Comparison Study

# Comparison Philosophy

To find the most flexible solution to be used for the project integration.

Favouring correct and extensible code.

Using a trade-off study to guide the choice.

Phase A

# Key metrics

**Correctness**

* James’s code hits wall in worlds 2,4,5
* Dan’s code prints extra characters in CSV file and hits wall in world 4.
* Taimoor’s code hits wall in worlds 2,3,4,5

**Modularity**

* James’ and Dan’s code had good separation of classes.
* Taimoor didn’t use any functions or classes.

**Extensibility**

* James’ and Dan’s code is extensible, and more features can be added with small adjustments.
* Taimoor’s code is not very extensible because considerable effort would be required for changes.

# Improvement to solutions

**James**

* Needs more error checking throughout the code.
* Some repetitive switch-case blocks could be avoided.
* Nested control loop is overly complicated. Can be simplified.
* Heading relies on order of path instead of information from the environment. Added complexity.

**Dan**

* Heading relies on order of path instead of information from the environment. Added complexity.
* TaskControl class structure can be made clearer.

**Taimoor**

* Should use more classes and functions to make the code more modular
* Needs more error checking throughout the code.
* Nested statements for CSV can be greatly simplified
* Control loop is overly complicated. Can be simplified.

# Chosen solutions

REPLACE

James’ implementation cleanly separated functionality into small classes which follows open-close principle therefore highly extensible.

However class initialisations cluttered the main function making it harder to read/understand.

Dan’s implementation used a TaskControl which hides class initialisations and greatly simplifies the control loop.

Compare distance sensor vs lidar for wall perception robustness.

Phase B

# Key metrics

* Correctness
* Modularity

# Improvement to solutions

Dan:

* Directed graph (good time and space complexity)
* Very well written CPP code
* Modular
* Solution was not correct

James:

* Floodfill with adjacency matrix
* Modular
* Not CPP styled
* Solution was correct

Taimoor:

* Rigorous floodfill implementation
* Code could be separated into functions
* Solution was not correct

# Chosen solutions

James -> works and can be easily integrated

Phase C

# Key metrics

**Correctness**

* James’ code passes all Test cases (including the Tests used for marking).
* Taimoor’s code doesn’t pass some of the robot heading tests.
* Dan’s solution doesn’t pass some of the robot heading tests. Maze code is incomplete so fails those tests.

**Modularity**

* Dan’s code was the most modular. Had a lot of smaller functions that worked together. Each Task had its own main method.
* James’ code has functions for some tasks but not for all tasks.
* Taimoor’s code didn’t have any functions to make the code modular.

**Efficiency**

* (Add stuff about BFM in the speech)

**Extensibility**

* All members have code that is somewhat extensible, but the degree of extensibility varies based on their specific implementations for certain tasks. Dan’s and James’ code are more extensible than Taimoor’s because they used functions, which means the rest of the code is less likely to break if more features are added.

# Improvement to solutions

Dan

* The robot heading code often gives the opposite heading. Can be fixed.
* Map generation code is incomplete. Can be completed.
* Perspective transform for some mazes results in a black void. Can be fixed.
* Code was designed using wrong version of Python. Python 3.6 can be used to redo it to resolve backwards compatibility issues.

James

* More code can be put inside functions i.e. one function each for horizontal and vertical walls

Taimoor

* Robot heading code can be fixed.
* Target location code can be fixed.
* More functions to avoid repeating code.
* Method of keeping track of Cyan cornerstone is inefficient, can be improved.

# Chosen solutions

* James’ Solution was used for the code but Dan’s style/template was used as the skeleton.

Solution Integration

## RSA Principles

Our integration was vastly improved with a robotic software architecture which foresaw code implementation and separated functionality allowing our code to be highly modular, extensible, and maintainable. To achieve this, we consider two architectural principles:

* Stable Dependency Principle: Elements should only depend on more stable elements.
* Stable Abstraction Principle: Elements should only depend on more abstract elements.

Generally satisfying our principles, our modules should only depend on the interfaces they are implementing.

## RSA Development

To identify our interfaces and their functionality:

1. Identify architectural and functional similarities between all solutions, phases, and extra features we wanted to implement.
2. Convert similarities into interfaces.
3. Breakdown interfaces if necessary to encapsulate a single functionality.

The interfaces that we have identified are:

* Deliberator: Responsible for converting the user’s deliberation into some instruction.
* Localiser: Gets the current position and heading of the robot.
* Mapper: Gets the map as a string.
* PathPlanner: Takes in a graph, pose, heading, and destination and computes a sequence of motions.
* TrajectoryPlanner: Compute the kinematics to achieve a motion.
* MotionPlanner: Compute the motor setpoints from given kinematics.

## A screenshot of a computer Description automatically generated with medium confidence

Because we have identified these interfaces, we can add any kind of extra feature we want on a whim e.g. swapping robots requires an implementation of the MotionPlanner, or using Bang-Bang requires an implementation of the TrajectoryPlanner. These interfaces are also templated, so datatypes can be swapped out based on implementation.

## RSA Implementation & Integration Process

The implementation process is essentially playing with Lego blocks, so we just port our existing code to match the written interfaces.

A picture containing text, monitor, screenshot, black

Description automatically generated

A picture containing text, wall, monitor, black

Description automatically generated

* Phase A:
  + EPuckMotionPlanner
    - Just deals with the kinematics of the EPuck to produce desired motor setpoints.
    - Implements MotionPlanner
  + DeadReckoning
    - From a motion/instruction, computes the trajectory to achieve that.
    - Implements TrajectoryPlanner
  + PathSequencer
    - This is a new class that previous solutions did not have. Having this class increases the software’s extensibility.
  + TaskControl
    - This was significantly improved by reducing the functionality of this class to just contain locks and timers.
  + MotorController
    - Only deals with feeding motor setpoints to Robot class.
* Phase B
  + Grapher
    - Simply converts map string into adjacency list graph
    - This is a new class that previous solutions did not have. Having this class increases the software’s extensibility.
  + BFSDFS
    - Gets graph, destination, initial pose and initial heading and plans a path between initial pose and destination.
    - Complexity is O(V+E)
    - Fixed bug from Dan’s solution where the motion was incorrectly given “LL” to “LLF”.
    - Implements PathPlanner
* Phase C
  + CVProcessor
    - Implements 3 interfaces (Deliberator, Localiser, Mapper)
    - Wraps Python code with Cython and encapsulates Cython function calls in C++ by CVProcessor class.
    - The whole CVProcessor was improved in robustness and flexibility by using the in-built camera in Webots instead of manually-made images (discussed in more detail as an extra feature).
    - Significantly more control over robot’s behaviour.

## Control Loop

As a result of all our hard work with RSA implementation, the effort of defining our robot’s behaviour is trivial. We do this in the control loop which becomes the single point of modification in the program and use a TaskControl class to give us tools to locks and timers to fine-tune our robot’s behaviour.

Diagram

Description automatically generated

And this is our robot in action.

SLIDE FOR INTEGRATION VIDEO & VOICE OVER ABOUT SOLUTIONS CHOSEN

Extension Features

We are putting the mouse in micro-mouse competition by adding a cat.

Functional features:

* Teleoperation
  + This can be toggled by pressing spacebar to switch between teleoperation and autonomous mode.
  + Q = L
  + W = F
  + E = R
* Camera module
  + Takes a snapshot of the world image through a function call.
  + Can now compute location, heading, and destination in middle of control loop.
  + Heading no longer relies on AR tags. Uses rotation from homography matrix to get heading.

Behavioural features:

* Cat and mouse game
  + RSA hardwork lets us define complex behaviour such as cat and mouse.

**Extension Feature Introduction**

For our extension feature, we decided to implement a Cat and Mouse game by introducing a second epuck.

* The first epuck (mouse) has a ladybug on top, and would be teleoperated using manual keyboard inputs.
* The second epuck (cat) will be autonomous, will chase the mouse in the maze.
* Epucks both have bird’s eye view camera modules. The camera is used by the autonomous epuck to snap images of the world and locate the mouse.
* The game ends when both epucks collide with each other

**Overview of Code for Extra Features**

* The teleoperation module used input from the keyboard to move the ‘mouse’ epuck forward, left, or right (can show teleoperation class for this section)
* OpenCV code was adjusted to process higher resolution images from the bird’s eye view camera modules
* (insert bit about how the heading of the cat was calculated by using the modified H matrix and not the AR marker)
* Different controllers for both cat and mouse

**Demo of a ‘Cat and Mouse’ game**

Video recording of the game demo

**Contribution of each team member**

|  |  |
| --- | --- |
| **Team Member** | **Contribution** |
| Dan |  |
| Taimoor |  |
| James |  |

# Introduction

* Title Page
* Member intro

# Phase Comparison

* Phase A
* Phase B
* Phase C

# Improvements

* RSA is the improvement

# Integration

* Run the Integration World (Voice Over)

# Extension Features

Contribution