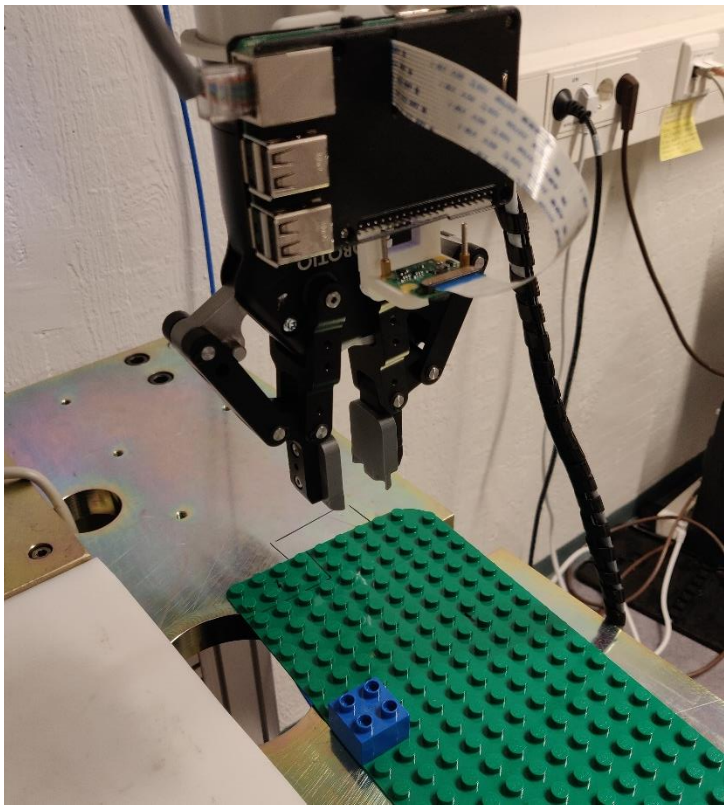


Figure 23. *Calibrating camera with one blue 2x2 DUPLO.*

5.5.4 Building the Model

Building a model requires that the user has successfully completed color calibration, platform teaching and camera calibration. The program informs user with the number of required bricks by size and color. It also displays an error message if the 3D model contains unsupported bricks.

Run the main program and select the *start building* subprogram. Before starting to build the program asks whether to continue from a saved state. If you choose to build a new one the program loads the LDraw model specified in *config.yml*. The robot starts building autonomously until all bricks are built.

Lay required bricks to the pick-up area for the robot to see them. If the robot doesn't see what it's looking for it waits until the user adds the required brick. You can see brick requirements printed to the console when the program is ran.

5.6 Troubleshooting

While running the program there might be some issues that could pop up, they could be easily avoided and if needed, repaired by applying theses simple fixes.

If there is any problem with the build of the structure, one simple problem could be that while constructing the model, the rule of thumb for the designing wasn't taken into an account, and robot physically is not able to place the block, so the solution would be going over the plans and evaluating if it is valid.

There could be situation where robot tries to grasp the block but fails, this issue could be due to not accurate enough machine-vision calibration, and in this case checking if the mount of the camera has kept the same position as it was while calibrating. Or recalibrating camera module should be carried out.

In case when robot cannot place the blocks on the platform, problem might be solved by checking if after placement of bricks, the build platform stays in place, and then recalibrating the platforms.

When there are some problems regarding machine vision, for example, finding the right blocks, color recalibration should fix the problems, it is advisable to recalibrate color every time when the lighting changes.

If robot is staying only at picturing position, make sure that all the blocks are in the pick-up area, that all necessary blocks are provided, bricks aren't too close together to each other, which could lead to problem determining blocks.

Situations when robot triggers a protective stop or a pop-up mentioning that it is close to singularity might be because one of the joints has reached its limits or robot is operating too close to base, so it must be assured that both plates are in reasonable distance from robot.

Most of problems might be caused by inaccuracies of the calibration or due to change of circumstances, so just redoing the process could resolve occurring issues.

6. EVENTS AND DEMONSTRATIONS

There were multiple events where a presentation was required. We defined specific goals for each date and this chapter is used to show what was achieved for each event. The state and results of the project were also demonstrated on multiple occasions even when not accounting for the mid-stage seminar presentations.

6.1 Kick-Off Seminar

The kick-off seminar was held on Friday 5.10.2018 after the topic selection and first meeting with project supervisor. The purpose was to present the initial project plan and vision to other students.

At this point we had a clear goal of what we wanted to achieve and had already begun the build platform inspection. Some very notable design choices were also established, for example we decided to create our implementation using Python without ROS.

6.2 Mid-Stage Seminar

The first mid-stage seminar was held on Friday 2.11.2018 to present what was achieved so far and each group shared their progress. At this point, we had already created most of the back-end, the integration from LDD and LDraw in to build instruction, the calibration process and created the second round of 3D design prototypes for gripper fingers and mount.

Even though our project plan and timetable had shifted multiple times before the seminar, we were approximately two weeks ahead of our original schedule.

6.3 Opintosuunnistus – Student Fair

The student fair was held on Tuesday 6.11.2018 and we agreed to demonstrate our project to any students who were interested in studying robotics. Student were free to walk in to the laboratory we were working in and we would give them a quick demonstration and answer any questions.

The goal for this demonstration was to get our machine vision solution to be accurate enough to pick up the bricks and integrate the build instructions so that we could replicate a simple model. Only 2x2 bricks were supported in only two colors, but we reached this goal, even though there were still slight accuracy errors in picking up the bricks.

6.4 European Robotics Week – RoboLab Open-Door Event

The open-door event was held on 22.11.2018 and we volunteered to demonstrate our project yet again, this time to outside visitors, including kids and international seminar participants. The goal was to make sure that our implementation can be used on another robot cell and it was a success. We noted that lightning conditions caused some problems with the machine vision but were fixed with recalibration. We also aimed to fix our accuracy issues and stop the robot arm to rotate unnecessarily when picking up the bricks. In the end the project was demonstrated successfully with almost no problems and it got a lot of attention from visitors.

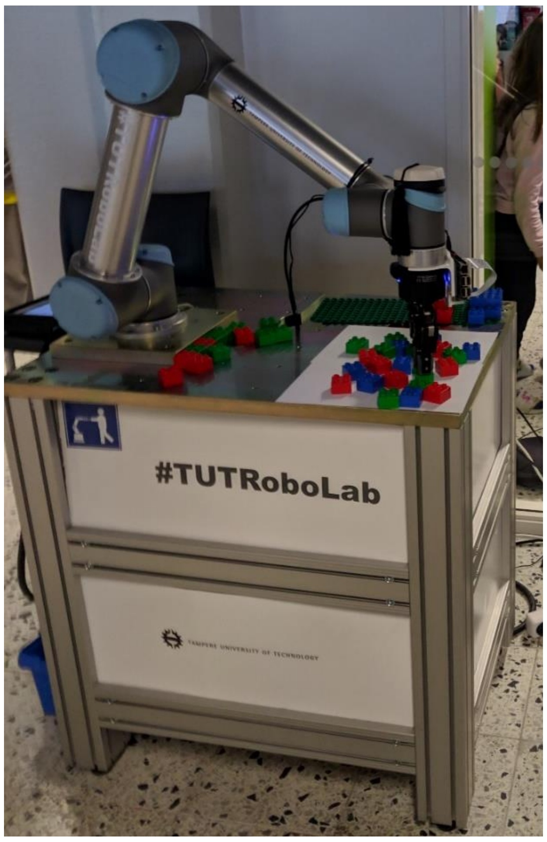


Figure 24. *Robot setup at the open-door event.*

6.5 Second Mid-Stage Seminar

The second mid-stage seminar was held on Friday 30.11.2018. This time the main goal was to implement the 2x4 bricks. Initially they had caused problems with machine vision as the system could not reliably decide the orientation of the brick and that would lead the robot to fail placing the brick correctly. This and accuracy problems caused by this were fixed by the time of the seminar and we were able to present working project that checked every single one of the initial requests that were given to us at the start.

6.6 Upcoming Demonstrations

Due to the time restrictions created by the fact that the team has one exchange student who is due to leave before the end of the year the final demonstration has to be moved forward. This means that only the development made before this demonstration is to be graded. The date of the final demonstration and the submission of this project report is set to be on 14.12.2018. To goal for this demonstration is to make everything more robust and find and fix possible errors in the code. Almost all development for additional features is halted for the last two weeks before this specific date.

After we have submitted everything for grading we move in to something we call a “volunteering phase”. In this phase we will continue to develop our project outside the course requirements even though it does not affect our grade. The goal is to implement few additional features, such as the ability to use the camera module to read QR codes that could include the information of a specific model. The deadline we have set for these additional features is 18.1.2019 which is the original date for demonstrations. In this event we are going to present our project to other teams and discuss about the course.

The final demonstration set for this project is on 25.1.2019 as part of Tieteen Päivät – science fair in Tampere-talo. After the successful demonstration at RoboLab Open-Door event we were asked if we were interested to present our project there as well. The demonstration itself is planned to be similar to the one we can achieve for the earlier event mentioned above. Planning and creation of additional flyers and ads are also required.

7. CONCLUSION

Our goal was that the robot can replicate the 3D model without any assistance, so user just needs to provide the blocks and the model. System on itself would analyze the model, read the coordinates, look for the necessary bricks in pick-up area, determine the rotation, pick up the block and place it in previously determined place.

We reached this initial goal and full-filled each requirement given to us at the begin of the course which were

- ✓ Create build instructions from 3D model
- ✓ Support for two brick sizes, 2x2 and 2x4
- ✓ Support for three colors, in this case blue, green and red
- ✓ Implement machine vision for picking up the bricks

We also implemented multiple additional features and developed things further than required. For example, our implementation has very robust calibration process for both the machine vision and the platform so that the system is very easily initialized on different cells in different environments. The machine vision does not care about the placement and the rotation of the bricks when picking them up even though initially it was originally suggested that they would only need to be found on pre-determined locations.

In the span of four months of development and project work there were multiple new things we had the chance to learn and also enhance our existing skillsets. We improved our skills in project management and teamwork. We had the chance to tackle multiple problems and brainstorm for different solutions that could drive us forward. Some of us learned plenty about robot programming while some kept improving their knowledge more and more. We all became very familiar with the UR5, its controller and capabilities and different ways to program it. It acted as a great exercise that also supported other simultaneous courses. Group members' main contributions to this project were

Julius – LDraw conversion to build instructions, head of communications
Tomi – System integration, machine vision, robot controller
Matiss – 3D-design and manufacturing.

There are multiple ideas for additional features and other suggestions we have for future development done either by us after the course is finished or by anyone interested to continue working on improving this system.

- → Web-based user-interface that could be accessed anywhere. User could easily drag-and-drop the 3D model and send it to the robot over the internet. Progress could be monitored through the camera module.

- → Implement QR-codes which would include the build information. Robot would automatically build the model after it reads the QR-code using the camera module.
- → Capability to detect when the robot fails to pick up a brick. This can be done using the camera as it can see between the gripper fingers, so it could check if something was gripped.
- → Build a complete brand around the project with ads, flyers and enclosures.
- → Make the code run completely on the Raspberry Pi. Would work perfectly with the WebUI.
- → Add support for more brick sizes and colors.
- → Safety study.
- → Improve accuracy.

Looking back at the project there are a few things that could have been done better. Even though the team reached their goal, we could have gone beyond that by giving ourselves stricter timetable with more sub-goals while at the same time constantly updating our Gantt-charts and project structure to better track our progress. In the end this came down to the fact that team members were at the same time busy with other courses. The work could have also been divided better between different team members and some of the solutions could have been implemented better. We learned from these things and we are going to pay even more attention to these particular aspects on our future endeavors.

SOURCES

[1] Introducing the UR5 – A Flexible Collaborative Robot Arm, Universal Robots, Web-page. Available (13.12.2018): https://www.universal-robots.com/products/ur5-robot/

[2] Adaptive Robot Gripper 2-Finger Instruction Manual, Robotiq, Web-page. Available (13.12.2018): https://assets.robotiq.com/production/support_documents/

[3] Raspberry Pi, Raspberry Pi Foundation, Web-page. Available (13.12.2018): https://www.raspberrypi.org/

[4] Raspberry Pi Case, Thingiverse user adamwdraper, Web-page. Available (18.12.2018): https://www.thingiverse.com/thing:2539269

[5] File Format, Ldraw Standards Committee, Web-page. Available (13.12.2018): http://www.ldraw.org/article/218.html

[6] URScript documentation SW3.4, Universal Robots, Web-page. Available (20.12.2018): https://www.universal-robots.com/download/?option=27572#section27343

ATTACHMENTS

Attachment 1: List of files and media that should be found included with this report.

Project repository: https://gitlab.rd.tut.fi/sepponen/robot-lego-assembler/

3D Models

- Gripper fingers
- Raspberry Pi mount
- Extension piece for the camera module
- Mount fitting piece

Template and test file

- LDD template named *LegoAssembler*
- LDraw file named *demo.ldr* for testing

Code

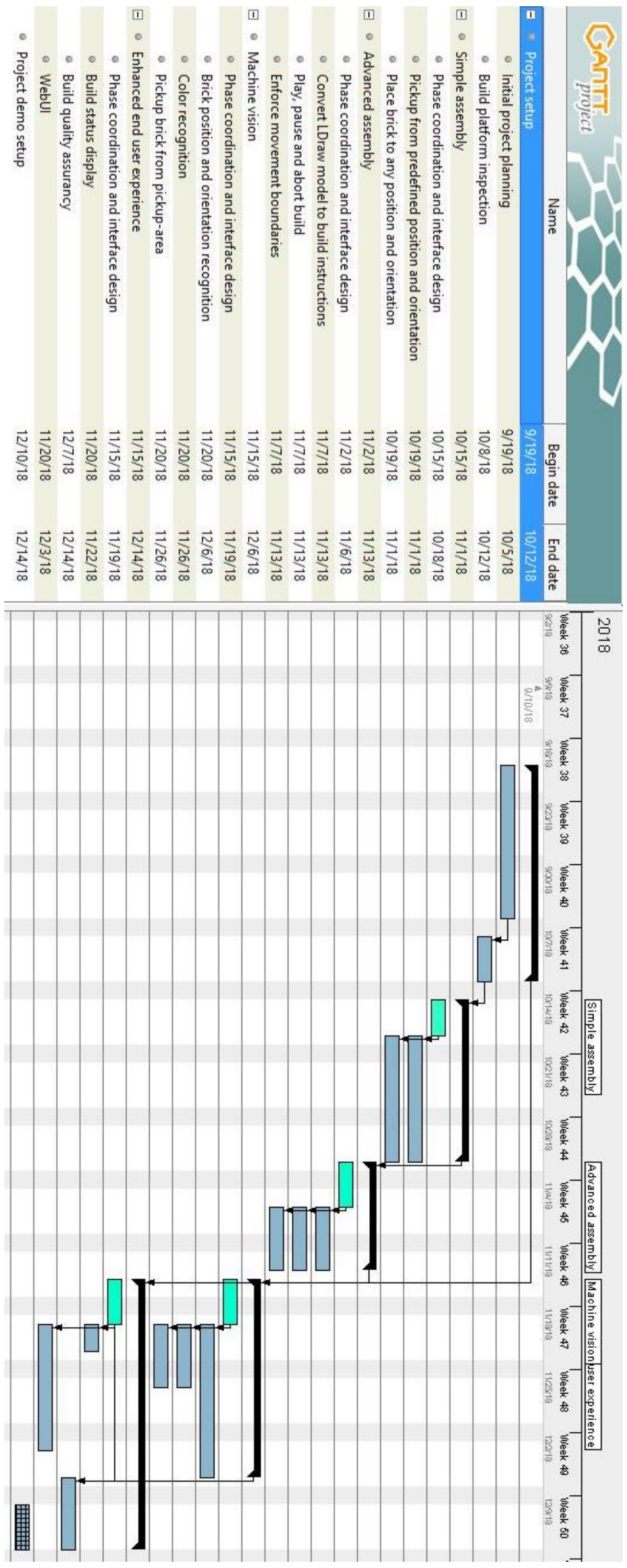
- All the code required to run *Robot-Lego-Assembler*
- Raspberry Pi setup guide
- Python dependencies requirements files *requirements_*.txt*

Media

- Video demonstration

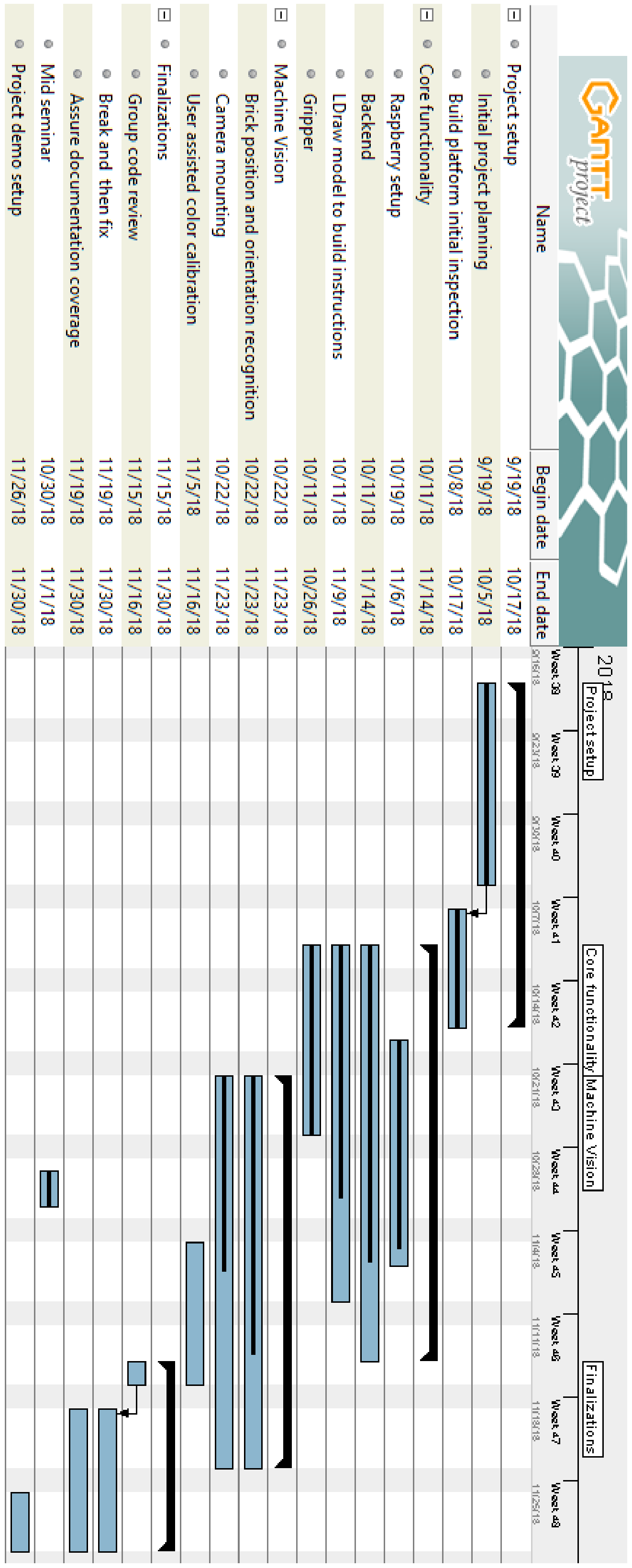
Attachment 2: Gantt-charts.

Integration model



Name	Begin date	End date
Project setup	9/19/18	10/12/18
Initial project planning	9/19/18	10/5/18
Build platform inspection	10/8/18	10/12/18
Simple assembly	10/15/18	11/1/18
Phase coordination and interface design	10/15/18	10/18/18
Pickup from predefined position and orientation	10/19/18	11/1/18
Place brick to any position and orientation	10/19/18	11/1/18
Advanced assembly	11/2/18	11/13/18
Phase coordination and interface design	11/2/18	11/6/18
Convert LDraw model to build instructions	11/7/18	11/13/18
Play, pause and abort build	11/7/18	11/13/18
Enforce movement boundaries	11/7/18	11/13/18
Machine vision	11/15/18	12/6/18
Phase coordination and interface design	11/15/18	11/19/18
Brick position and orientation recognition	11/20/18	12/6/18
Color recognition	11/20/18	11/26/18
Pickup brick from pickup-area	11/20/18	11/26/18
Enhanced end user experience	11/15/18	12/14/18
Phase coordination and interface design	11/15/18	11/19/18
Build status display	11/20/18	11/22/18
Build quality assurancy	12/7/18	12/14/18
WebUI	11/20/18	12/3/18
Project demo setup	12/10/18	12/14/18

Phase model



Name	Begin date	End date
Project setup	9/19/18	10/17/18
Initial project planning	9/19/18	10/5/18
Build platform initial inspection	10/8/18	10/17/18
Core functionality	10/11/18	11/14/18
Raspberry setup	10/19/18	11/6/18
Backend	10/11/18	11/14/18
LDraw model to build instructions	10/11/18	11/9/18
Gripper	10/11/18	10/26/18
Machine Vision	10/22/18	11/23/18
Brick position and orientation recognition	10/22/18	11/23/18
Camera mounting	10/22/18	11/23/18
User assisted color calibration	11/5/18	11/16/18
Finalizations	11/15/18	11/30/18
Group code review	11/15/18	11/16/18
Break and then fix	11/19/18	11/30/18
Assure documentation coverage	11/19/18	11/30/18
Mid seminar	10/30/18	11/1/18
Project demo setup	11/26/18	11/30/18