DESIGN PORTFOLIO

Clark Jeffrey

Vancouver, BC

+1(647)379-9531
cjeffreybda@outlook.com
linkedin.com/in/cjeffreybda

BRIEF

My name is Clark, and I'm an incoming third-year mechanical engineering student at UBC, specialising in mechatronics. I am particularly interested in controls, and more generally the cross-section between mathematics and mechanical, software, and electrical design.

In my free time, I enjoy playing music, writing, and baking. Of these, I've been playing music the longest, having started at the age of seven. I play three instruments – the cello, for which I have my ABRSM Grade 8 certification, the piano, and the bass. I've played with many ensembles, including the Bermuda Philharmonic.

CONTENTS

*

1 MECHANICAL DESIGN

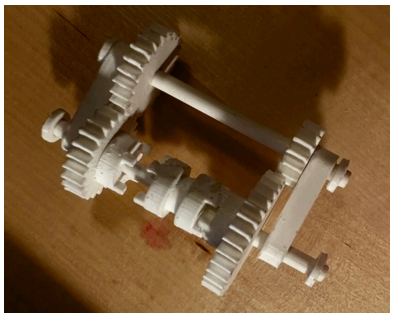
1.1 *Manual transmission – MECH 223*

During the first project of MECH 223, I designed a two-speed manual transmission. It implements a dog gear system, where a servo is used to move a selecting gear between the two output gears. This allowed the user to adjust the torque output of the motor in real time. The process of creating this design involved:

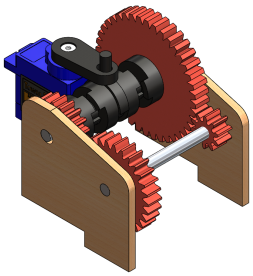
- calculating the torque required to drive the vehicle up the slope, based on predicted mass, friction coeffient, and other factors
- prototyping gear assemblies to assess viability
- recreating accurate gear models in SolidWorks based on physical measurements and gears' proportional relationships
- designing with tolerances and compliancy, allowing for error in parts without comprimising functionality
- stress-testing components in SolidWorks to refine the design and assess viability
- converting user input into mechanical actuation using a servo with a wireless transmitter.

Some challenges I encountered were:

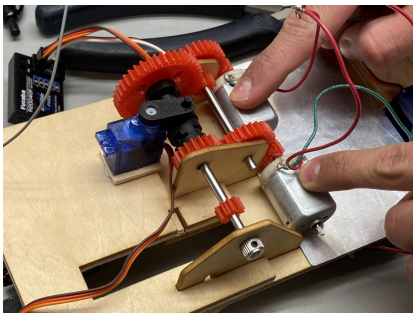
- the available gears didn't have a consistent module, meaning that to have two different ratios, there were two 'ideal' distances, for which a good medium had to be found – the CAD model was used to determine limits
- on prototype tests, binding of the gears was a problem, so to address this, I added a slot to allow the gears to kick back as opposed to binding.



Initial transmission prototype.



Final transmission CAD.



Assembly of final transmission.

2 SOFTWARE DEVELOPMENT

2.1 *Chess engine*

In my free time, I'm developing a chess engine with C++. Due to the rapidly growing number of positions that the computer has to evaluate, the key problem is optimisation. Over the course of this project, I've continually been making optimisations, to the point where now the computer is able to evaluate over 200,000 positions in less than a second, and it's continually improving. Some of the optimisations I've made are:

- passing by reference whenever possible to avoid making expensive copies
- using structs when passing related variables to allow them to be loaded next to each other in cache
- using an alpha-beta pruned search to avoid wasting time computing evaluations for worse positions
- using a position hash map to quickly look up scores of previously-evaluated positions
- using bitboards of piece locations and attacks to avoid expensive loops.

Some challenges I have encountered are:

- small overcounting errors in the move-counting algorithm drastically increase runtime, and have to be debugged by comparing to known move counts determined by Stockfish
- ordering moves by likeliness to be good will improve speed, but without running expensive checks, may result in double-counting many moves – a tradeoff that has to be carefully balanced
- small tweaks to the evaluation algorithm can drastically change which moves are considered good, and as such, evaluation has to be extensively tested
- as evaluation is run for every single move, any calculations performed must be as fast as possible, so as much as possible needs to be bitwise.

An in-progress game against the engine (black). The GUI was created programmatically using the wxWidgets library.

```
std::array<std::variant<int, float>, 3> ChessEngine::getBestMove(BoardProperties& props, bool isWhite, int depth, float alpha, float beta, const
  std::chrono::steady_clock::time_point& startTime, int ms, const std::vector<std::array<int, 2>>& priorityMoves)
{
    if (depth == 0)
    {
        return { 0, 0, ChessEngine::quiesce(props, isWhite, alpha, beta, startTime, ms) }; // exhaustively evaluates captures to avoid blundering
    }

    std::vector<std::array<int, 2>> moves = ChessEngine::getAllLegalMoves(props, isWhite);

    if (priorityMoves.size() > 0)
    {
        moves.insert(moves.begin(), priorityMoves.begin(), priorityMoves.end());
    }

    std::array<int, 2> bestMove = {};

    for (auto& move : moves)
    {
        auto stop = std::chrono::high_resolution_clock::now();
        auto duration = std::chrono::duration_cast<std::chrono::milliseconds>(stop - startTime);

        if (duration.count() >= ms && depth != 1) // avoid returning too early on depth 1 and getting an illegal move
        {
            return {0, 0, (float) 0};
        }

        char captured = ChessEngine::makeMove(props, move);

        auto opponentsMove = ChessEngine::getBestMove(props, !isWhite, depth - 1, -beta, -alpha, startTime, ms, {});
        float eval = -std::get<float>(opponentsMove[2]);

        ChessEngine::unmakeMove(props, move, captured);

        if (eval >= beta)
        {
            return { 0, 0, beta };
        }
        if (eval > alpha)
        {
            alpha = eval; // alpha acts like max in MiniMax
            bestMove = move;
        }
    }

    return { bestMove[0], bestMove[1], alpha };
}
```

The recursive function for searching moves.

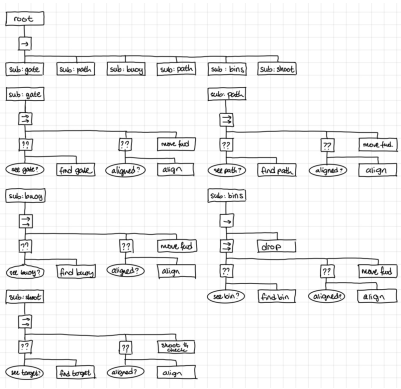
2.2 *Mission planner – UBC Subbots*

My primary project on the UBC Subbots software team is developing a 'mission planner' which coordinates the autonomous underwater vehicle's (AUV's) actions. This package will serve as the backbone to the AUV's internal logic, interpreting data from sensors, and using them to inform decisions. The process of developing this package has involved:

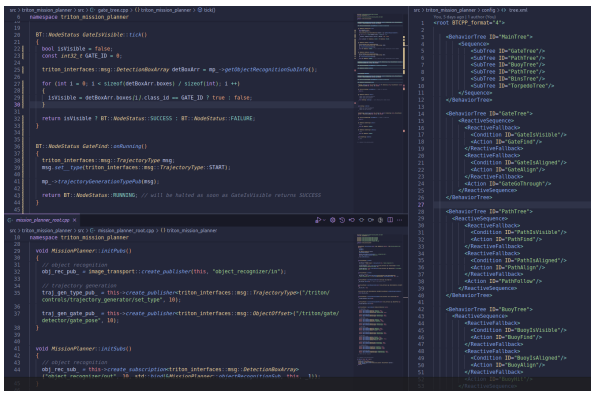
- designing a tree architecture that will allow for conditions and actions to be run in parallel, facilitating continual adjustment as the AUV receives new information
- implementing and integrating the BehaviorTree nodes with ROS 2 nodes, allowing for communication across established ROS topics
- handling and interpreting the data sent by other nodes
- use of C++, Python, and XML.

Some challenges I've encountered are:

- both the main 'handling' ROS node and the smaller BT nodes rely on each other's function definitions, and so I had to implement forward declarations to prevent recursive references
- the AUV has to be able to adjust if any previous tests suddenly fail, so a tree architecture had to be carefully designed to incorporate these tests continually running in parallel to the other actions.



Initial sketch of behaviour tree architecture.



Small code snippet of implemented behaviour tree.

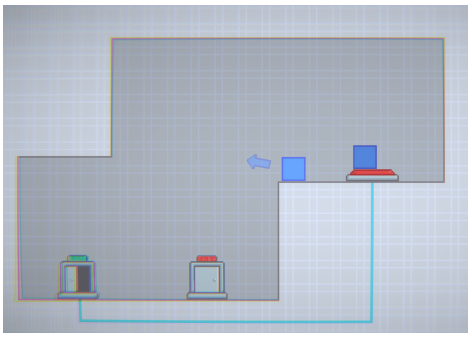
2.3 *Unity video game*

As a hobby project, I developed a short video game in Unity. The premise was a puzzle game where you can eject duplicate boxes in order to propel yourself, or activate switches. I chose this combination of mechanics to give me a space to experiment with both real-time physics simulations, and logic-based object interactions. The process of developing this game involved:

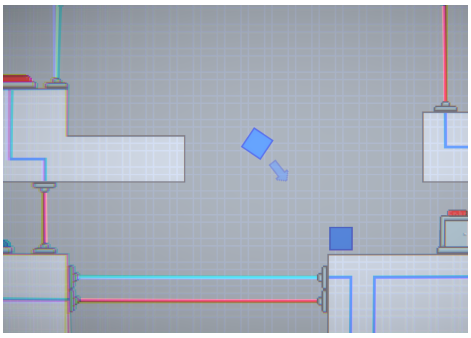
- writing in-engine physics simulations for calculating player movement and interactions with the environment
- creating a solid class architecture, to make level design and object interactions simple to implement
- use of C#, Unity, and Java.

Some challenges I encountered were:

- physics processes in Unity run on a different time scale than the rest of the game, which has to be taken into account when writing custom physics simulations
- collision detection and correction has to be implemented carefully, to ensure that no matter when the game updates, the player never leaves the play area.



Level showcasing simple switch behaviour.



Level showcasing many object interactions.

3 METALWORKING

3.1 *Metal-cast key*

In my final two years of high school, my friend and I began working on a book together, in which metal keys with crystals inlaid in the handle played an important role. To commemorate one year of working on this project together, I wanted to give my friend a real-life version of this key. The process of working on this project involved:

- modelling the key in Blender, both to experiment with size and shape, and to see how the final key might look using Blender's rendering abilities
- creating a DXF file of the key's profile, to prepare it for laser-cutting
- laser-cutting the key out of acrylic of the desired size
- using plaster of paris to make a mould out of the acrylic key
- melting and casting metal in the mould, and subsequently cooling the cast slowly using an oven, in order to minimise brittleness
- correcting the final key with a hacksaw and a range of files.

Some challenges I encountered were:

- drastic temperature changes would cause the mould to crack, so care had to be taken at every stage, including pre-heating the mould, and bringing the temperature back down very slowly
- small interior peices of the mould were not strong enough, and metal that filled these areas had to be removed by hand.

Completed key.

4 WOODWORKING

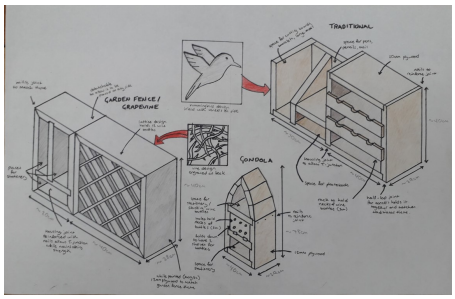
4.1 *Kitchen stationery organiser – GCSE Design Technology*

For my final project in my GCSE Design and Technology course, I designed, modelled, and built a kitchen organiser. Over the course of the project, I gained experience with the entire design process, from setting specifications, to concept generation and evaluation, client consultation, and detailed design. The process of completing this project involved:

- designing and constructing a physical product incorporating continuous client consultation
- modelling the design in Blender to create a realistic image, for the purpose of collecting client feedback
- prototyping aspects of the design to assess viability
- manufacturing and assembling parts with the use of laser cutters, hand tools, and workshop machinery.

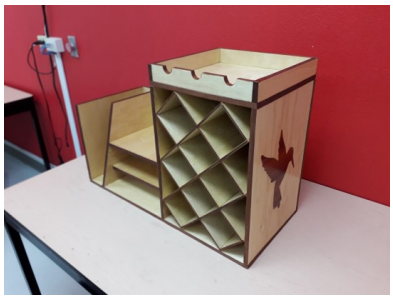
Some challenges I encountered were:

- the shelves had to be mounted between two faces, neither of which had an accessible back for drilling – instead, I used a push-pin inserted in the dowel hole of the shelf to mark where I had to drill on the facing sides
- the organiser edges had to be aligned exactly to hold the lattice design securely – blocks were used while clamping to ensure a 90° angle.



Initial concept sketches.

Final design rendered in Blender.



Completed organiser.

4.2 *Chess board*

For my grandfather's birthday, I wanted to make him a hand-made chess board, as we would often play chess together when I saw him. To do so, I used woodworking techniques that I'd developed in my high school GCSE Design and Technology course, including:

- creating a CAD model to determine appropriate dimensions and construction techniques
- using a bandsaw to precisely cut wood strips
- using a mitre saw to cut decorative edges for the sides of the board
- evenly varnishing the final product in multiple coats to both protect the material, and make it more appealing.

Some challenges I encountered were:

- small errors in cuts led to unevenness in square sizes, which had to be corrected by hand
- some strips would shift under the clamping pressure required to join them, so a custom jig had to be made to keep it in place.

Final assembled board.

Completed board, after varnishing.