Robotic Waffles

System Design Description

[1 Introduction 3](#_Toc21687190)

[1.1 Background 3](#_Toc21687191)

[1.2 Purpose 3](#_Toc21687192)

[1.3 Scope 3](#_Toc21687193)

[1.4 Version history 3](#_Toc21687194)

[1.5 References 3](#_Toc21687195)

[1.6 Abbreviations and Definitions 4](#_Toc21687196)

[2 System Overview 4](#_Toc21687197)

[2.1 System Behaviour 4](#_Toc21687198)

[2.2 Robot system limitation 4](#_Toc21687199)

[3 System architecture 5](#_Toc21687200)

[3.1 Architectural Design (High level) 5](#_Toc21687201)

[4 Hardware 5](#_Toc21687202)

[4.1 Overview 5](#_Toc21687203)

[4.1.1 Difficulties 5](#_Toc21687204)

[4.2 Work cell layout 5](#_Toc21687205)

[4.2.1 COSIMIR 5](#_Toc21687206)

[4.3 Robot interface limitations 5](#_Toc21687207)

[5 Software 6](#_Toc21687208)

[5.1 Overview 6](#_Toc21687209)

[5.1.1 Robot control 6](#_Toc21687210)

[5.2 Communication viewpoint 8](#_Toc21687211)

[5.3 Dependency viewpoint 8](#_Toc21687212)

[5.4 Logical viewpoint 8](#_Toc21687213)

[5.5 Information viewpoint 8](#_Toc21687214)

[5.5.1 Positioning 8](#_Toc21687215)

Choosing the relevant views

Documenting a view

Documenting information that applies to more than one view

# Introduction

## Background

Bought robot from foreclosure, friend will host a wedding party, everybody loves waffles.

## Purpose

This document describes the system design with both hardware and software for the Robotic waffle system. The purpose of the document is, similar to the requirement specification, to address and solve problems in a top-down fashion before any implementation takes place to save time and improve the quality of the system as well as document it for a further audience.

## Scope

This document should not go into detail about tasks that are already known by the developer, for information to exist in this document it should be documenting design decisions and giving the big picture to aid in identifying problems/features/modules.

Example of details which may be omitted is low level commands used, wiring, simpler part of the software etc. ie, things that can only be done in one way or when how it is solved doesn’t really matter.

## Version history

|  |  |  |
| --- | --- | --- |
| Version | Changed by | Changes |
| V1 | RA | First released version |

## References

#1 This section should list all the applicable and reference documents, identified by title, author and date.

1. 6

## Abbreviations and Definitions

#1 This section should define all terms, acronyms and abbreviations used in this document. Particular care should be taken to define terms that are specific to the application.

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

# System Overview

## System Behaviour

## Robot system limitation

The movements supported by the controller seems to be atomic commands, this has the unfortunate side-effect that for all movements we should know in advance where it should stop. This works well in static environments where there are no surprises inter-movement. This definitely a limitation where the workaround is a “micro-step” with start/stops that puts excessive wear on the robot unless properly deaccelerated (which can be done but slows down the robot a lot).

In a way this is not all that bad. In reinforcement learning modelled as a markov-decision process the state-space is discrete. If the state space does not contain a time-dependent component such as the case with velocity/acceleration, then the solution wont be affected by the robot controller’s limitations.

Continuing with the MDP kind of problems, the action-space could include increasing/decreasing joint-angles, tool-position, tool-angle, tool-rotation.

# System architecture

## Architectural Design (High level)

Cosimir: Set positions and debug movement between positions

Python: Call movement between positions to accomplish a task.

# Hardware

## Overview

**Setup for attaining and pouring batter**

How the batter is to be poured depends on robot reach and flexibility, it is only 5 axis. This affects the positioning of the waffle irons and the design of the pouring tool.

The ladle should solve attaining batter which is easiest to putting it straight down vertically.

The ladle should also pour batter in a controller way, by tilting from vertical to horizontal while maintaining about the same x,y,z position above the waffle iron.

### Difficulties

## Work cell layout

### COSIMIR

Workspace is modelled inside COSIMIR to aid in development by creating a virtual 3d environment of the workspace in which the robot positions and movement type can be decided by jogging the robot to position.

# Software

## Overview

### Robot program

There are two main ways of controlling the robot, via controller or PC.

On closer consideration according to the rationale below the most suitable way is using the PC as the master and let the controller hold the positions of the workspace and execute MELFA BASIC commands sent from the PC. The PC will in most cases not need to know about the co-ordinate system, only how positions relates to different tasks and the relationship between positions (such as which positions are legal moves from another position).

If the PC needs to travel to positions not in Controller memory, then relative co-ordinates should be preferred. This at least makes the PC de-coupled from the global co-ordinate system.

#### Controller

To use the controller a program is written in MELFA BASIC III which is supported by the controller and can run stand-alone without need for a computer. This could likely be a feature complete solution which can be entirely developed and tested in COSIMIR.

#### PC

The second main way of operating the robot is running the program on a computer which communicates with the robot controller only in terms of robot-positions and reading/writing IOs.

*Controller vs PC solution*

|  |  |
| --- | --- |
| **Pros controller** | **Pros PC** |
| Can test run the entire system in 3D and simulate inputs | Can use state of the art programming languages and libraries |
| Don’t need to bring a computer | Can interface to cameras and other advanced sensor |
| Likely much more reliable during operation, with serial communication is sometimes goes into error mode. | Can be much more dynamic |
|  | A lot easier to program |

Out of these two options the PC is much more suitable for my “experimental”/prototyping needs while the controller method would be more suitable for a production environment. The problem of not being able to test-run in 3d could be solved by still use the COSIMIR for simulation. This is achieved by having the PC program dry-run and write its movement commands into MELFA BASIC format and afterwards import the commands into COSIMIR for test running in the virtual 3d environment.

#### Controller & PC together

A third way is running a mix between PC and controller, letting the robot controller be responsible for executing all position changes/steps of movement relating to a task and run a program on a PC which simply tells which task to perform and ship with it a few parameters. This leads to lose coupling between the PC and Robot in terms that the PC will never know about the co-ordinate system or work-cell setup, it simply has an interface that accepts a set of tasks with parameters.

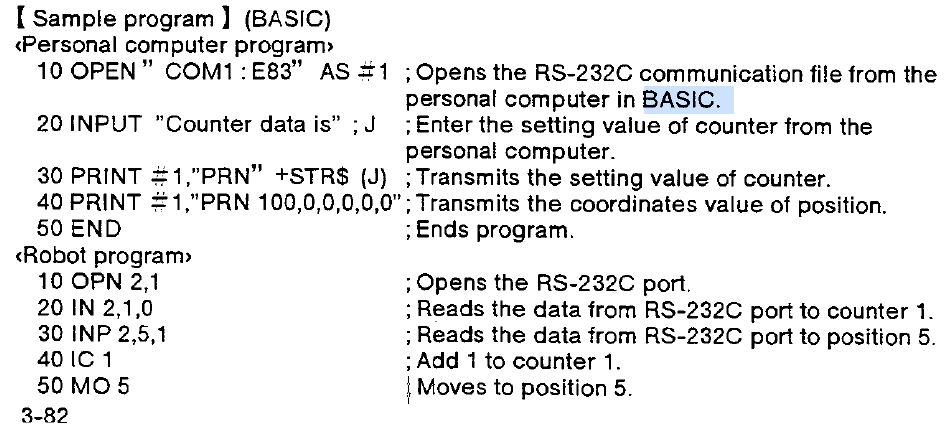


Figure 1. Example of reading from PC within Controller software

However, looking at code example above suggests that passing arguments from PC to Controller is a tricky process, where the controller needs to poll the serial-port and it is far from obvious how variables are set from serialport and used in conditional statements. An ideal scenario would be that one could send “Start waffle-iron, 3” and the controller would parse it and call a suitable sub-routine passing the #3 as parameter. This does not seem possible in straight forward manner.

Therefore, the viable option for this system seems to be the second case, using the PC as the master and the Controller will simply execute the MELFA BASIC commands that the PC sends.

One thing the Controller could help out with, is storing the various robot positions since these can be called by the PC and as long as all positions required to solve the tasks are present in the controller, the PC will never have to deal with co-ordinates, it simply tells the robot in which way it will move between positions in memory. When the memorised positions are not enough, then it can send co-ordinates as well (but with the dangers of crashing).

### Robot movement

The robot can move with two different kinds of commands:

* position-commands using position indexes for positions stored in the controller
* coordinate-commands where nothing is stored in the controller

Both are possible to use in this project, and as mentioned in chapter 5.5.1, the benefit of storing positions in the controller is that they can be downloaded in the virtual 3D environment of COSIMIR. Otherwise it would be needed to use the “where” command on the robot once it has been jogged in place, and copy that information into the source-code as constants. That’s an additional step where mistakes can be made and it gives the source-code the responsibility of storing the global co-ordinate system of equipment.

To get better encapsulation of the global co-ordinate system the controller will take care of it and the source code’s only connection to the global system will be through the position indexes. We will see if this is possible!

**Offsetting positions**

To offset positions, we could take a global position and sum with our offset but that requires us dealing directly with the global coordinate system and we said we shouldn’t do that.

So instead we will use the Melfa command “Move approach” which will sum two positions that are stored in the controller. Since only the equipment origin for each piece of equipment will be stored in the controller, we need to send our offset-position to the controller before we can call the “move approach”.

Adding a position to the controller is done by “PD <posinx>,x,y,z,A,B” where <posinx> is an integer between 1-999, just need to make sure that it doesn’t overwrite equipment positions. To prevent keeping track of multiple position-indexes a single specific “working memory” position-index can be used and overwritten each time.

**Safeguarding the offset**

Since moving to a position offset would require two commands (i.e. not atomic), first store the offset and then execute movement, it opens up for an error where the first command is not received, and in that case the previous used offset would be used! A way of safeguarding would be resetting the offset-position to zero after each move and prior to any move, read the offset-position and make sure its zero, then in worst case it would simply not offset and in turn requiring that the origin of an equipment is such that if the robot goes there, it will not crash.

### Task scheduling

Since multiple jobs(waffles) will be executed in parallell with the robot arm as shared resource, the turn order for executing tasks in each job needs to be determined.

A task is the smallest possible meaningful discrete state change, list of tasks below:

* [Common state change] Changing tool
* Retrieving batter and pouring it into iron
* Open iron
* Close iron
* Grabbing fried waffle from iron and putting it on tray

Changing tool is of special consideration since it is the only task where one job might directly interfere with another. A scenario that illustrates this is where tool Z is equipped and one job requires tool X and a second require tool Y and each job gets to execute every other turn. Then it would change between tool X and Y without progressing with the waffle making.

**Task priority**

This is taken care of by prioritizing between jobs in this order:

1. Jobs that needs to serve a waffle as next task (if multiple exists then use 2.)
2. Jobs that needs the same tool as already equipped

## Communication viewpoint

The PC connects via serial and needs to have error handling for serial communication and perhaps resetting the robot controller once error has occurred.

## Dependency viewpoint

## Logical viewpoint

See class diagram in external StarUML file.

### Tasks

The task design entity is responsible for executing the Equipments operations. It has no connection to the robot control and the tool changes required by the equipment for operation is done inside the robot module. Since knowing if a job would do a tool change in the next step is important for prioritization between jobs, it must be possible to ask a job if it intends to change tools for the next task (to which tool is not important, we just want to avoid tool changes).

How should tasks be implemented? As can be seen in Tasks 5.1.3, there is a variety of different tasks. By letting the tool change be part of the equipment operation that requires it, it does not need to be implemented as an explicit Task. But what about the rest?

Either we have each task as a class with these properties:

* Finishes itself in a single execution (atomic), since we don’t want it to encroach on Job’s responsibility of making multiple meaningful state changes.
* Can tell if it will need to do a tool change with current robot state
* Can be ask for Equipment needed and be supplied with that Equipment on which to perform operations on, such as a Bowl or a Tray.
* Calls one or more operations on the equipment in a fixed order, this is the main responsibility.
* SHOULD NOT explicitly do tool changes, that’s the equipment’s responsibility.
* Should have a few simple settings in its constructor so we don’t need one Task for close iron and one for open iron. I don’t know if this is sane though...

Or we could have everything hardcoded into its superior, the Job, which would reduce the number of classes needed.

In that case, job would still need to store the requirements of each individual Task and collect the Task code into distinct methods corresponding to each task-step. That definitely breaks single responsibility principle. Also making modifications to tasks would be difficult.

### Jobs

The design entity Job in this project is responsible for going through the Tasks required to achieve a goal, what goal is determined by the Job implementation itself.

It needs to have the following functionalities:

* Execute tasks in order
* Get information about from which bowl to take batter
* Get information about which tray to output the waffles to
* Pass the Equipment objects that the Tasks need when needed[1]
* Be loaded with a list of Tasks, preferable via dependency injection from some other class that creates both the Job and the Task list.

[1] This is difficult and here I have the option of passing a ref to the ResourceHandler into the Task instead of supplying the Equipment. It doesn’t feel alright to let the silly little Task borrow things directly from ResourceHandler without letting its superiors have a say. Maybe I should trust my Tasks more?

What could be the problem of letting a bunch of Tasks from various Jobs borrow freely without knowing about what the other Jobs and Tasks are doing and will do?

Thankfully we only need to think about the two resource cases for now:

* Which bowl to take batter from
* Which tray to output waffles to

In no way does these choices have any side-effects for the other Tasks.

If we think about what the limitations could be:

* If we want things outside Tasks and ResourceHandlers scope to determine what resource to use via some logic. Still, that would be solved with an addition to the ResourceHandler since that is its responsibility.
* If we want to reuse the same Equipment given by the resource handler for multiple tasks. Not an issue in this project and can be solved by borrowing by an id.

Ok, we are good to go, Job can trust the Task to get the Equipment it needs.

**Delays between Tasks**

During a job there will be times when the job needs to wait for a certain precondition before it will continue executing the next task. Such a case exists when the lid has been closed on the iron and the batter starts frying, then we want it to wait either a certain amount of time or until a “waffle ready” indicator lights up.

Preferable the iron itself should keep track of how long time it has been frying, and the jobcoordinator should ask each job if they are ready to execute (in a sense it already does by asking priority).

First I thought using a “delay task” was the solution, but when thinking more carefully, it’s more of a characteristic of a existing task to have pre-requisites (such that the iron has been frying for X seconds).

## Information viewpoint

### Positioning

Robot positions in the workspace environment are stored within the controller.

The positions can be used as a clever way of storing global co-ordinates of equipment instead of robot arm and robot tool position (which is the normal).

The way to do this is calling PR (position read) for a specific position. Based on that position, it can be calculated with relative co-ordinates where the robot arm needs to position itself to perform operations on the specific workspace object. Imagine having a music keyboard, it would be a lot easier to only program the origin position of the keyboard and then use relative positions for all the keys instead of adding all the keys as stored positions in the controller. Then the keyboard-object itself would keep track of where all its keys are and thus encapsulate that information and present a nice interface that accepts the name of button as it should be.

## Interaction viewpoint

This section describes how

The waffle irons part in the production of waffles is what will be the bottle-neck of production capacity and is time-critical in terms of waffle-removal to prevent over-frying. Therefore, it is important to be clear about how the OrderHandler and WaffleRecipe organizes its tasks when multiple waffles are in the making and how the “tasks to be done” are stored and triggered.

Events: A Task is completed, A timer reaches zero, waffle finished indicator lights up, A waffle order is made.

Example scenario: Four waffles are ordered, have one single waffle iron and one dual waffle iron that both are empty.

WaffleIron

Available: true/false

Cold, Empty, Frying

Trivial:

1. Get available waffle irons
2. If orderSize >= biggest available waffleIron, deploy task on the biggest waffleIron and set it to not available.
3. Repeat 2 until all orders are placed or we run out of available waffle irons.
4. Let the control of the waffleIron (waffle recipe?) start doing the recipe tasks (frying waffle).
   1. Waffle Cook