Sorting

By a blueprint

Project Report
By SW5-02-E16

Aalborg University

AALBORG UNIVERSITY

STUDENT REPORT

Computer Science
Aalborg University
Cassiopeia
Selma lagerløfs vej 300
9220 Aalborg Øst
http://www.aau.dk

Title:
Sorting

Theme:
Embedded Systems

Project Period:
Fall Semester 2016

Project Group:
SW5-02-E16

Participants:
Daniel Thiemer Sørensen
Thomas Krause-Kjær
Anders Sørensen
Nikolaj Lepka
Patrick Lynnerup

Supervisor:
Giovanni Bacci

Copies: 1

Page Numbers: 32

Date of Completion:
October 31, 2016

Abstract:

Sorting Lego bricks...

Contents

Todo list

Chapter 1

Introduction

This project deals with the construction and design of a machine capable of sorting and funnelling LEGO bricks into another machine which will then construct what is specified by the blueprint.

Doing so requires the bricks to arrive in the correct order, and for the bricks unrelated to the project to be rejected altogether.

To ensure cost-efficiency, this will be done with embedded systems, due to their low price and electrical usage.

1.1 The Initial Problem

How can a machine capable of sorting LEGO bricks be constructed, so that it can funnel the bricks into an assembler in the correct order?

Chapter 2

Analysis

derebyggelse introduk- on

This chapter serves the purpose of examining and deciding the components and software used to construct the sorting machine.

2.1 Existing Sorting Technologies

In this section, the different existing technologies for sorting will be described with a focus on the sorting technologies that can work with a blueprint.

2.1.1 Colour Sorting

A sorting option is one that uses colour to recognise different objects. Depending on the implementation of this technology, the colour sorting can account for visual defects, or just the colour itself. This is for instance used to sort apples[1].

A benefit of sorting via colour, is its simplicity. Colour can be represented by a few variables, for instance red, green, and blue. By using a single 32-bit variable, simplicity in algorithms can be maintained, which lowers the complexity.

One disadvantage however, lies in the fact that colour is not sufficient to properly describe or recognise objects, which limits the potential uses. When a versatile system is designed, it becomes necessary to use other variables, such as size and weight, to properly sort the objects.

2.1.2 Sorting by Image Recognition

Sorting via image recognition requires a camera and an algorithm to recognise different aspects of the scanned object; aspects such as surface defects, or general surface features. The main benefit is the ability to check for multiple aspects of the scanned object, which can lead to more sorting categories than if simply sorted by colour alone.

An example of this could be sorting soda cans. Simply sorting by colour would—for example—result in a blue and red pile; but sorting by the logo, the machine would instead sort the cans by their brand regardless of colour.

The downside of image recognition compared to a *tri-chromatic colour sensor* would be the complexity and execution time of the associated algorithm; though this downside would be outweighed by higher precision.

2.1.3 Sorting by Photo Interruption

Photo interruption consists of a infrared sender and receiver. When an object passes between the sender and receiver, it creates a temporary interruption in the signal received by the Central Processing Unit (CPU) [2]. By measuring the time of the interruption, the length of the object is found. Though this requires knowledge of the object's speed while passing through. This requires little processing power, as it only requires the ability to measure time, and perform a couple of simple calculations.

One disadvantage of this, is that it does not describe the shape of the object. This means the method is not well suited for detecting deformity. One way to use the photo-interruptor to detect these deformities would be to use several photo-interruptors placed at different angles and positions.

2.2 Preliminary Design

To sort LEGO bricks a conveyor belt and a series of assorted sensors will be used. This will run until the bricks are pushed through in the order the assembly-machine requires as specified by a blueprint.

The idea was to have bricks arrive at multiple sensors on a conveyor belt. The sensors would take care of measuring the different aspects of bricks, such as dimensions and colour. These aspects would then be cross-referenced with the blueprint and sorted according to three conditions:

1. The brick is the next one needed; in which case it will be moved to the output part of the machine.
2. The brick is needed later; in which case it will be moved to the back of the queue.
3. The brick is deformed or not needed at all; in which case it will be removed entirely from the queue.

The most obvious downside of this system, is that if n bricks are ordered in reverse order, then it will take $O(n^2)$ cycles to get the bricks out in the correct order. But as an initial idea this was considered sufficient.

Based on these ideas the following sketch was made.

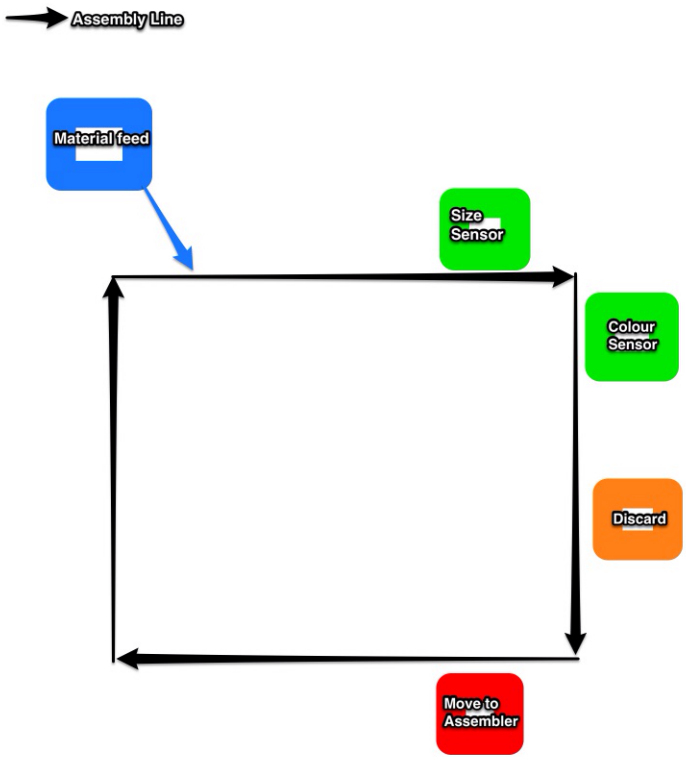

Figure 2.1: Preliminary design sketch

The sketch in [Figure 2.1] shows that the necessary hardware for this project would be the following:

- Colour sensor.
- Size sensor.
- Method for discarding.
- Conveyor belt.
- Method for directing the bricks from start towards goal state.
- Brick funnelling mechanism.

The important hardware choices will be made in section 2.3. However, this will only cover the choices that require software, meaning for instance that the conveyor will not be explained.

2.2.1 Program Mock-Up

This section elaborates what the software should handle for the project solution. The mock-up will serve as a guideline on how the software should be structured, and only contain the general structure of the software. It can can also be used to do fast prototyping and early testing, as it is one of the techniques used in Extreme Programming[3]. It will not contain any specific parts of the software, such as the code or pseudo-code.

Based on the functionality described in 2.2, the program should implement the following:

- Controlling conveyor belt
  - Starting and stopping the conveyor belt
  - Making multiple conveyor belts to work synchronously
- Identify brick, based on the blueprint
  - The laser-sensor checks the size of the brick by light interruption.
  - The program checks the blueprint instruction to determine a brick's relevance.

- The system saves the bricks required by the blueprint in memory
- If the brick is needed next, then it is sent through.
- If the brick is not found on the blueprint instructions, then it is discarded.
- If the brick is required later, then it is send to storage.

2.3 Hardware

This section will elaborate and compare different technologies to help determine their usefulness, and describe the choices concerning the project.

2.3.1 Choice of Sensors

In order to measure the length of objects, sensors capable of measuring the distance between an object's front and back are examined.

A sensor capable of reading an object's colour is also required, therefore different sensors are examined to find suitable sensors for these tasks.

A Depth Sensor allows for measuring both length and colour of an object, such as a Red, Green, Blue, and Depth (RGB-D) sensor. This type of sensor has been rejected, as it requires more processing power than most sensors. for small embedded systems where computational resources are few, sensors and actuators are preferred to ensure a lack of computational resources is not the cause of system failure.

A Distance Measuring Laser allows for calculating the length of an object, by measuring the time it takes an object to pass the sensor. It would not require much processing power, by sending a single signal and then measuring the time it takes for the signal to reflect[4]. A problem is how to measure the objects; it would be necessary to stop the objects at a specific place, it would require a system with a high level of positional accuracy. It would also require an incredibly accurate laser, as it would need to measure millimetres.

A Photo-Interrupter's requirements, is that the objects would have to pass through the photo-interrupter's sensor area at a certain angle at a constant speed. This requires the conveyor to move without slowing down. However, the low processing power

requirement along with the simplicity of the calculations makes it a more viable candidate, as outlined in 2.1.3.

The Colour Sensor is a LEGO NXT tri-chromatic colour sensor which has 3 different modules: one for measuring light, one for measuring colour, and one for giving proper lighting[5]. To detect colour, it emits three colours of light on the object, Red, Green, and Blue. The reflected light is collected by a light sensor[5].

Gik vi ikke væk fra at bruge NXT sensoren? -N Vi kunne ba ikke få den til at virke, v har ikke rigti fået taget et ordentlig bes lutning om det -ADT

The Chosen Sensors

The photo-interrupter has been chosen due to its ability to measure the size of an object while the conveyor belt is moving.

The colour sensor has been chosen due to its availability and ability to scan colour while objects are moving.

2.3.2 Choice of Motor

The motors are a primary part of the final system controlling the conveyor belt, by moving the belt when it is required, however, it is not a big impact on the software implementation.

A Servo Motor would have to be modified before it could grab the object, and get the angle at which the servo is positioned. This could be useful in some cases where a conveyor belt is either not useful, or not enough. However, needing to locate the object creates an unnecessary point of failure, compared to only using a conveyor belt.

LEGO DC Motors provide several built-in methods for controlling the speed and direction of an object. It might require special hardware to properly access the built-in methods, but as it uses the same connectivity method as the colour sensor, it should not pose a problem.

This is wank ellers er det formuleret meget mystisk og skal omskrives alligevel -T forslag i kom mentar -A

2.3.3 Considered CPUs

As a choice of CPU, three different platforms were examined: LEGO NXT, Raspberry Pi, and the Arduino Micro Controller. For any chosen sensor, the processing power

behind it must be kept to a minimum due to hardware cost compared to computational cost.

The LEGO NXT uses a 32-bit ARM7 micro-processor[6] running at 55 Megahertz (MHz) [7]. This version was introduced in 1994 and is not the newest on the market, but is provided by Aalborg University for the project. The ARM processor is generally a powerful tool, and too powerful for the tasks and jobs in this project.

The Raspberry Pi uses an ARM1176 processor[8], which runs at 750 MHz and can be clocked up to over 1 Gigahertz (GHz)[9]. Since the Raspberry Pi is magnitudes more powerful than the NXT, it is safe to assume that it is also too powerful for the project.

The Arduino is a small open-source platform, providing sets of digital and analog Input/Output (IO) pins. To program an Arduino, the Arduino Integrated Development Environment (IDE) which is written in Java will be used[10]. The Arduino language is based on C and C++[11].

There are many different varieties of Arduino, some of the most popular have been listed here:

- **Arduino Uno**
  The Arduino Uno is based on the ATmega328 using a clock speed of 16 MHz and a 32 Kilobyte (KB) flash memory[12]. The Uno has 6 analogue IO pins and 20 digital IO pins, where 6 provide Pulse-Width Modulation (PWM) [12] for simulated analogue output.

- **Arduino Mega**
  The Arduino Mega is based on the ATmega2560 using a clock speed of 16 MHz and a 256 KB flash memory[13]. The Mega has 16 analog IO pins and 54 digital IO pins, where 15 provide PWM output[13].

- **Arduino Yún**
  The Arduino Yún is based on two processors, the ATmega32U4 for the Arduino and the Atheros AR9331[14]. The ATmega32U4 uses a clock speed of 16 MHz and 32 KB of flash memory[14]. The Atheros AR9331 uses a clock speed of 16 MHz and 16 Megabyte (MB) flash memory[14]. The Yún has 12 analog IO pins and 20 digital IO pins, where 7 provide PWM output[14].
  Additionally, the Atheros processor runs a small Linux environment to let the Yún connect to the Internet.

Some Arduino models might not have enough General Purpose Input/Output (GPIO) pins to support the necessary amount of sensors and motors. However, as the only difference in code between the Arduino models is the designation of pins. It will be possible to use a smaller model first, and then upgrade it later if necessary.

The chosen CPU is the Arduino Mega because of the increased number of GPIO pins. This CPU will give a large number of GPIO making it easier to create the system, while also being economic. It will also be easier to change and use a different Arduino board, due to the code compatibility.

2.3.4 Hardware Testing

In this section, preliminary ideas for tests of the hardware used will be described in this section, to later evaluate the chosen sensors 2.3.1.

Photo-Interrupter

- **Latency**
  The latency testing will show the maximum amount of LEGO bricks which can be put through the photo-interrupter in a certain amount of time. While the photo-interrupter is still able to differentiate between the different LEGO bricks.
- **Consistency**
  Checking how consistently the photo-interrupter senses a certain object, making sure there is as little deviation between the tests as possible.
- **Proximity**
  Testing the proximity in which the photo-interrupter works. This is not the proximity from the photo-interrupter to the LEGO bricks, but between the sender and receiver of the photo-interrupter itself.

Colour Sensor

- **Proximity**
  For testing proximity, a LEGO brick will be scanned at different lengths. The results will be compared to see which will be the most suitable for this project.

- **Latency**
  For testing the latency, the maximum amount of bricks that can be sent through the colour sensor area—while still maintaining accuracy—will be counted.
- **Differentiation**
  For testing the differentiation, different coloured bricks will be run through the sensor, the outputs will then be compared to see which are the most suitable.
- **Reflective Light**
  During this test, the sensor will be tested under different lighting conditions, and it will be estimated which one is the most accurate.

Motor

- **Braking-Precision and Accuracy**
  For testing braking-precision, the conveyor belt will be run at increasing speed while measuring the braking distance.
- **Revolutions Per Minute (RPM)**
  Here the motor's power consumption is measured at certain speeds, to achieve proper control of the motors.

2.4 Requirement Specification

To ensure quality of the system, it is crucial to specify appropriate requirements, the MoSCoW method will be used to improve this. This method splits the requirements into four categories *Must, Should, Could, and Won't have*, this specifies a rating of the different requirements depending on their importance to the product.

Must Have

- **Sorting of Shape**
  Selecting the proper object based on size criteria given by the blueprint, and rejecting it if any deformity or size not specified by the blueprint is found.
- **Conveyor Belt**
  A functional conveyor belt capable of transporting at least ten objects—through the sensor area—per minute, without any objects falling off, turning over, or providing inaccurate information for the sensors.

- **Reading a Blueprint**
  The product should receive a sorted list of objects, which it will output according to the order in which the blueprint requires them. It must be capable of deciding, whether the required list of objects can be recognised by the sensors.

Should Have

- **Colour-Sorting**
  Selecting the proper object based on colour criteria given by the blueprint, and rejecting it if the colour is not the specified in a blueprint, or returning it to the back of the line.
- **Efficiency**
  Adjust the speed of the conveyor belt, to minimise the amount of idle processing time for the system. Meaning there will be as little time as possible between the calculations on blocks.
- **Intelligent Sorting**
  Having the system attempt to calculate the optimal order, in which the objects travel towards the sorted output position.

Could Have

- **Assorted objects**
  Sorting a broader variety of objects, compared to basic LEGO square objects.
- **Internal Communication**
  Separating each sensor into its own module, with each module communicating any potential problems between them.

Won't Have

- **Assembler Segment**
  The segment after sorting the objects through the conveyor belt, wherein the assembly process of the blueprint takes place.
- **General Purpose**
  The sorter is not meant to be general purpose, it will as such only support special building materials.

- **External Communication**
  Letting an external part of the system—such as the aforementioned assembler segment—communicate with the sorter by sending it custom requests.

- **Underside Checking**
  The sorter will not check the underside of an object to see if there is any deformity, or for measuring the object's size.

2.4.1 Non-Functional Requirements

To ensure consistent expectations, constraints for the design has been made.

- **Usability**
  Ensuring that the orientation of the object does not affect the system.

- **Efficiency**
  The system should be able to process at least 10 objects per minute

- **Space**
  The prototype should not require more physical space than $1m^3$, whereas the final system solution should be scalable.

- **Dependability**
  There should at most be 1% error rate processing size, 2% error rate for colour and 0.5% error rate for deformity.

2.5 Problem Definition

Based on the previous analysis, these questions were constructed to formalise what was needed of the system.

- How could a sorting machine be constructed, capable of sorting LEGO bricks based on both colour and size?

- How can such a system be designed and built on an embedded system?

2.6 Prototype

The purpose of the prototype iterations, are to get a better understanding of how the physical prototype works in comparison to the design and problems it encounters.

The First Iteration of the Conveyor Belt gives insight into how the LEGO bricks moves along the conveyor belt. As the bricks might not have the correct orientation on the conveyor belt, it is important that the bricks face the sensors in a way whereby the sensors can process them correctly. The first optimisation gradually narrows the width of the conveyor track, such that the bricks can only face a certain direction.

This implementation can be seen in [Figure 2.2], where the gradually narrowing mechanism is highlighted by blue. A problem with this solution is that the bricks get stuck coming into contact with the narrowing mechanism at certain angles.

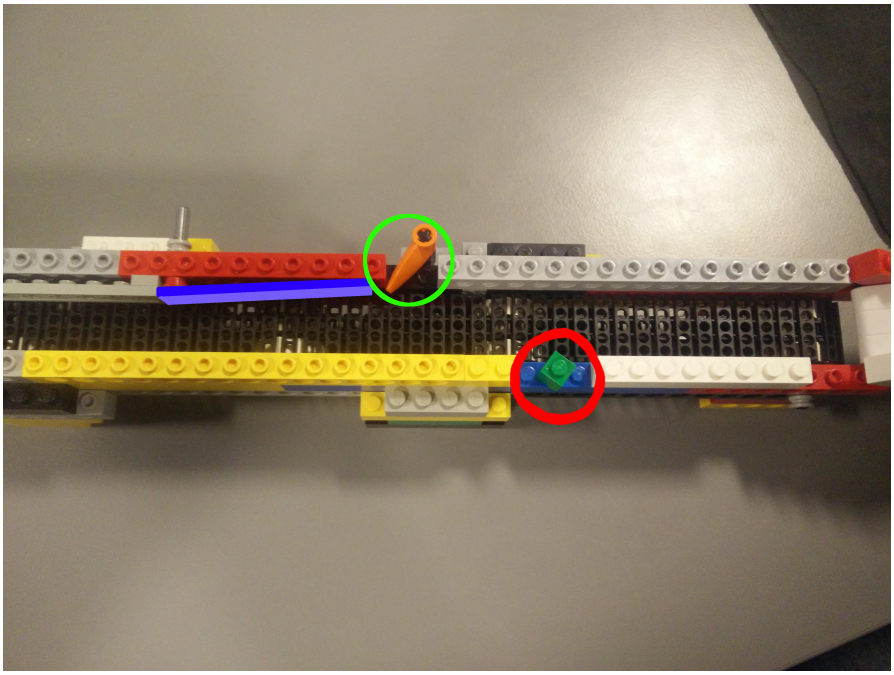

Figure 2.2: Prototype

This will be solved by adding the piece marked by green in [Figure 2.2], this piece rotates the bricks to reduce the eventuality that they get stuck when the conveyor belt track starts narrowing.

It is not very clear from [Figure 2.2], but there is a very slight indent towards the conveyor belt from the LEGO brick.

This will be solved by adding an additional LEGO brick, marked by red in [Figure 2.2] which forces the bricks away from the side, hereby rotating them slightly. It is not very clear from [Figure 2.2], but there is a very slight indent towards the conveyor belt from the LEGO brick.

Chapter 3

Design

This chapter illustrates and explains the product design. The first section covers the overall design, while the following sections refer to the individual parts of the design with a more detailed explanation.

giovanni: ret tet fra "in-depth"

After some consideration, the original sketches have been expanded, which have resulted in what's presented in [Figure 3.1].

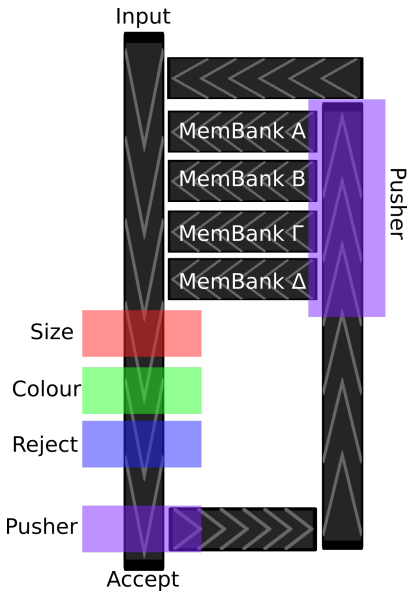

Figure 3.1: The chosen design

The differences between [Figure 2.1] and [Figure 3.1] will be explained later in the chapter.

3.1 Size Detection

The size of a brick will be found using photo-interruption. The length will be found based on the velocity equation:

$$velocity = \frac{distance}{time}.$$

The velocity calculation of the conveyor belt, is based on the amount of power drawn by the motors. This enables the system to continuously adjust the speed of the conveyor belt, allowing it to pick an efficient speed to work at.

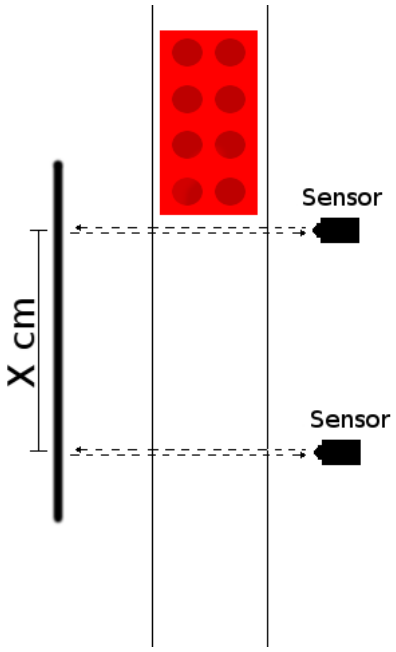

Figure 3.2: Design for size detection

The bricks pass the first sensor and breaks the stream of continuous data, and sends new data for an amount of time calculated by the formula mentioned at the beginning of this section. Another interrupter will be placed after the distance X to keep track of the brick's location.

3.1.1 Identification throughout the system

Photo-interrupters are used throughout the system to identify potential locations, making it possible to measure their size at any time. It would work by measuring a

brick's size at each photo-interrupter, and comparing it to the expected brick at that location. This is not a necessary feature but a constant check and comparison in the system, guarantees proper knowledge of the objects order throughout the system.

3.2 Colour Detection

The colour sensor will be responsible for detecting any passing brick, by measuring a difference in colour. The sensor will compare it to black, to have one coherent coloured background. A photo interrupter should be placed next to the colour sensor, as this will enable the system to also detect black bricks. The process continues until either the blueprint's requirements are met, or an error occurs.

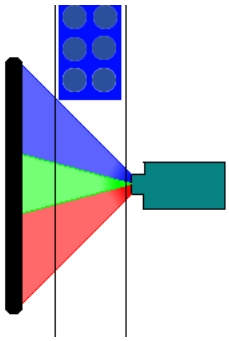

Figure 3.3: Design of Colour sensor

The bricks on the conveyor belt are moved in front of the colour sensor as seen in [Figure 3.3], where the colour sensor reads the brick's colour and sends the signal to the processor.

3.3 Sorting Machinery

This section elaborates how the system sorts bricks and explains the theory behind the used concepts.

3.3.1 Funnelling Mechanism

The funnelling mechanism is responsible for supplying bricks to the conveyor system, ensuring proper orientation of bricks. This mechanism is mainly hardware-oriented,

whereby it will be structured to increase the chance of a brick being oriented correctly before it gets to the sensors.

3.3.2 Conveyor System

A series of conveyor belts move the bricks around to the various parts of the system. The speed of the conveyor will be set dynamically to enable run-time speed changes using the measurements from the built-in tachometer of the LEGO dc motor.

3.3.3 Memory

The bricks will then be guided to the storage part of the system as seen in [Figure 3.1]. To detect where in the system the bricks are; photo-interrupters will be used.

3.3.4 Pushers

To push the different bricks to the various conveyors, servos will be used. They will control a simple arm which will be extended whenever a brick needs to be moved.

3.3.5 Scheduling of Bricks

In order to properly schedule the system's tasks, a priority queue will be utilised to ensures critical parts are prioritised higher, which gives a list of tasks in an order ready to be executed at any time. The most important task will be the one currently running; the size sensor has a higher priority than the conveyor belt moving the brick.

3.4 System Architecture

This section explains the architecture of the system.

The class structure as seen in [Figure 3.4] has been developed to give a general overview of how the system should be structured.

Since the Arduino is used, the built-in functions `setup()` and `loop()` will be used as the primary entry points for the program. The `controller` class handles and delegates

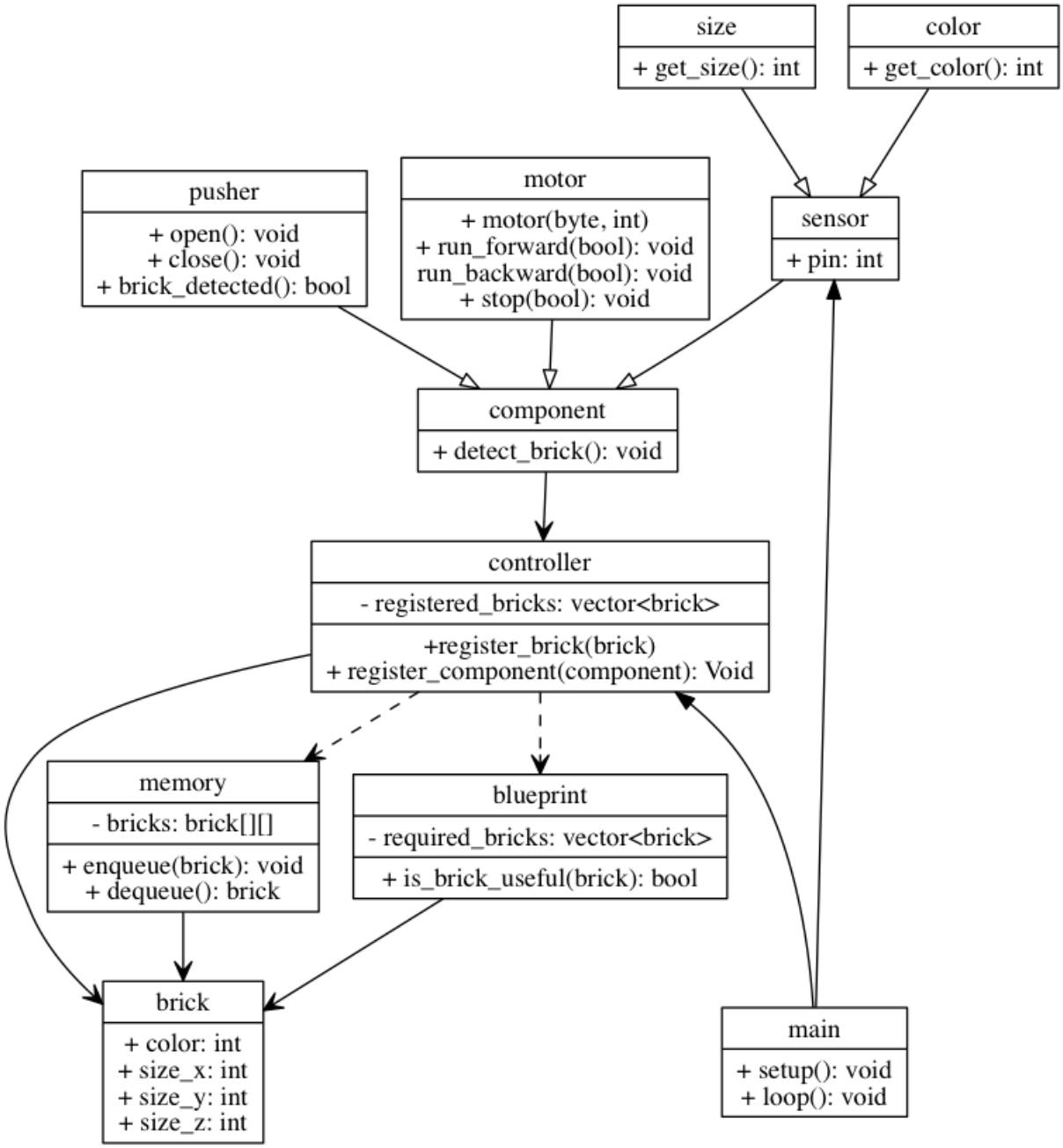

Figure 3.4: First iteration of the system structured.

the work done by the rest of the system, it will be accessed based on context of the individual brick read by the sensors.

A run-through of the program could go this way: the size sensor first calls the `detect_brick()` method with the newly instantiated brick. It then calls the `register_brick` ↪ `()` method in `controller`. The `controller` determines, from using `is_brick_useful` ↪ `()` from `blueprint`, that this brick should be placed in the memory bank, so it calls the `enqueue()` method with the brick as its input.

This structure puts a lot of emphasis on the `controller` class, and makes it easier to properly organise how each part runs, while also letting the separate classes focus on their own calculations. A disadvantage of this is that the large emphasis on `controller` can create a class with too many responsibilities, which is considered bad practice in object-oriented design. The problem is that minor changes in the object can result in unintended side-effects This large object will only pose a problem if not designed with scalability and testability in mind; meaning that for this project the benefit will outweigh the disadvantage, whereby this structure was chosen. It is important to note, that `main` in [Figure 3.4] is not an actual class, but rather the C++ file that contains the entry point of the program.

Chapter 4

Implementation

The implementation chapters purpose is to highlight significant parts of the code, solutions and the reason for implementing them.

Chapter 5

Test

Chapter 6

Scheduling

Chapter 7

Discussion

Chapter 8

Conclusion

Chapter 9

Reflection

Bibliography

[1] Guo Feng and Cao Qixin, editors. *Study on Color Image Processing Based Intelligent Fruit Sorting System*, June 2004. Research Institute of Robotics Shanghai Jiao Tong Universify. URL http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=1343622. last seen: 2016-09-20.

[2] *LTE-302 + LTR-301 Datasheet*. LiteOn, 2016. URL https://www.sparkfun.com/datasheets/Components/LTE-302.pdf,https://www.sparkfun.com/datasheets/Components/LTR-301.pdf. last seen: 2016.09.20.

[3] Never add functionality early., 1999. URL http://www.extremeprogramming.org/rules/early.html. last seen: 2016.10.26.

[4] *GP2Y0A21YK Datasheet*. Sharp, 2016. URL https://www.sparkfun.com/datasheets/Components/GP2Y0A21YK.pdf. last seen: 2016.09.20.

[5] Color sensor, 2012. URL https://shop.lego.com/en-US/Color-Sensor-9694. last seen: 2016.10.07.

[6] LEGO. Nxt intelligent brick. website, 2006. URL https://shop.lego.com/en-US/NXT-Intelligent-Brick-9841. last seen: 2016.09.18.

[7] At91sam7s256, 2012. URL http://www.atmel.com/devices/AT91sam7s256.aspx. last seen: 2016.10.12.

[8] *BCM2835*. Raspberry Pi, 2016. URL https://www.raspberrypi.org/documentation/hardware/raspberrypi/bcm2835/README.md. last seen: 2016.09.18.

[9] *ARM1176 Processor*. ARM, 2016. URL http://www.arm.com/products/processors/classic/arm11/arm1176.php. last seen: 2016.09.18.

[10] Download the arduino software, 2016. URL https://www.arduino.cc/en/Main/Software. last seen: 2016.10.07.

[11] Language reference, 2016. URL https://www.arduino.cc/en/Reference/HomePage. last seen: 2016.10.07.

[12] Arduino uno, 2016. URL http://www.arduino.org/products/boards/arduino-uno. last seen: 2016.10.07.

[13] Arduino mega 2560, 2016. URL https://www.arduino.cc/en/Main/ArduinoBoardMega2560. last seen: 2016.10.07.

[14] Arduino yún, 2016. URL https://www.arduino.cc/en/Main/ArduinoBoardYun. last seen: 2016.10.07.

Glossary

CPU Central Processing Unit

MHz Megahertz

GHz Gigahertz

RGB-D Red, Green, Blue, and Depth

GPIO General Purpose Input/Output

IO Input/Output

IDE Integrated Development Environment

PWM Pulse-Width Modulation

KB Kilobyte

MB Megabyte

Appendix