Interactive Robot Chess

CS39440 Major Project Report

Author: Your Name ([ibo1@aber.ac.uk](mailto:ibo1@aber.ac.uk))

Supervisor: Dr/Prof. My Supervisor (supervisorid@aber.ac.uk)

28th February 2021

Version 1.0 (Draft)

This report is submitted as partial fulfilment of a BSc degree in  
Computer Science (G400)

Department of Computer Science

Aberystwyth University

Aberystwyth

Ceredigion

SY23 3DB

Wales, UK

Declaration of originality

I confirm that:

* This submission is my own work, except where clearly indicated.
* I understand that there are severe penalties for Unacceptable Academic Practice, which can lead to loss of marks or even the withholding of a degree.
* I have read the regulations on Unacceptable Academic Practice from the University’s Academic Registry (AR) and the relevant sections of the current Student Handbook of the Department of Computer Science.
* In submitting this work, I understand and agree to abide by the University’s regulations governing these issues.

Name …………………………………………

Date ……………………………………………

Consent to share this work

By including my name below, I hereby agree to this project's report and technical work being made available to other students and academic staff of the Aberystwyth Computer Science Department.

Name …………………………………………

Date ……………………………………………

Acknowledgements

I am grateful to…

I’d like to thank…

Abstract

Include an abstract for your project. This should be approximately 300 words.

The abstract is an overview of the work you have done. Highlight the purpose of the work and the key outcomes of the work.

Contents

[1. Background 8](#_Toc134081669)

[1.1. Introduction 8](#_Toc134081670)

[1.2. Robotic Chess systems 8](#_Toc134081671)

[1.3. Game of Chess 9](#_Toc134081672)

[2. Analysis 10](#_Toc134081673)

[2.1. Introduction 10](#_Toc134081674)

[2.2. Interface 10](#_Toc134081675)

[2.2.1. Bluetooth 10](#_Toc134081676)

[2.2.2. Wired connections 11](#_Toc134081677)

[2.2.3. Internet 11](#_Toc134081678)

[2.2.4. Conclusion 11](#_Toc134081679)

[2.3. Robotic System 12](#_Toc134081680)

[2.3.1. Which robot arm to use? 12](#_Toc134081681)

[2.3.2. How to detect the chess pieces? 12](#_Toc134081682)

[2.3.3. Creating a chess engine or using existing once? 13](#_Toc134081683)

[2.3.4. How to deal with possible security issues? 13](#_Toc134081684)

[2.3.5. Which programming language to use? 13](#_Toc134081685)

[2.4. Game of Chess 13](#_Toc134081686)

[2.5. No upfront prototyping 13](#_Toc134081687)

[2.6. Project Objectives 14](#_Toc134081688)

[2.6.1. User Stories 14](#_Toc134081689)

[2.6.2. Functional Requirements 16](#_Toc134081690)

[2.6.3. Technical Objectives 17](#_Toc134081691)

[3. Process 18](#_Toc134081692)

[3.1. What is expected of the development process 18](#_Toc134081693)

[3.2. Chosen development process 18](#_Toc134081694)

[3.3. Implementation of Scrum 19](#_Toc134081695)

[3.3.1. Product Backlog and Sprint Backlog 19](#_Toc134081696)

[3.4. Version Control System 19](#_Toc134081697)

[4. Design & Implementation 20](#_Toc134081698)

[4.1. Introduction 20](#_Toc134081699)

[4.2. Overall Architecture 20](#_Toc134081700)

[4.2.1. Communication Protocol 20](#_Toc134081701)

[4.3. ROS Robotic System Overall Architecture 21](#_Toc134081702)

[4.3.1. Introduction 21](#_Toc134081703)

[4.3.2. Preliminary concept to detect chess pieces 21](#_Toc134081704)

[4.3.3. ROS Robotic System Initial Architecture 23](#_Toc134081705)

[4.3.4. Current ROS Robotic System Architecture 25](#_Toc134081706)

[4.4. ROS Robotic System Components 27](#_Toc134081707)

[4.4.1. Introduction 27](#_Toc134081708)

[4.4.2. Communication API 27](#_Toc134081709)

[4.4.3. Chess Engine Wrapper 27](#_Toc134081710)

[4.4.4. System State Machine 29](#_Toc134081711)

[4.4.5. Robot Arm 33](#_Toc134081712)

[4.4.6. Chess piece detection 38](#_Toc134081713)

[4.5. User Interface 38](#_Toc134081714)

[4.6. Other Relevant Sections 38](#_Toc134081715)

[5. Implementation 39](#_Toc134081716)

[6. Testing 40](#_Toc134081717)

[6.1. Overall Approach to Testing 40](#_Toc134081718)

[6.2. Automated Testing 40](#_Toc134081719)

[6.2.1. Unit Tests 40](#_Toc134081720)

[6.2.2. User Interface Testing 40](#_Toc134081721)

[6.2.3. Stress Testing 40](#_Toc134081722)

[6.2.4. Other Types of Testing 41](#_Toc134081723)

[6.3. Integration Testing 41](#_Toc134081724)

[6.4. User Testing 41](#_Toc134081725)

[7. Critical Evaluation 42](#_Toc134081726)

[8. Figures 43](#_Toc134081727)

[9. References 44](#_Toc134081728)

[10. Appendices 47](#_Toc134081729)

[A. Third-Party Code and Libraries 48](#_Toc134081730)

[1. Code 48](#_Toc134081731)

[a) C++ Socket Creation on Linux 48](#_Toc134081732)

[b) C++ bytes to integer conversion 48](#_Toc134081733)

[c) C++ ChessBoard class 48](#_Toc134081734)

[d) C++ SubProcessHandler class 48](#_Toc134081735)

[B. Code Samples 49](#_Toc134081736)

[C. Communication Protocol 50](#_Toc134081737)

[D. Internal Communication Protocol 52](#_Toc134081738)

# Background

## Introduction

For the MMP a robotic or machine learning themed topic had to be chosen. The proposed topics by the university did not peak enough of an interest, that a decision was made to propose a topic. Previous experience engaging in 6/7dof robot arm systems, and the natural interest and enjoyment of working with such robotic systems that interact with the human environment, led to the decision to theme the topic around a 6/7dof robotic arm systems.

Robotic arm systems are widely used in the industry ranging from NASA’s Curiosity Rover [1] for exoplanetary exploration to ABB’s [2] automotive robotic systems for painting vehicles in the automotive manufacturing industry. The theme of the project had a wide range of possible solutions.

## Robotic Chess systems

An initial investigation into robotic systems in the game of chess, due to an inherent interest in the game of chess, lead to a rich history dating back to the 18th century when Wolfgang von Kempelen's famous "Turk", also known as The Turk or Automaton Chess Player, chess machine wowed audiences with its ability to play chess against humans. The “Turk” was a mechanical illusion that hid a human chess master inside of it to operate it. [3]

The first true chess automation would be created in 1912 by a Spanish engineer named Leonardo Torres y Quevedo. This early fascination in creating physical world automated chess systems wouldn't last long. Due to the fact that the last substantial and ground-breaking advancement into a physical automated chess system was made in 1983. In the current day and age most automated chess systems come from DIY(Do-It-Yourself) developers. [4]

This most likely resulted from the introduction and development of chess engines, notably IBM’s Deep Blue [5] chess engine. The rise of chess engines in the 20th century marked a significant turning point in the development of automated chess. Unlike traditional chess machines, which relied on a mechanical arm to move pieces, chess engines use sophisticated software algorithms to analyse positions and make moves. This allows them to play at a level that far surpasses any human or mechanical opponent. The advent of computer technology and the development of powerful algorithms, combined with the ability to easily update and improve the software, has made chess engines the dominant force in automated chess. Today, these engines are used not only to play chess but also to analyse games, teach players, and even compete against each other in specialized tournaments.

The inherent difficulty in creating a new robotic chess system and the significant interest in developing one led to focus of the project on to this area. During the investigation, a newspaper article by The Guardian that reported an incident in which a chess robot reportedly grabbed and broke the finger of a seven-year-old opponent [6] was found.

The natural complexity and variety that can be achieved with robotic chess systems is vast, ranging from abstract chess pieces with markings in a controlled environment, such as the ROS Final Project [7] created in ROS [8], to an all-inclusive Chess-Robot [9] project that use a combination of hardware and software technologies. For instance, the Chess-Robot project is implemented using a Raspberry Pi and a user interface written in Python. However, with this complexity also come challenges, such as ensuring the safety of participants and preventing potential malfunctions that could cause harm to people or damage to property, as seen mentioned in the guardian newspaper article above.

## Game of Chess

Chess is a two-player board game played on a board with 64 squares of alternating colours.

Although the origin of chess is a topic of debate, it is commonly believed to have developed from the Indian game of chaturanga. This game had two key features that later chess variants retained: varied powers of different pieces and the victory objective of protecting one piece, the king. Over time, chaturanga evolved into shatranj, a two-player game that travelled east, north, and west, taking on different characteristics in each area. Chess eventually reached Europe through Persia, the Byzantine Empire, and the expanding Arabian empire. The modern rules and appearance of chess pieces gradually evolved with a lot of regional variation until the 15th century, when two essential rule changes became widespread: the counsellor changed gender to become the queen, and the bishop acquired increased mobility. These changes added a dynamic new element to chess and made checkmates more frequent. Although castling and the en passant capture rules were also added, it took them longer to become widely accepted. [10]

The board is labelled vertically from a to h. Horizontally, the board is labelled from 1 to 8. Each player starts with sixteen pieces, including a king, a queen, two rooks, two knights, two bishops, and eight pawns. The objective is to checkmate the opponent's king, putting it in a position where it is under attack (in check) and there is no way to avoid capture on the next move. Each player takes turns to make a move, which involves selecting one of their pieces and moving it to a new square on the board according to its individual movement rules. Chess requires strategic thinking, careful planning, and the ability to anticipate and respond to the opponent's moves. In some cases, the game may end in a draw, which can occur if neither player is able to checkmate the opponent's king or if both players agree to a draw. The game has a rich history and continues to be popular among players of all ages and skill levels. [10]

# Analysis

## Introduction

To design a robotic chess system, it was essential to identify its fundamental parts. Two possible strategies were considered for building the system.

The first approach involved using only a robotic arm, which would make the information flow within the system problematic. It would be challenging to provide the user with accurate information about the system's state and offer viable solutions for errors. Additionally, it would be difficult to initiate a chess game, stop the game, report back on illegal moves made by the human player or change settings for the chess engine.

The alternative strategy involves creating a system with a robotic arm and a user interface for users, aka human chess players. This approach would simplify information access for users and make it easier to control specific features of the system, such as setting the AI player's difficulty level and above-mentioned examples. Moreover, such a system would enable developers to create modules that provide easy access to and control over specific functions, specifically for developers.

Based on experience working with robotic arms, both in university and industry, combined with the above-stated projects, the decision was made to develop a system with a user interface. However, to ensure that the project could be completed within the allotted time limit, the interface would be designed as a prototype rather than a fully-fledged software solution. The interface would serve as a helpful module for the entire system.

## Interface

For a robotic chess system to be effective, users must be able to communicate with it through an interface and gather information from it. The interface can be made available to human players either by running it on the same hardware as the robotic system or by using an external device. With an external device interface, users can utilise their smartphones, computers, or other hardware systems, making the interaction environment more flexible and user-friendly.

There are several possible solutions for achieving this, including technologies such as Bluetooth [10], wired connections, or the internet. Each of these technologies has its own advantages and disadvantages

### Bluetooth

Bluetooth is a low power wireless radio connection that streams data over 79 channels in the 2.4GHz unlicensed industrial, scientific and medical frequency band [11]. It is widely used in hardware and is growing market [12] in ways to link devices that need to interact with one another. Bluetooth has the main downside of still being unreliable, due its inherent radio frequency band being part of the industrial, scientific and medical zone of the RF spectrum [13].

### Wired connections

Wired connections utilizing protocols such as Recommended Standard 232(RS-232) [14], Universal Serial Bus (USB) [15] and others have a major upside of being reliable. As they are a direct physical connection between two systems. The issue with wired connections is that, in general, mobile devices such as smartphones, tablets, and other portable electronics aren't designed to communicate with other systems via USB, which is the primary wired connection they support. Further a direct connection to the system would be needed constantly making it more inconvenient for the user to use.

### Internet

By using the internet for communication, it is possible to link some devices wirelessly or directly over a wired connection. Because of the environment, devices that favour wireless connections or do not support direct connections can now achieve both. The internet is widely utilised, and practically every device that may be used for communication or interfacing have built-in or easily expanded functionality for using the internet.

Further the internet provides a multitude of protocols ranging from BGP, HTTP, UDP, TCP all the way to IPsec [16].

### Conclusion

After conducting research on the possible technologies for interfacing with the robotic chess system, it was found that internet provided the best solution, offering the benefits of both direct and wireless connections for any device. However, a decision had to be made on which protocol to use, with the two main contenders being TCP and UDP.

While UDP is faster, it has the downside of potential package loss, making it less reliable for the system's requirements. As reliability is crucial for the system to function effectively, TCP was chosen as the protocol. [17]

An interest in networking and learning how to achieve this led to not exploring any 3rd party libraries as possible solutions.

Ensuring data security is paramount, and sending unencrypted data may pose a significant security risk. However, as data encryption falls outside the scope of the present project, it was deemed appropriate to defer this aspect to a future iteration of the system.

To create an interface that could be run on any kind of hardware, a software language that has high compatibility with different hardware had to be chosen.

Python [18], Java [19] and Kotlin [20] all enable the running and building of their respective compilers in all major desktop environments and android. Experience in creating user interfaces in JavaFX [21], which uses Java as a base, was the deciding factor for developing in Java.

## Robotic System

The robotic system that enables the user to play chess could be achieved in a multitude of solutions. Following problems had to be addressed:

* Which robot arm to use?
* How to detect the chess pieces?
* Creating a chess engine or using existing once?
* How to deal with possible security issues?
* Allow different system states?
* Which programming language to choose?

### Which robot arm to use?

It was of interest to develop a system that could work on a multitude of robot arms with little redevelopment of the system. To achieve this a robotic middleware would suite this interest. Robotic Operating System (ROS) [8] is such a robotic middleware. Moreover, experience in developing systems in ROS made it the only plausible solution while limiting the scope of the project.

To choose which robotic arm to use for the system the university was consulted. As there was a possibility that the developed system could be run on an actual robotic arm from the university. The university provided following two possible robot arms:

* Universal Robot UR3e [22]
* Kinova GEN3 [23]

Both robot arm specifications were investigated. As both are more than capable of achieving its goals in the system. The Kinova GEN3 was chosen due to its possibility of attaching a camera to the robotic arm for chess piece detection.

Creating a system that can function efficiently in a simulated environment is crucial, as there may be instances where running the system on an actual robotic platform may not be feasible.

In order to meet this requirement, ROS Gazebo [24] will be utilized, which enables the creation of a simulated environment for the chess board, chess pieces, and robotic arm.

### How to detect the chess pieces?

The initial investigated projects that are comparable to this project showed different ways of detecting the chess pieces. As the idea for this system is supposed to play an actual game of chess it is not feasible to create an abstraction for the chess pieces.

Detecting chess pieces on their own is quite difficult. This is a consequence of chess pieces blocking the view of others and lightning and shadows having further impact on the detection of chess pieces [25]. It would be necessary to create a module to make this challenge simpler in light of those issues and the possibility that spotting chess pieces on a board is an MMP project in and of itself.

Creating a module to allow for flexible recognition of chess pieces would be useful for future upgrades to the system.

### Creating a chess engine or using existing once?

Developing a chess engine requires a significant amount of work. To achieve success in this project, it was necessary to use a well-established chess engine. Therefore, an existing chess engine was selected to reduce the scope of the project. While one option was to create a module that uses a specific chess engine, it was an intriguing idea to explore the possibility of building a wrapper around the chess engine to play chess against a wide range of various chess engines. In order to determine the feasibility of this idea, research was conducted to investigate how chess engines interact with the outside world and whether there is a standard protocol for doing so.

During the investigation, it was discovered that the Universal Chess Interface (UCI) designed by Rudolf Huber and Stefan Meyer-Kahlen could enable such interfacing [26]. Many chess engines, such as Stockfish [27], Komodo [28] and others, utilize this protocol. This indicated that a plug and play system that could utilize different chess engines was possible.

### How to deal with possible security issues?

As previously noted in the background section, a newspaper article highlighted an incident where a 7-year-old chess player was injured by a robotic arm, underscoring the significant safety risks involved in such a system. In order to ensure the safety and well-being of any human interacting with the robotic system, object detection combined with safety shutdown features would be necessary. However, implementing such a system would considerably increase the scope of the project, and as a result, it was decided not to incorporate it in the current version. However, this critical safety feature would be taken into account during the design of the overall system, and it could be incorporated in future versions of the project.

### Which programming language to use?

Python [18] and C++ [29] development is both supported by ROS [22]. The University provided experience in developing Python on the ROS middleware. Given the prior knowledge, it was obvious to select Python as the primary programming language for the robotic system. Even though Python offered more expertise and experience, the choice to use C++ for this project was motivated by a desire to advance one's C++ skills.

## Game of Chess

To limit the scope of the project the basic variation of chess will be implemented. En pesante will not be taken into consideration for this project.

## No upfront prototyping

It was decided not to prototype any of the aforementioned modules that will constitute the system. This decision was made due to the intrinsic requirement of dealing with various new software, hardware, and coding methodologies, such as networking, among others. The development process should enable the acquisition and integration of new techniques while progressing the project simultaniously.

## Project Objectives

The preceding sections have outlined the various methods for creating the proposed system and the chosen approaches. Drawing upon these methods, user stories were generated and presented in section 2.4.1. As a result, a series of requirements have been developed to break down the high-level user stories into specific requirements.

### User Stories

The succeeding user stories have been created with the help of Figma [30]. In order to present the user stories, it is necessary to first define the different types of users involved in the system.

#### Types of Users

The system will consider 3 types of Users:

* General User
  + A General User is not defined at the point of interaction and cannot be classified as any of the following user definitions.
* User
  + A User is a user that is interacting with the system user interface and has access to the user functions of the system.
* Admin (AKA Developers)
  + An Admin is a user that has access to backend information through the user interface in addition to interacting with the system in the same way a User does.

#### User Stories for a general user

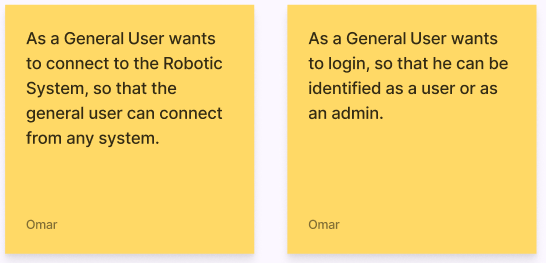


Figure 2.1 General User- User Stories

#### User Stories for User and Admin

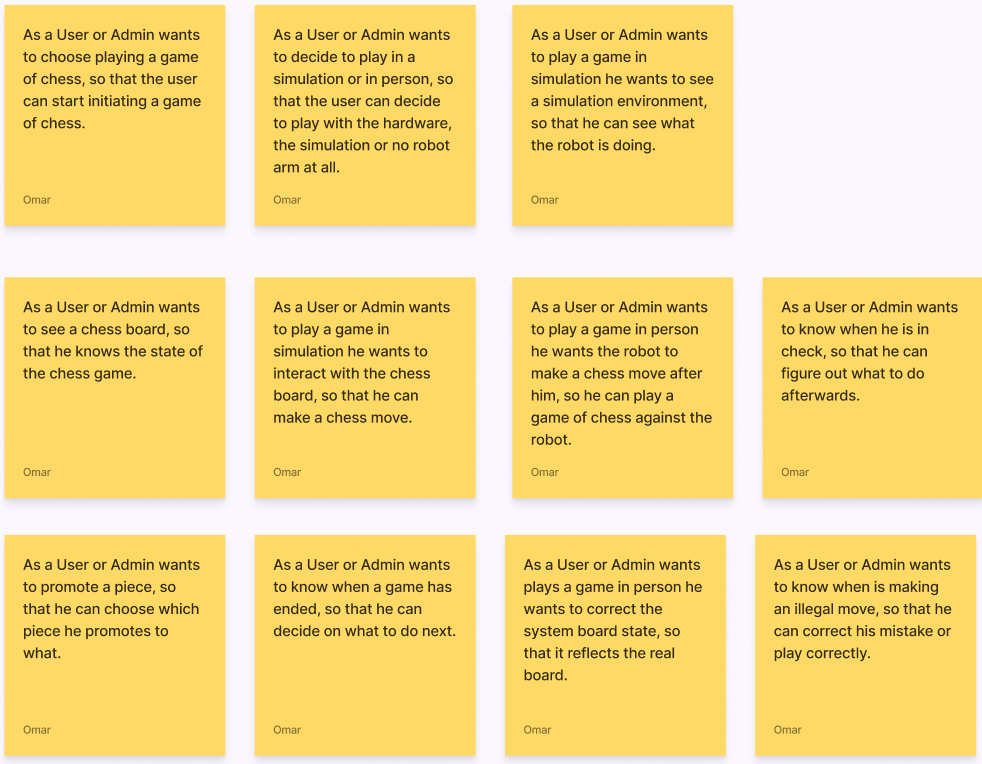


Figure 2.2 User and Admin - User Stories

#### User Stories for Admin

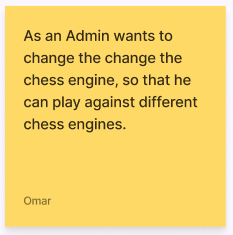


Figure 2.3 Admin - User Stories

### Functional Requirements

The functional requirements that have been identified and describe the expected interaction of a user with the system.

#### FR1 Connect to the system

A General User connects to the system via the internet, through a wired or wireless connection from different systems.

#### FR2 Loging in

A Gener User logs in to be identified as a User or Admin.

#### FR3 Start system in simulation

A User or Admin can set the system into a simulation state. The User or Admin will be able to view the simulation world with its robot arm, chess board and chess pieces.

#### FR3.1 Initiate a game of chess in simulation

A User or Admin can initiate a game of chess in the simulation state of the system and is able to choose which chess engine to play against.

#### FR3.2 Making a player move in simulation

A User or Admin will make a player move through the UI interface when playing in simulation. The User or Admin wants to be notified if an illegal move occurred.

#### FR3.3 Feedback of the chess board in simulation

A User or Admin wants to be prompted with a visual representation of the chess board on the user interface.

#### FR3.4 Feedback of the system

A User or Admin wants to be prompted with feedback of various types, such as an illegal move, if his king is in check and more.

#### FR4 Start system in person state

A User or Admin can set the system into a in person state.

#### FR4.1 Initiate a game of chess in person

A User or Admin can initiate a game of chess in the person state of the system. The User or Admin is able to choose which chess engine to play against.

#### FR4.2 Making a player move in person

A User or Admin wants the robot arm to make a chess move after the User or Admin made a chess move. The User or Admin wants to be notified if an illegal move occurred.

#### FR4.3 Feedback of the chess board in person

A User or Admin wants to be prompted with a visual representation of the chess board on the user interface. This enables the user to identify possible problems with the state of the robotic system, if the internal representation does not match the real-world game state.

#### FR4.4 General feedback of the system

A User or Admin wants to be prompted with general feedback of various types, such as an illegal move, if his king is in check or more, on the user interface.

#### FR5 Information about game end

A User or Admin wants to be prompted the end of a chess game, if the User or Admins king or the chess engines king is in check mate.

#### FR6 Changing the types of chess engines available

As an Admin you should be able to chess the possible chess engines that are available on the system.

### Technical Objectives

#### TO1 Opening a TCP connection

Opening a TCP connection between the user interface and the system will be required.

#### TO2 Interfacing with a multitude of chess engines

It will be important to interface with a multitude of chess engines.

#### TO3 Provide a way of switching system states

It will be important to switch between simulation and in person states. This will include different ways of handling how information is processed and chess moves are executed.

#### TO4 Identification of chess pieces

As mentioned earlier on it will be important to provide a module that does not consume a lot of work to create, to identify the chess pieces and the chess board state.

#### TO5 Modular system design approach

To ensure that upgrading or modifying specific components of the system does not require a complete redevelopment, it is crucial to construct the system using a modular approach. This is particularly essential for the user interface, which serves only as a prototype and must be readily replaceable for different devices or in the event of a redesign.

#### TO4 Documentation of the code

Given the modularity of the system and the likelihood of future expansions or adaptations, it is imperative that the code is adequately documented.

# Process

## What is expected of the development process

The development process is expected to meet following requirements:

* The ability to deliver functional software on a frequent basis
* The capacity to accommodate research on specific subjects
* The flexibility to adjust to evolving requirements, even during the late stages of development
* The use of functional software as the primary measure of progress.

The mentioned requirements are rooted in the necessity to obtain technical competencies, including the ability to network between systems, master C++ programming, and work with unfamiliar hardware, such as the Kinova GEN3. This decision was touched upon in section 2.5, where it was determined that upfront prototyping would not be included.

## Chosen development process

Agile methodologies, particularly Scrum, are well-suited for projects that require adaptability and flexibility. The ability to deliver working software frequently and adapt to changing requirements, even late in development, aligns with the needs of the process outlined in the above-mentioned section. The iterative approach of Scrum provides the opportunity to test and validate new ideas and allows for revision of the implemented ideas, which ultimately results in a better product. [31]

As Scrum is a lightweight framework that helps teams generate value through adaptive solutions for complex problems. It requires a Scrum Master, Product Owner, and Scrum Team who work together to turn a selection of work into an increment of value during a Sprint. The Scrum Team and its stakeholders then inspect the results and adjust for the next Sprint. Scrum is purposefully incomplete, only defining the parts required to implement Scrum theory, and can be adapted to work with existing practices. The rules of Scrum guide relationships and interactions, making visible the relative efficacy of current management, environment, and work techniques so that improvements can be made. [31]

As Scrum is purposefully built as an incomplete framework, it necessitated the need to define and explain each constituent part of Scrum and how it is being applied. It should be noted that Scrum is primarily designed for teams, whereas the present project is a solo endeavour, thus necessitating modifications and adaptations.

## Implementation of Scrum

Due to the solitary nature of the project, several team-oriented features of the Scrum framework, including the roles of the Product Owner and Scrum Master, have been omitted. Instead, the creation of backlog items in accordance with Scrum methodology was conducted in consultation with the project supervisor, either in advance or during the planning stage. The daily progress reviews were conducted on a less frequent basis, occurring every second to third day. Furthermore, the Sprint reviews were carried out in collaboration with the project supervisor. Sprint planning was undertaken after each Sprint and presented to the supervisor for feedback on areas that could be improved.

The sprints were flexible in terms of duration, within the constraints of a minimum of one week and a maximum of three weeks.

### Product Backlog and Sprint Backlog

In Scrum, the Product Backlog is an ordered list of all the work that needs to be done to complete a project and the Sprint Backlog is a selection of items from the Product Backlog for that Sprint [31].

The representation of the Product Backlog and Sprint Backlog offers a great deal of flexibility. As a result, Kanban [32] has been selected as the preferred approach, where the user stories are populated on the Kanban board.

For the Sprint Backlog specifically the high-level User Stories were split into Spikes and Tasks. Spikes referring to research needed to be done and Tasks referring to work that needed to be done.

The Sprint Backlog was split into 4 sections:

1. Sprint Backlog
   1. Holding the User Stories from the Product Backlog to be finished in this Sprint.
2. To Do (Tasks or Spikes)
   1. Holding the created Tasks and Spikes that still needed to be done.
3. In Process (Tasks or Spikes)
   1. Holding the Tasks and Spikes that were worked at in the moment.
4. Awaiting Review (Tasks or Spikes)
   1. Holding the Tasks and Spikes that were waiting for review and acceptance to be completed.
5. Done (Tasks or Spikes)
   1. Holding the Tasks and Spikes that were finished in this Sprint.

## Version Control System

A version control system (VCS) was utilized in the project to hold the Product and Sprint Backlog and enable the branching for Sprints to not affect the main state of the project it certain features are finished. For the VCS GitHub [33] was used. The Product and Sprint Backlog were represented as GitHub Projects [34].

# Design & Implementation

## Introduction

As elaborated in Chapter 3, the project underwent an iterative research and development process, resulting in the evolution of the design over the course of the development cycle. To provide a comprehensive account of the project's progress and current implementation, the chapter will be structured as follows:

* Firstly, an overview of the overall architecture will be presented, which remained unaltered throughout the project.
* Subsequently, the evolution of the robotic architecture will be discussed, including its initial and current state, with an explanation of the factors that led to its current configuration.
* Following that, a detailed description of the key robotic system elements will be provided.
* Next, the User Interface architecture will be examined.
* Finally, the most significant user interface modules will be inspected in detail.

## Overall Architecture

In order to ensure smooth development of the system from an architectural standpoint, it was crucial to formulate an initial concept of the overall architecture and establish how communication between the user interface and the ROS robotic system would be realized.

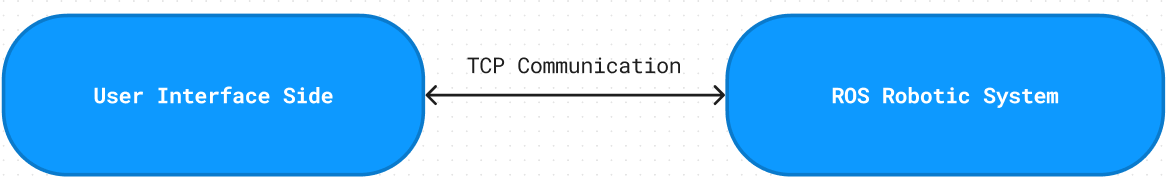


Figure 4.1 User Interface - ROS Robotic System Communication

Figure 4 illustrates the fundamental communication link between the user interface and the ROS robotic system. The implementation employs a TCP protocol agreed upon by both systems. This entailed adhering to the requirement for communication based on an Internet TCP protocol, as previously established in the Analysis chapter. The utilization of such a protocol aligns with the goal of enabling communication with various external devices, including but not limited to laptops, mobile phones, and tablets.

### Communication Protocol

In the early stages of the project, it was necessary to establish the foundation of the communication protocol, given the uncertainties regarding the amount and type of data that would be transmitted. Moreover, to guarantee the system's future-proofing, a flexible protocol was deemed necessary. As such, the following solution was developed:

#### Definition

The communication protocol adopted for the project is composed of two entities: a client and a server. While the client initiates the requests, the server is responsible for responding to these requests. It is mandatory that the server be configured on the robotic system, and the client be configured on the user interface system.

The communication protocol that has been established comprises three components:

* cmd:
  + Size: 1-byte
  + Description: Defines the intended action to be performed.
* sizeData:
  + Size: 4-byte (equivalent to int)
  + Description: Specifies the size of the data to be sent along with the command and ensures that only the required bytes are read and that the size of the data received can be accurately verified.
* Data:
  + Size: n-bytes
  + Description: Contains the actual data to be transmitted in accordance with the command.

A decision was made to utilize the cmd byte as an indicator of unsuccessful completion of the client's request. For instance, in a scenario where the user interface sends a command to execute a certain action, and the ROS system is unable to perform it, the value of the cmd byte will be utilized to indicate an error.

#### Current Protocol

The current protocol conforms to the original design of the protocol. It is attached to this document as Appendix C: A.C

## ROS Robotic System Overall Architecture

### Introduction

During the analysis phase, the components of the ROS robotic system were identified and characterized, except for the chess piece detection module. To create an initial architecture to enable smooth development a preliminary concept for simplifying the detection of chess pieces had to be developed.

### Preliminary concept to detect chess pieces

From a conceptual standpoint, it was identified that the steps of a chess game are well-defined and can be utilized to simplify the chess piece detection module. The assumption was made that a game of chess on the system would always start from the initial position. Based on these assumptions, a concept was developed to avoid the need for actual detection of the type of chess pieces.

The use of an external RGB and depth camera positioned directly above the chess board was considered as a means to detect the chess board cells and the chess pieces with their respective colours. To utilize the external camera's position data for the robot arm, it is necessary to integrate its position information into the ROS environment for the purpose of conversion. By utilizing this information, it would be possible to extract the board state of the chess board and the general position of the chess pieces in the world.

To determine the movement of a chess piece by the player, a comparison can be made between the current and previous board states of the chess board. By utilizing the colours and internal state of the chess board, the movement of a piece can be inferred. This method can be employed for legal and illegal move checking purposes. However, the issue of how to handle promotion moves arises. Since a pawn can be promoted to several pieces, this presents a challenge. To address this, an initial investigation into the frequency of promotion moves was conducted.

Due to the lack of existing research, the information used to support the decision-making process may come from sources that are not fully reliable or verified, such as Reddit. According to a Reddit post, out of 92 million examined games, only 15.5 million featured promotion moves [35]. This indicates that promotion moves occurred in only 16% of games. This was supported by another Reddit post claiming that out of 20million games examined only 4 million promotions occurred [36]. The examined games were majorly blitz games. This would equate to a promotion per 20million games of 20%.

Based on the statistics gathered, considering that promotion moves happen approximately every fifth game or even less frequently, it was determined that it is reasonable for the player to indicate the promotion move in the user interface when playing chess in the real world, compared to approaching a different design.

To pick up the chess pieces, the robotic arm would utilize the previously extracted general positions of the chess pieces to move to the designated chess piece. The arm would then use its own colour and depth sensor to grasp the piece with accuracy.

### ROS Robotic System Initial Architecture

Figure 4.2 Initial Architecture of the ROS robotic system

Figure 4.2 depicts the initial approach in designing the architecture of the robotic system. Its goal was to establish a modular, extensible, and highly adaptable system. This approach allows for targeted modifications to individual modules without necessitating a substantial overhaul of the remaining system components in the event of any future alterations.

In Figure 4.2, the grey circles symbolize the hardware and the corresponding operating environment. The arrows illustrate the flow of information.

#### System components

The main components of the system, as depicted in Figure 4.2, predominantly comprise the following components:

* **Communication API**
  + Its main responsibility is to facilitate communication with external devices by adhering to the prescribed protocol. In addition, it is tasked with the transmission of commands to the state machine.
* **System State Machine**
  + The role of the state machine is to establish a cohesive sequence of executing commands by linking all the modules together in a logical fashion.
  + This module further incorporates a safety system that facilitates the termination of system operations in the event of potential safety hazards.
* **Robot Arm State Machine**
  + The objective of the arm movement module is to grasp a chess piece and move it to a designated location.
* **Image Processing Robot Camera**
  + The responsibility of this module is to generate data that enables the robot arm to securely grasp the chess piece.
* **Chess board detection**
  + The responsibility of this module, as described in 4.2.2 is to extract the chessboard state and general chess piece position.
* **Chess Engine Wrapper**
  + The function of this module is to enable the system to engage in a game of chess by utilizing the chess engines.

Each of these system components is referred to as a ROS Node.

ROS nodes are computational processes that are interconnected through a graph structure. They communicate with each other via streaming topics, RPC services, and the Parameter Server. A robotic control system will usually compromise of multiple nodes that are employed, each assigned with specific tasks such as laser range-finding, wheel motor control, localization, path planning, and graphical display. [37]

Furthermore, in order to maintain consistency between internal and external communication, it was determined that adherence to the external protocol would be preferable for communication among individual modules.

### Current ROS Robotic System Architecture

Figure 4.3 Current Architecture of the ROS robotic system

Figure 4.3 presents the current robotic system architecture, which differs from the initial design due to modifications in the flow of information and the absence of a module. Due to the consequence of time constraints, image processing on the robotic arm was not attempted, and hence, this module is missing in the current iteration of the architecture. Moreover, the information flow between the chess board detection module, which is responsible for processing overhead camera images, and the System state machine is absent as it is deemed unnecessary. The system state machine does not require knowledge of the chess board's detection data in its current state, and only the robot arm state machine and the target selection module need this information.

The target selection module is introduced to set and clear targets for the robot arm based on the data from the chess board detection module and the move command from the system state machine.

In the current architecture, an internal protocol has been established for communication between nodes, which differs slightly from the external protocol. This was necessary due to the differences in the internal workings of node communication from the original intent. Certain commands or executions are meant to be performed only internally, hence the need for a different protocol. The internal protocol was designed around the foundation of the external protocol and shares the similarity in certain command bytes for both internal and external commands. However, the internal protocol introduces a new component called the sender, which is used by the nodes to identify the origin of received information.

The reasoning for doing this was that in ROS there is a concept of topics that can be published and subscribed to.

A ROS topic refers to a named communication channel through which nodes can exchange messages with one another. These topics operate on a publish/subscribe model where the production and consumption of information are decoupled from one another. [38]

In order to reduce the frequency of call-backs that a single node must handle, it was determined that nodes should subscribe to their own topic and that other nodes wishing to send information to that node should publish to that topic. Therefore, a sender identifier must be included with the message to enable the recipient node to identify the source of the information.

#### Definition – Internal Protocol

* **cmd**
  + Size: 1-byte
  + Description: Defines the intended action to be performed.
* **sizeData**
  + Size: 4-byte (equivalent to int)
  + Description: Specifies the size of the data to be sent along with the command and ensures that only the required bytes are read and that the size of the data received can be accurately verified.
* **Data**
  + Size: n-bytes
  + Description: Contains the actual data to be transmitted in accordance with the command.
* **Sender**
  + Size: 1-byte
  + Description. Defines the source of the protocol.

#### Current Internal Protocol

The current protocol conforms to the original design of the protocol. It is attached to this document as Appendix D: A.D

## ROS Robotic System Components

### Introduction

This section will provide an in-depth discussion of the critical components.

### Communication API

The Communication API, implemented in the ServerNode.cpp file of the ROS package, is responsible for establishing a TCP server socket to facilitate logging in and connecting to the ROS robotic system. Its primary function is to forward information to the System State Machine. This module strictly adheres to both the external and internal protocol established in the previous section.

To create the C++ server socket, this module utilizes third-party code, as indicated in the IRCServer.h file and Appendix A.1.a). Additionally, third-party code is utilized for converting bytes to integers in C++, as also noted in the IRCServer.h file and the Appendix A.1.b).

### Chess Engine Wrapper

The role of the chess engine wrapper is to act as an intermediary between the robotic system and chess engines. The wrapper was designed to support a variety of chess engines, thereby enabling a plug-and-play approach. This would only be possible if a uniform interface for chess engines were available. In the analysis chapter, it was determined that the Universal Chess Interface (UCI) offers such an interface and is widely used across multiple chess engines.

#### Initial Design

In the original design, the Chess Engine Wrapper was envisioned to act as a mediator between the robotic system and a separate class responsible for managing the chess engine. The chess engine class was intended to hold the instance of the chess engine run, with the expectation that the engine would be capable of implementing illegal moves, as well as detecting check and checkmate conditions. The goal of this design was to create a modular system that could easily accommodate a variety of different chess engines, with the Chess Engine Wrapper providing a standardized interface to facilitate communication between the system and any selected engine.

By decoupling the chess engine from the rest of the system, the wrapper aimed to allow for more flexibility in selecting and integrating different engines with the robotic system. Additionally, this approach allowed for the possibility of swapping out chess engines with minimal impact on the overall system, as long as they adhered to the common interfacing provided by the Chess Engine Wrapper.

Despite the initial design intentions, the actual implementation of the Chess Engine Wrapper underwent drastic changes as the project progressed. Insufficient research was conducted regarding the utilization of chess engines, leading to this outcome. Specifically, the way chess engines are run and that they do not implement examine legal, illegal, check and check mate conditions.

#### Current Design

In order to address this issue, the design was revised. The following class diagram only illustrates the most crucial design:

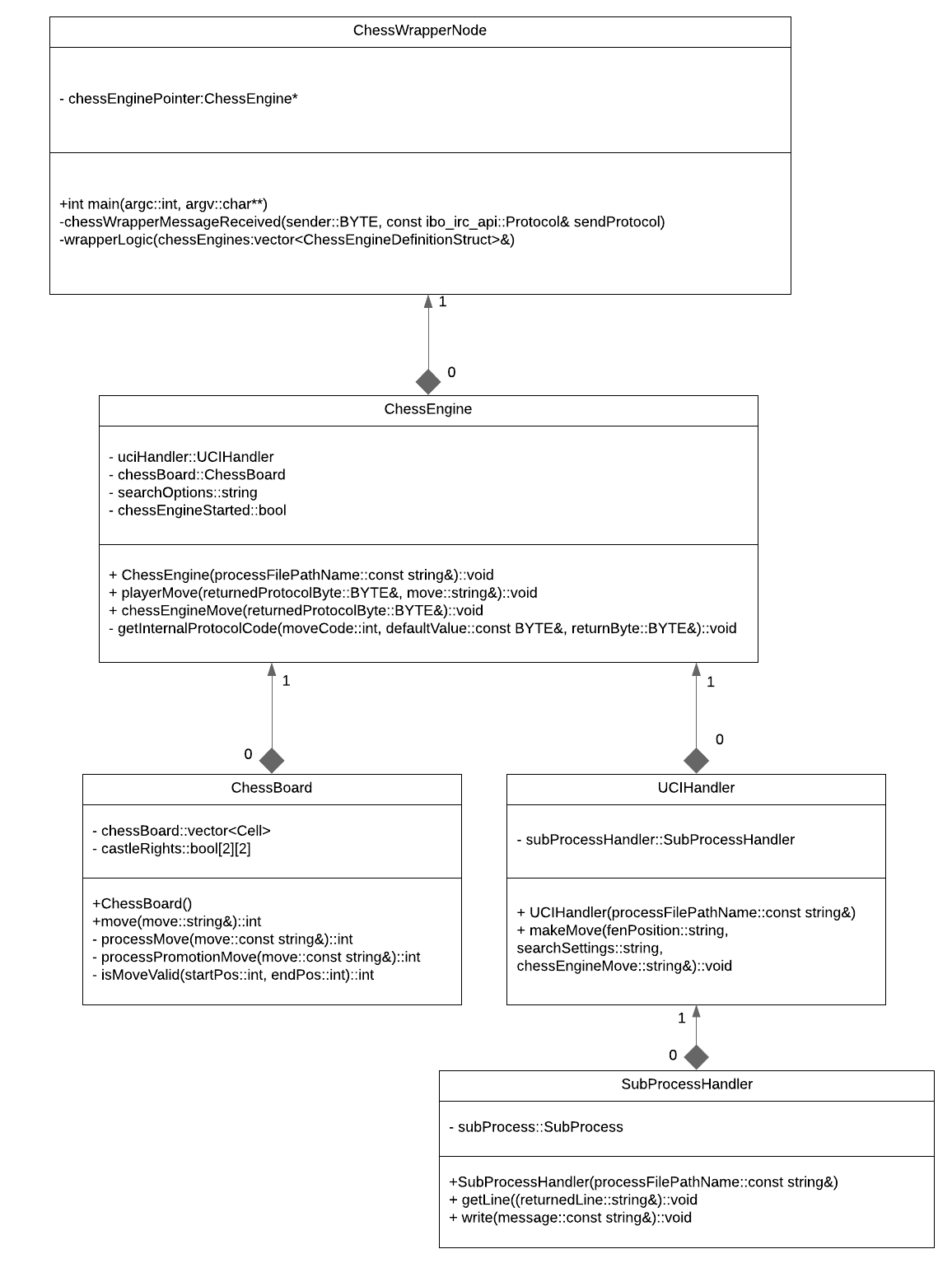


Figure 4.4 UML Class Diagramm ChessWrapperNode

The present design, illustrated in Figure 4.4, was developed to address the challenges encountered during the implementation of the Chess Engine Wrapper. The ChessWrapperNode is the main component of this design, which serves as a communication interface with the internal system. It interacts with other components such as the ChessEngine, ChessBoard, UCIHandler, and SubProcessHandler. The ChessEngine is responsible for holding the instance of the ChessBoard, which represents the internal state of the chess board. The ChessBoard class encompasses the necessary functionality for checking the legality of a move, detecting check and checkmate. The ChessBoard class uses 3rd party code, this is mentioned in the ChessBoard.h file and in Appendix A.1.c). The UCIHandler is attached to the Chess Engine and used as an interface to the various chess engines that may be started.

Starting a child process was necessary to initiate a chess engine process, which does not typically build from source code. However, the lack of experience in developing with C++ and the absence of knowledge of starting processes programmatically made the task very challenging. The SubProcessHandler component was developed to handle the communication with the child process, using third-party code as mentioned in the SubProcessHandler.h file and Appendix A.1.d).

Typically, the communication between a process and an interactor is done via their stdin and stdout. Child processes created inherit the stdin and stdout file descriptor of its parent. But to ensure that the child process does not get any information from the parents stdin, and doesn’t publish information to the parents stdout a different solution was needed. In Linux, piping was utilized to redirect the standard input and output of the child process to a reader and writer to interact and communicate with the child process. The use of this design facilitates the interface between the chess engine and the rest of the system.

The present design iteration provides the capability for a user to engage in a game of chess, with the exception of checkmate and changing the difficulty of the ai player which, at present, remains unimplemented.

### System State Machine

As previously discussed in the section on the overall architecture, the system's state machine plays a vital role in coordinating the execution of commands by linking all the modules together in a logical and cohesive manner. Essentially, the state machine functions as a state machine.

#### State Machine

Due to time constraints, the development of the system state machine was short and only one iteration was completed in a relatively brief period. This is the current version of the system state machine:

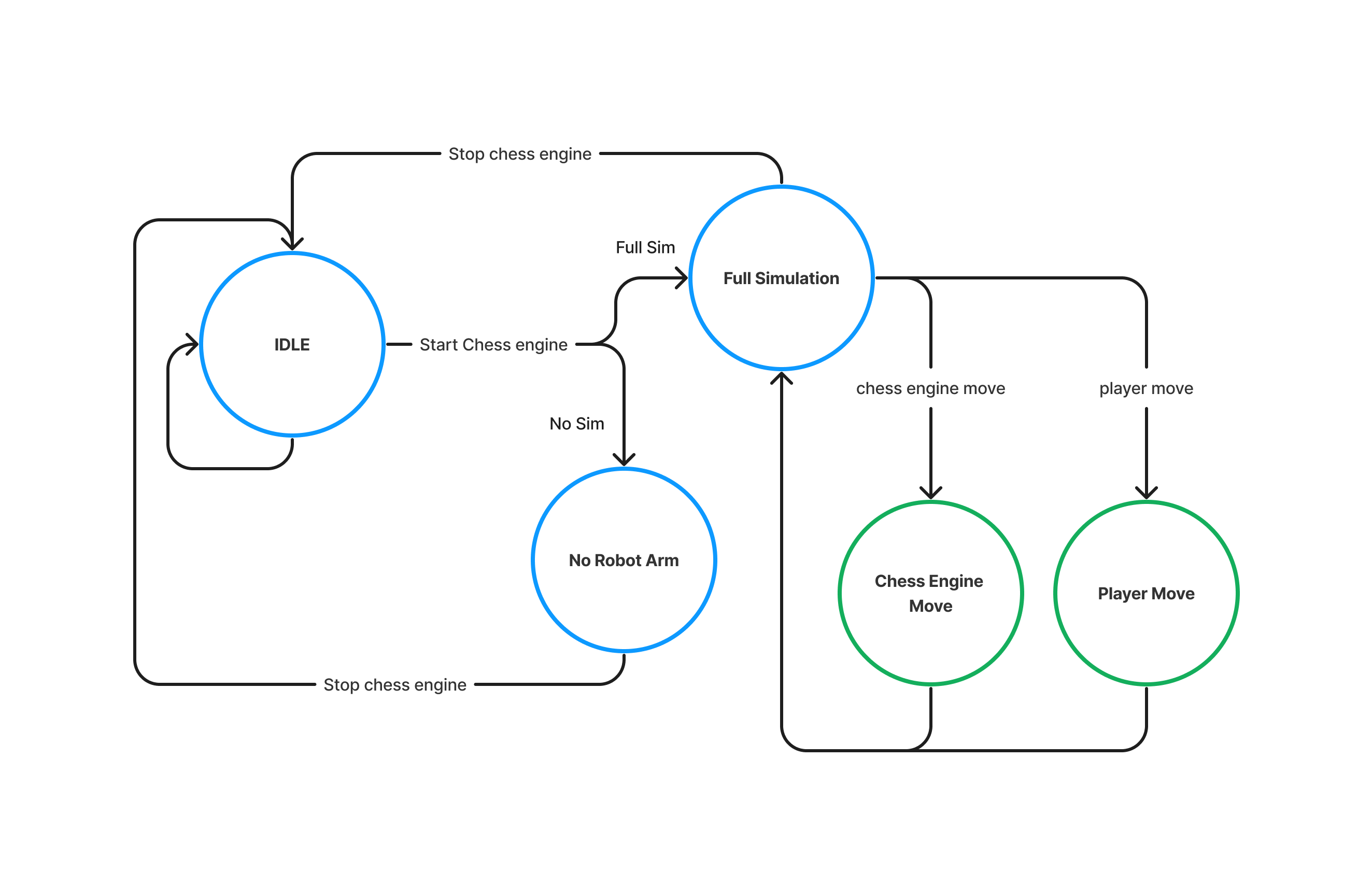


Figure 4.5 Current System State Machine

The System can currently enter 5 different states:

##### IDLE State

In the System State Machine Node, the IDLE state is responsible for receiving and forwarding messages related to changing settings and obtaining information. The current iteration of the IDLE supports following commands:

* Get chess engine names
* Start chess engine
* Set system without sim
* Set system full sim

In order to transition from the IDLE state, a "start chess engine" command must be received by the system.

Upon receiving the command, the chess engine will begin running and the system will determine which parameter the system state is currently set to. Presently, the two available options for the system state parameter are "full sim" and "no sim". It is possible to expand upon this design and include additional pathways in the state machine to accommodate for in-person use. Outside the IDLE state no other state allows the changing of parameters or obtaining this information.

##### No Robot Arm

In the "No Robot Arm" state, users are permitted to play directly against the chess engine by sending commands, such as "player move" and "chess engine move," directly to the ChessWrapperNode, which subsequently executes them. To return to the IDLE state, a "stop chess engine" command must be executed.

##### Full Simulation

In the "Full Simulation" state, users engage in a simulated game against a robot arm. This state triggers a specific simulation state machine that handles commands such as "player move," "chess engine move," and "chess engine stop" for a simulated world. However, no other command will be executed in this state. To exit the "Full Simulation" state, the chess engine stop command is invoked, which sets the system state machine back into the IDLE state.

When either the "chess engine move" or "player move" command is executed, a subsequent flow of tasks is initiated by the system. To provide a visual representation of this flow, a flowchart will be presented for both commands.

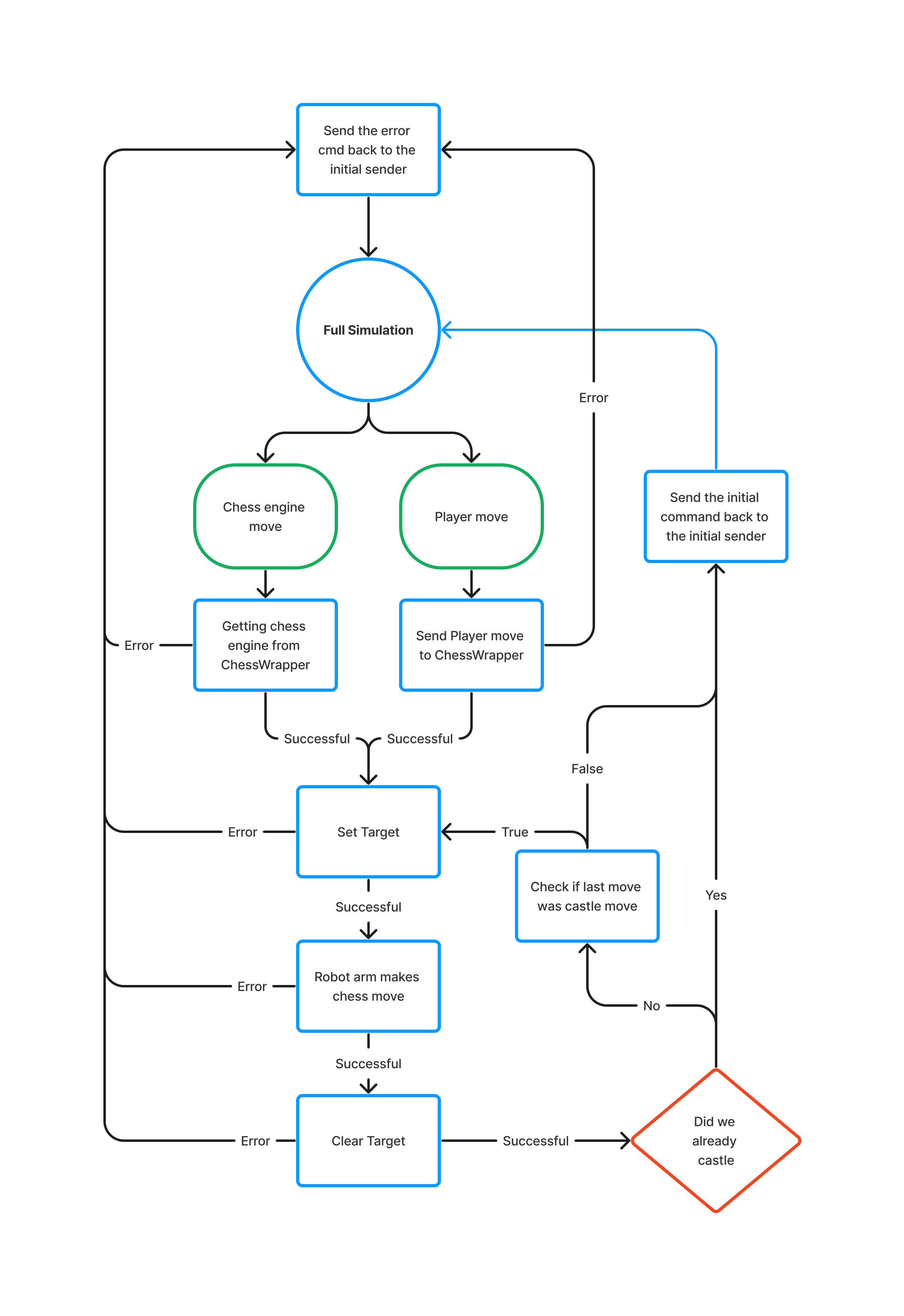
Chess engine and player move flow chart:

Figure 4.6 Chess Engine and Player move in Full Simulation Flow Chart

The provided flowchart demonstrates the sequential steps involved in executing a "chess engine" or "player" move within the context of the "Full simulation" state. The process starts by determining whether a "chess engine" or "player" move has been requested and subsequently establishes the necessary communication with the ChessWrapper. After verifying that no errors have occurred, the system sets the target for the desired move and commands the robot arm to execute it. Once the move is completed, the system clears the target and checks if a castle move was made. If a castle move was not performed, the system then checks if the last move was a castle move. The system then loops back and repeats the previous steps for the second move of a castle move. If no errors are encountered, the system sends the requested command's information back to the initial sender of the internal execution request. However, if any errors arise during the process, the system terminates the execution and returns the error message to the initial sender of the internal execution request.

#### Example frames when setting a target

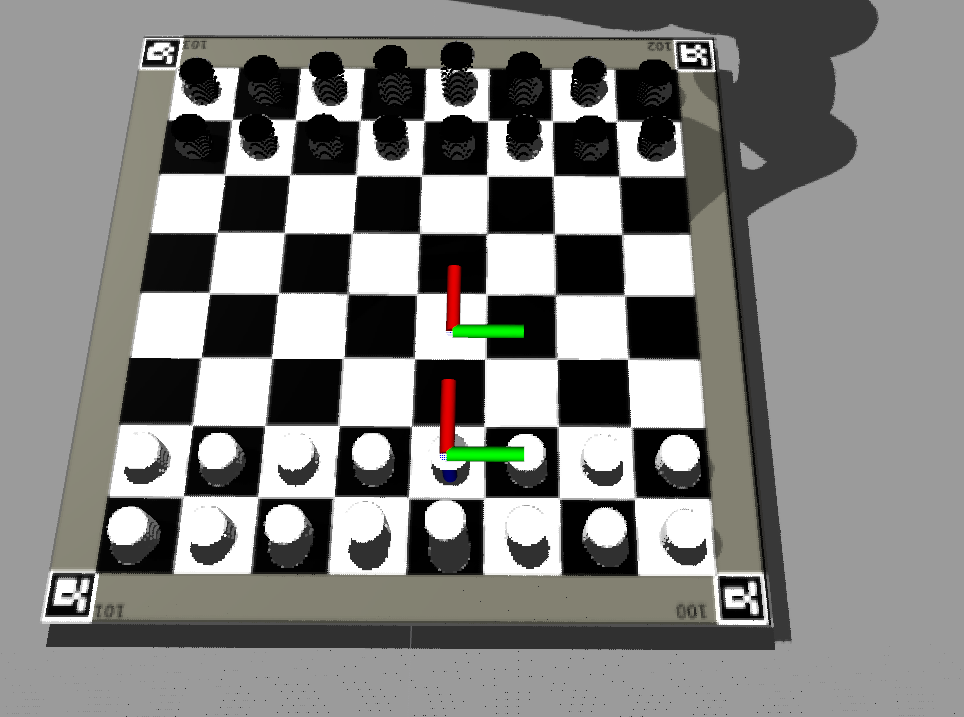


Figure 4.7 Set Target Example 1

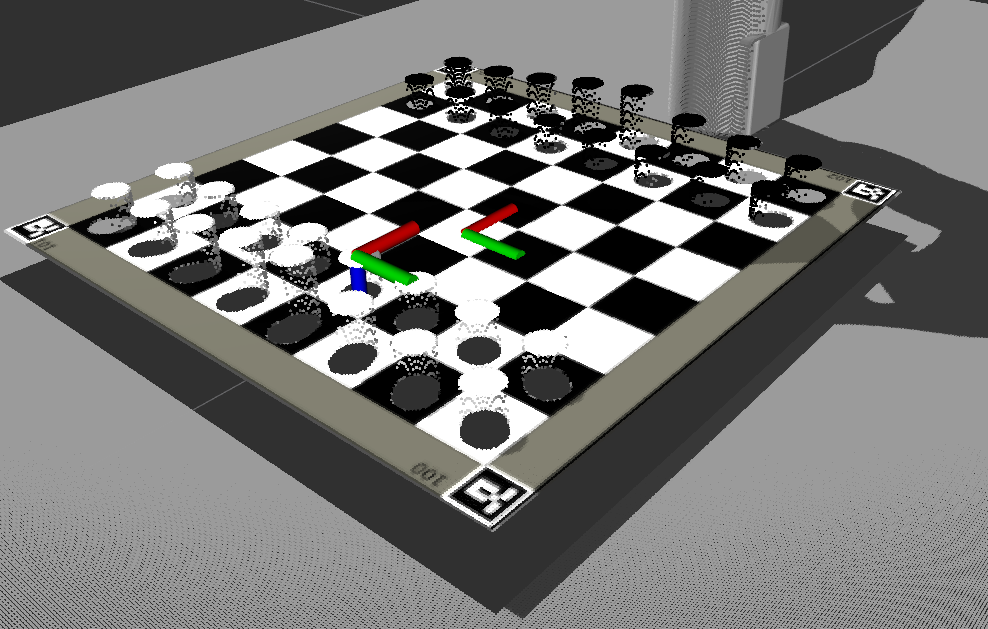


Figure 4.8 Set Target Example1 different point of view

It can be observed from Figures 4.4 and 4.5 that the target selector node generates two distinct entities: the pickup\_frame and drop\_frame. These entities are subsequently employed by the robot arm system to determine the spatial coordinates of the chess piece being retrieved and the target location to which the chess piece will be transported, respectively.

### Robot Arm

In a real-world setting, the Robot arm is tasked with executing chess movements and picking up chess pieces for the chess engine. In a "Full simulation" context, the robot arm would be responsible for both the player's and the chess engine's movements. To accomplish this, the arm relies on targets established by the TargetSelecterNode. The robot arm operates according to a primary state machine, which identifies the type of chess movement required. It then enters a pick and drop state machine specifically designed for picking up and dropping chess pieces.

Two types of chess movements must be handled: basic chess movements, in which a single chess piece moves from its original cell to its destination cell, and take movements, in which the destination cell must be cleared before the chess piece can move from its original cell to its destination cell.

The current iteration and only iteration of the overall state machine does not handle errors by the robot arm.

#### Overall state machine

Figure 4.9 Current Overall Robot Arm State Machine

##### IDLE State

Until a "robot arm move" command is received, the state machine remains in an idle state. When such a command is received, its validity is checked, along with the published information from the chess board cell detection regarding the state of the chess board. In cases where an error occurs, such as the chess board cell detection not publishing any information, or the move command being invalid, the system remains in the idle state and returns an error message to the command's initial sender.

However, if both sets of data are accurate, the destination cell's occupancy status is checked. If the destination cell is occupied, the chess piece occupying it must be removed before the chess piece at the origin cell can be moved to its intended destination. If the destination cell is unoccupied, the chess piece at the origin cell can be moved directly to its destination.

##### Clearing Destination Cell State

To clear the destination cell, the pick and drop state machine is provided with distinct origin and destination positions for picking up and dropping the chess piece. In a clearing pick and drop scenario, the pick and drop state machine is supplied with the destination cell as the pick position, and the "graveyard" as the drop position, which is a designated location in the world where captured chess pieces are moved.

Once the pick and drop state for clearing is completed, the system transitions to the moving the chess piece state.

##### Moving the chess piece State

In the moving the chess piece state, the pick and drop state machine is provided with the origin cell of the chess piece to be moved as the pick position, and the destination cell as the drop position. Upon successfully moving the chess piece, the system transitions to the sending finished back to the initial sender state.

##### Sending finished back to initial sender

During the sending finished back to the initial sender state, a message indicating successful completion of the command is sent back to the command's initial sender. Following this, the system returns to the idle state.

#### Pick and Drop state machine

The responsibility of picking up and dropping chess pieces lies with the pick and drop state machine, which receives the pick and drop positions as inputs. By providing flexibility in determining where to pick up and drop chess pieces, the pick and drop logic can be adapted to various scenarios.

##### The state machine:

Figure 4.10 Current Pick and Drop State Machine

###### Description

The pick and drop state machine remains in an idle state until a pick and drop execution is triggered. Once initiated, it initiates a sequence of executions that proceeds to the next execution when the robotic arm reaches the previous execution's position. In the event that the position is not reached, the state machine retries the same execution. Upon the completion of all executions, the pick and drop state machine returns to its idle state, and the overall state machine state is incremented.

The flow of execution is as follows:

1. Move to intermediate position for picking up a chess piece
2. Close Grippers slightly.
   1. This is to ensure that the robot arm fits in between chess pieces
3. Move to final position for picking up a chess piece
4. Close grippers to grip the chess piece
5. Move to intermediate position for picking up
6. Move to intermediate position for dropping
7. Move to final position for dropping
8. Open grippers slightly.
   1. Grippers are only slightly opened, as in the case of dropping it on the chess board there are possible other chess pieces in the vicinity. For the graveyard it would be possible to open the grippers fully. But to future proof and simplify this process the decision was made to only open the grippers slightly.
9. Move to intermediate position for dropping
10. Move to rest position and open grippers
11. Increment the overall state machine

##### Current iteration and why it is the way it is

The present version of the pick and drop state machine lacks the ability to handle potential issues that may arise during movement or gripping, such as unreachable positions, and fails to report them as errors. Instead, the system retries until the desired position is reached. This is due to a lack of sufficient time during the implementation process to address such issues. Although the current iteration does not present any issues with picking and dropping, it will require further development and refinement The decision to implement a state machine instead of a flow of executions was based on the rationale that it would enable the system to incorporate additional conditions for proceeding to the next execution in the future. Additionally, the pick and drop mechanism currently does not incorporate feedback from the robot arm camera, since the existing simulation only utilizes abstract chess pieces represented as cylinders that conform to the standard height of a chess piece.

The standard height definition of a chess piece height has been extracted from the International Chess Federation (FIDE) Handbook for standards of chess equipment [39].

### Chess piece detection

As delineated in the overall architecture for the ROS robotic system, the detection of chess pieces necessitated a simplified approach. To accomplish this, an external camera was employed to obtain a generalized location for the chess pieces and their respective colours, enabling the robot arm to determine their precise positions and move accordingly for grasping. The robot arm camera was intended for grasping the chess pieces, as previously discussed in the robot arm section. However, due to time constraints, this feature was not implemented. Moreover this extracted information would also be used for extracting the chess board state.

In order to overcome the challenge of extracting information regarding the chessboard cells without regard to the type of chess piece located therein, image processing techniques were utilized.

The OpenCV [40] library provides a multitude of tools for image processing. Within the ROS framework, the opencv\_apps [41] package offers a wide range of OpenCV functionalities.

#### Disclaimer

The solutions presented below are predicated upon an ideal environment and a static position for the chessboard, as time constraints prevented the exploration or implementation of a generalized approach applicable to real-world scenarios.

#### Initial Design

The initial approach involved utilized hough\_lines and contour\_moments.

hough\_lines utilizes the Hough lines transform which is a technique used in computer vision for detecting straight lines in an image [41].

contour\_moments utilizes the moments function of OpenCV. The moments function calculates the moments of a contour or image, which can be used to extract various properties such as the centroid, area, and orientations [41].

The hough\_lines function, using the straight lines typically present in a chessboard, can be utilized to identify straight lines in an image. By combining the resulting lines with contour moments, it becomes feasible to extract the precise positions of the individual chessboard cells.

##### Empty chess board:

Figure 4.11 contour\_moments of the hough\_lines from Figure 4.11

Figure 4.12 hough\_lines image empty chess board initial design

As illustrated in Figure 4.11, the image captured by the external camera, when processed with hough\_lines, generates lines for both the chessboard cells and the board itself. The integration of this information with contour moments permits the creation of a cell representation, as depicted in Figure 4.12.

Given the fixed position of the camera above the chessboard, and the constant dimensions of the board, it is feasible to extract the cell dimensions at the centre of the captured image. This provides insight into the expected area of a chess cell. Utilizing this expected area with a margin allows the extraction of all chess board cells.

##### Chess board with chess pieces

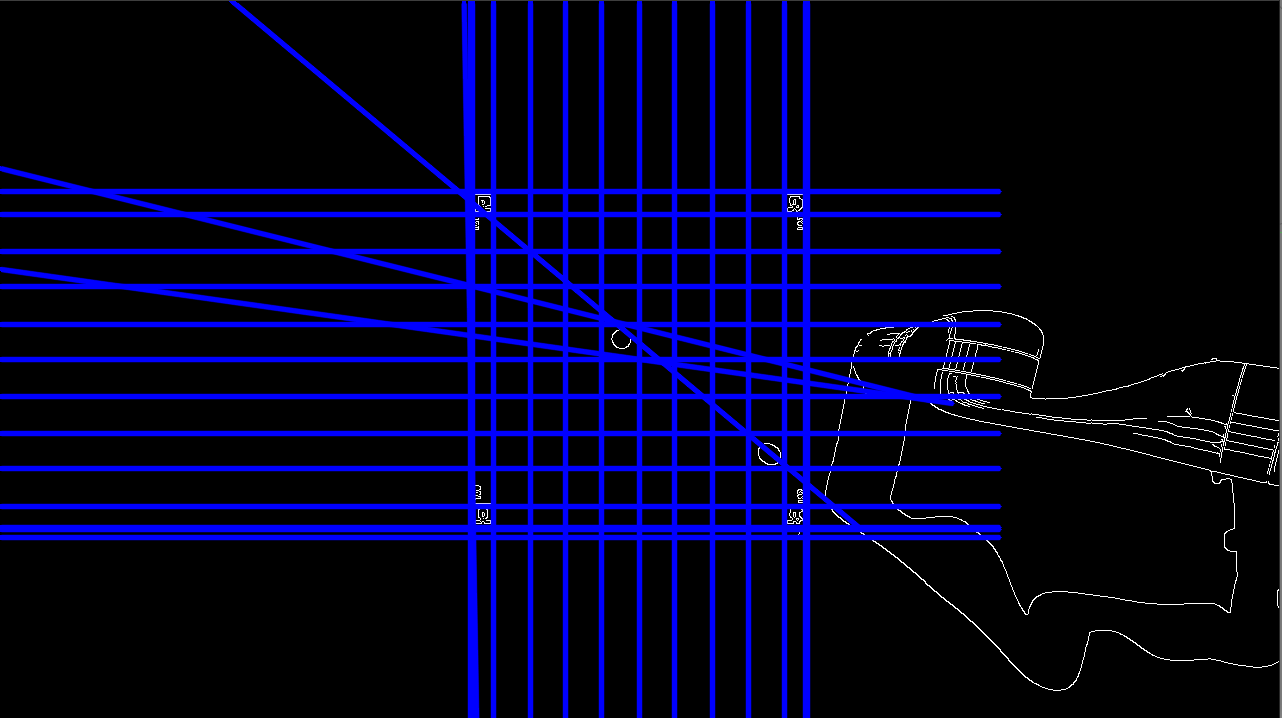
Investigating the changes of the approach with chess pieces in the cell.

Figure 4.13 hough\_lines with a chess piece initial design

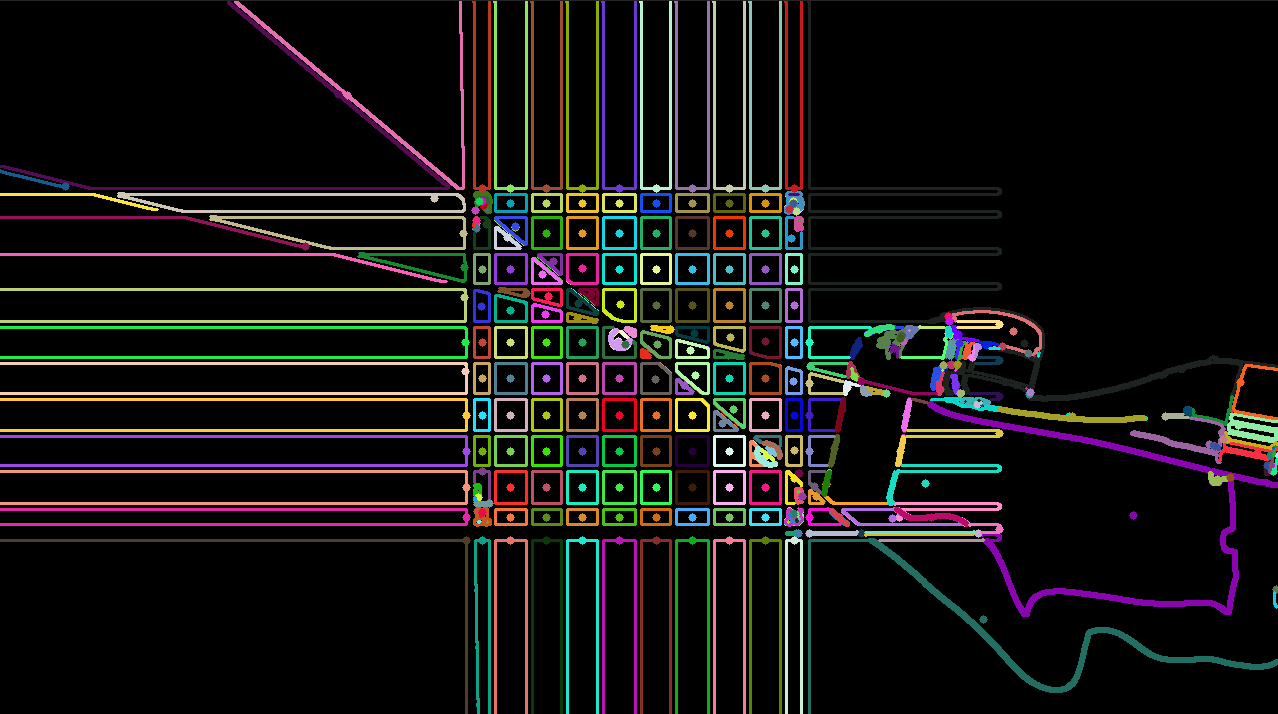
As depicted in Figures 4.13 and 4.14, the introduction of chess pieces and subsequent adjustments made by the robot arm generate additional lines that pose challenges for this design. Consequently, the initial design was abandoned, and an alternative approach was pursued.

Figure 4.14 contour\_moments of the hough\_lines from Figure 4.13

#### Current Design

Although the fundamental objective of utilizing OpenCV functions to streamline chess piece detection remained unchanged, alternative functions had to be explored. The GoodFeaturesToTrack function is capable of identifying the prominent corners within an image or a designated region of an image [42]. This library is made available in the opencv\_apps package of ROS under the name of goodfeatures\_track [41].

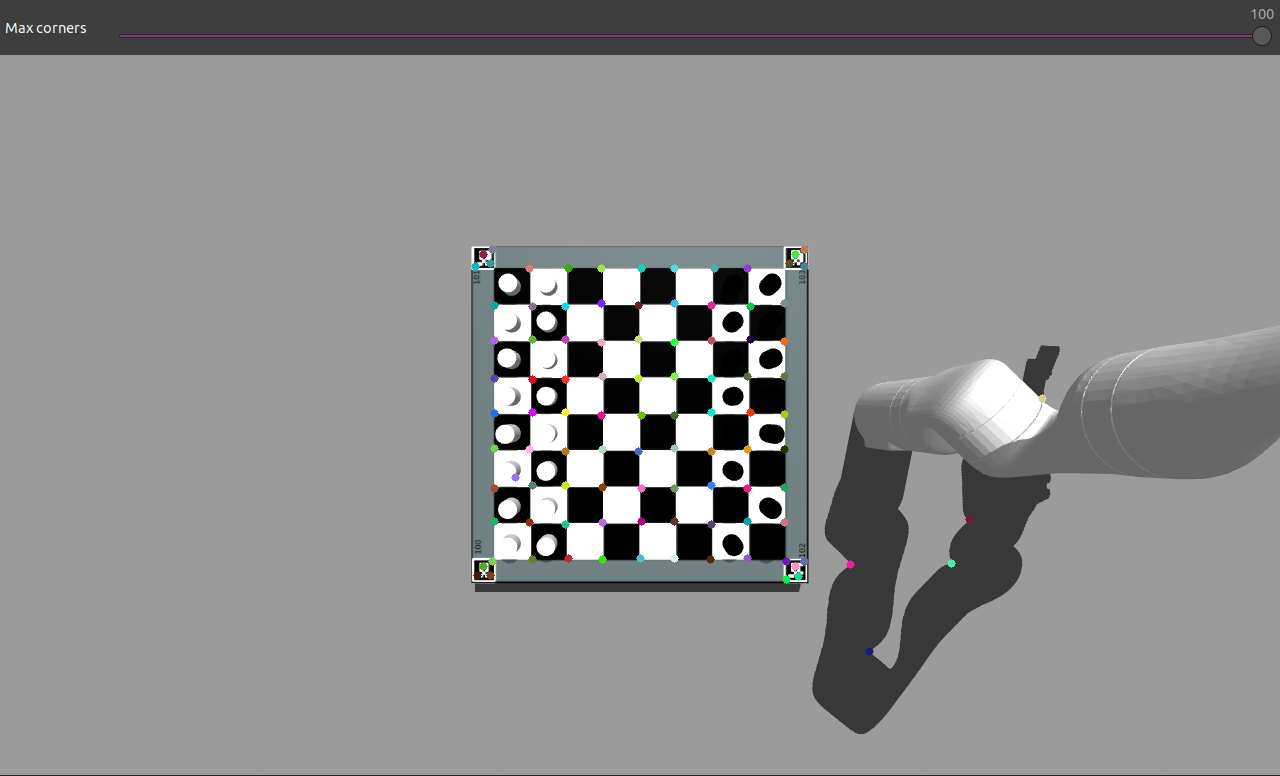
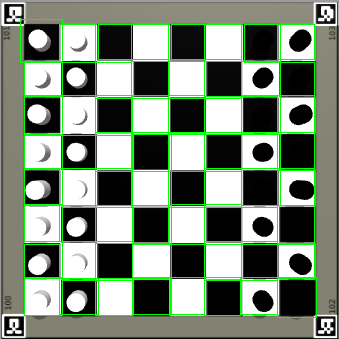


Figure 4.15 goodfeature\_track with chess pieces

Figure 4.16 extracted information from Figure 4.15

The functionality of the goodfeature\_track algorithm is demonstrated in Figures 4.15. Extracting the dimensions of the cells in the center of the captured image, given the camera's fixed position above the chessboard and the board's constant dimensions, enables us to understand how big a chess cell is supposed to be. This information enables the algorithm to estimate the expected distance between two points to form squares or rectangles for the cells. By comparing each point in the goodfeature\_track algorithm with other points based on their distances, the algorithm can determine the position of the chessboard cells and create the appropriate squares or rectangles. To optimize the algorithm's performance and avoid unnecessary redundancy, the points are sorted in a set by their x and y coordinates, thus allowing the algorithm to avoid checking points that have already been processed. These squares/rectangles are illustrated in Figure 4.15.

By utilizing the depth camera to analyze the depth of each pixel in the cell areas and having knowledge of the depth of an unoccupied cell, it is possible to determine whether a chess piece occupies a given cell and, if so, the distance of the piece from the camera. By utilizing the known depth of an empty cell and the measured distance, we can estimate the size of the chess piece. This information, combined with the RGB sensor of the camera, enables us to generate a representation of the chessboard state, indicating the location and color of each chess piece. This is illustrated in Figure 4.17. This information is then used by the target selector and robot arm.

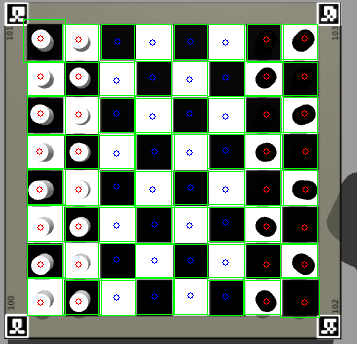


Figure 4.17 Chessboard state based on the information from depth sensor and rgb sensor

Figure 4.18 below shows the flow of data and its data type for the chessboard detection.

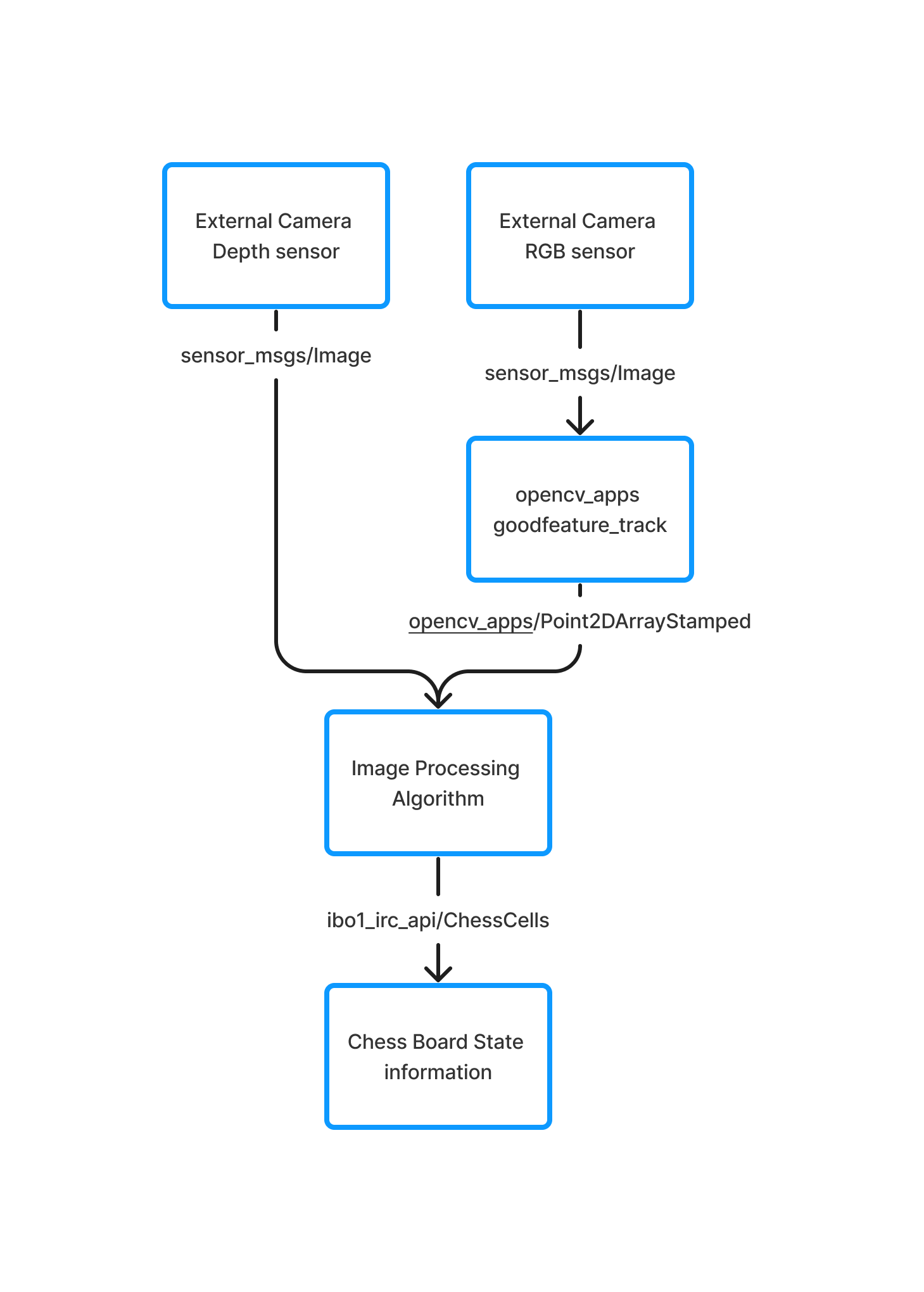


Figure 4.18 Flow of data and its data types for chess board cell detection

#### Robot arm restrictions

During the process of the robot arm moving to pick up a chess piece, the camera's view will be obstructed. Therefore, the algorithm's findings are only made available if all 64 chess cells are detected. Figure 4.19 and 4.20 provides an illustration of this obstruction.

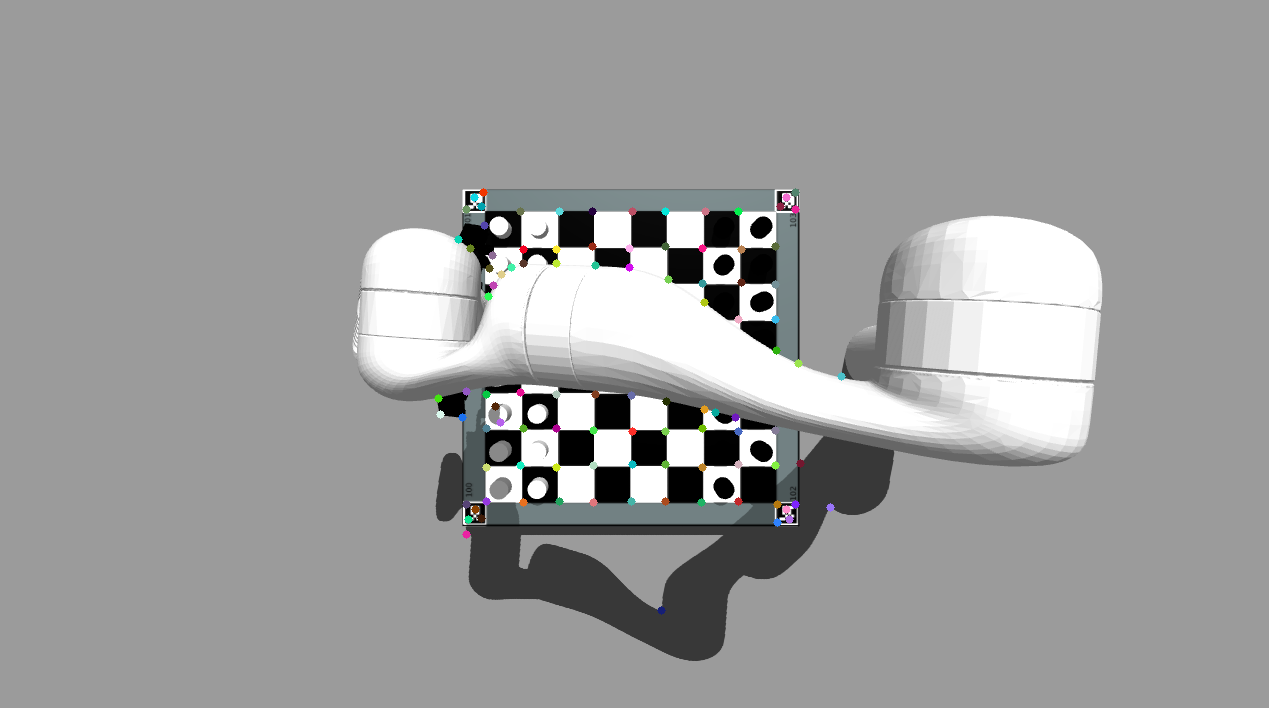


Figure 4.19 goodfeature\_track blocked by arm

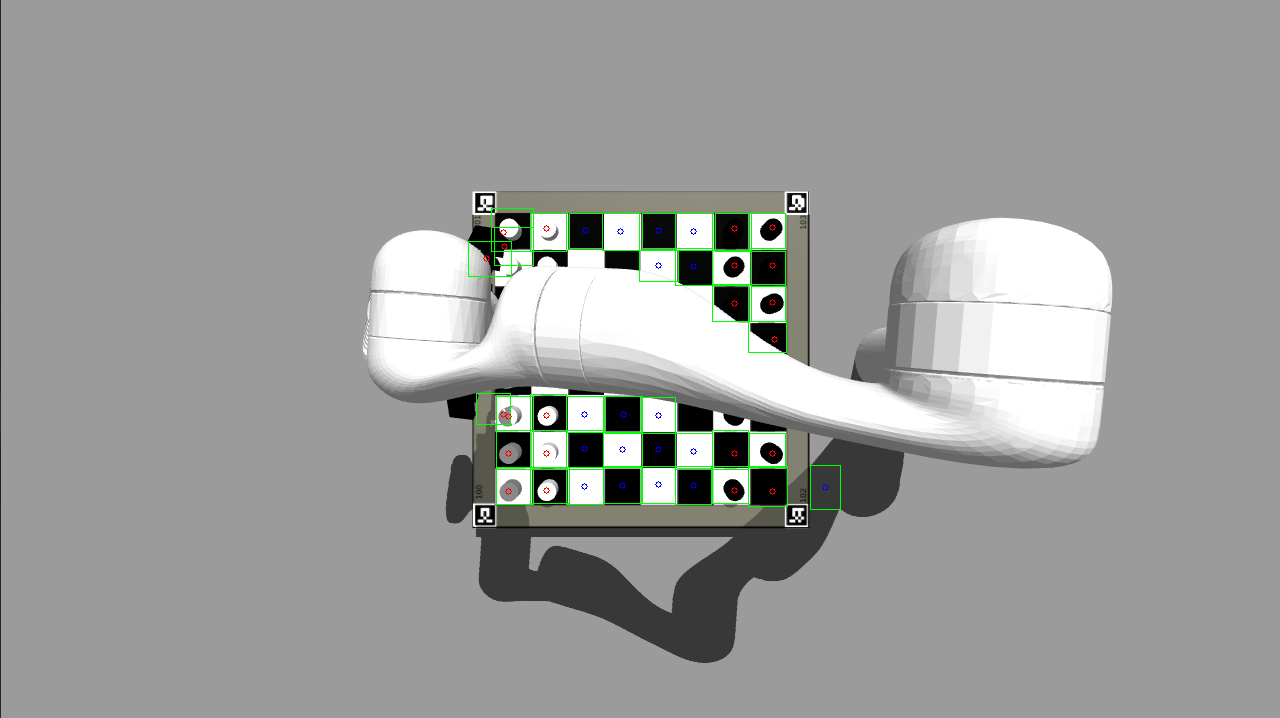


Figure 4.20 chess board information extraction blocked by robot arm

## User Interface

## Overall architecture

The project's user interface system comprises two primary components. The first is a library that facilitates the implementation of all system functions within any user interface framework. The second component is the user interface prototype, which serves as a demonstration of the library's functionality and interaction.

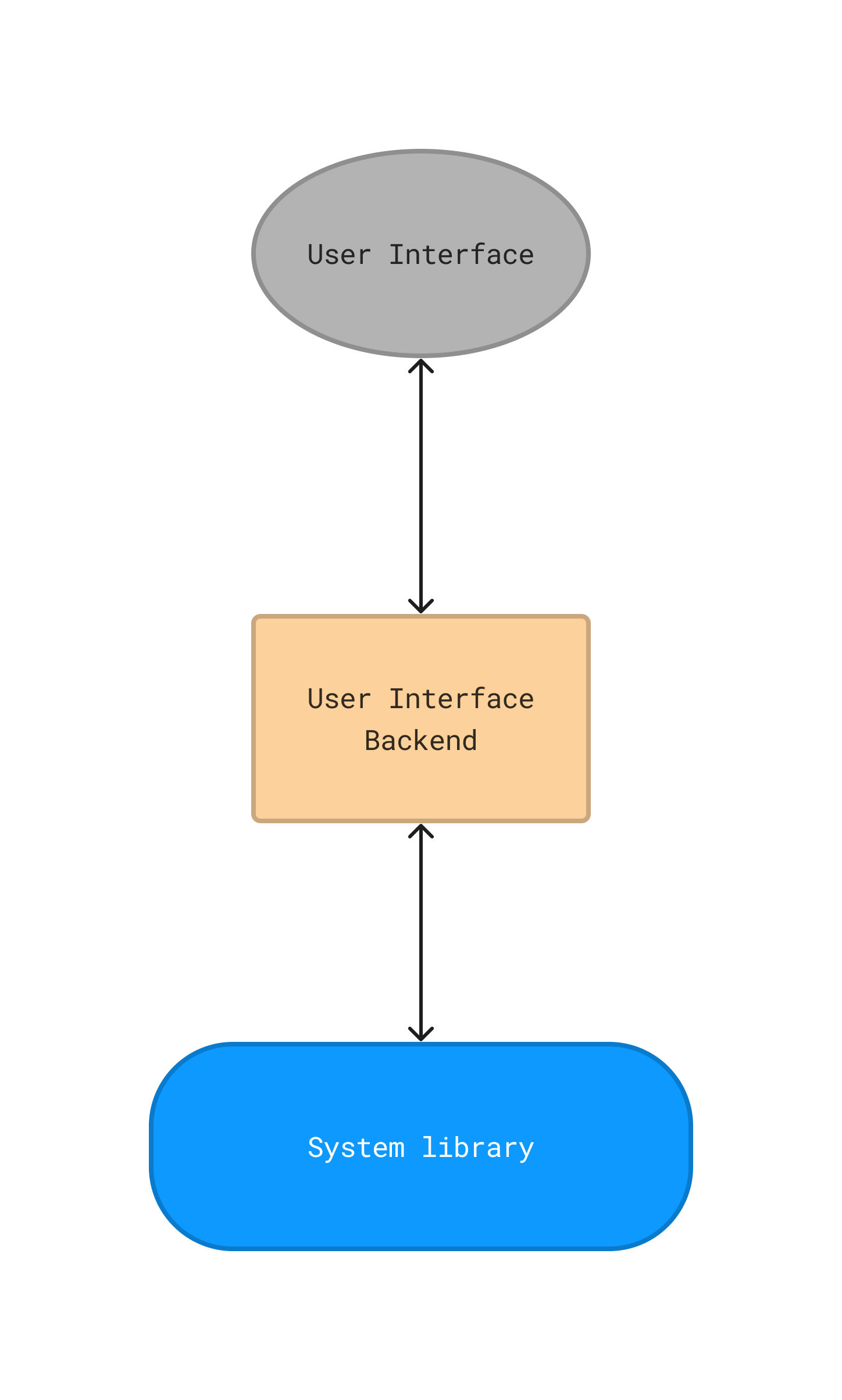


Figure 4.21 User Interface overall architecture

Figure 4.21 depicts the system library for the user interface, including the user interface backend, which specifies the visual design of the user interface. However, the backend utilized for this project is only a prototype, implemented solely to demonstrate the functionality of the system library. The user interface presented to the end-users is represented in Gray.

The System library is expected to facilitate the execution of the protocol as specified in section 4.2.1, and establish the requisite entities for usage in the user interface, thereby enabling users to engage with the functionality of the protocol through said interface.

As mentioned in the analysis chapter the system library will be written in Java [43] and the prototype of the user interface will be written in JavaFX [21].

As the user interface is a prototype to illustrate the implementation of the system library it is not deemed important enough to be mentioned in detail in this report.

## System library

### Introduction

As previously noted, the system library functions as the underlying mechanism that enables the implementation of the protocol for any user interface that is compatible with or utilizes the Java programming language. To achieve this, the system library incorporates a Java-based client that interacts with the ROS robotic system server. The library encapsulates the protocol's various commands as objects, with each object corresponding to their specific commands.

Moreover, the system library provides additional objects, such as the chess board and chess pieces, which simplify the implementation of a chess board on the user interface. The UI can utilize these objects by representing them visually, thus enabling users to employ them as necessary.

Each protocol object throws exceptions for protocol errors. These exceptions are represented as ProtocolErroExceptions.

### UML Class Diagram

Due to the size of the UML class diagram, it will be split up into multiple categories.

#### Cell and Chess pieces

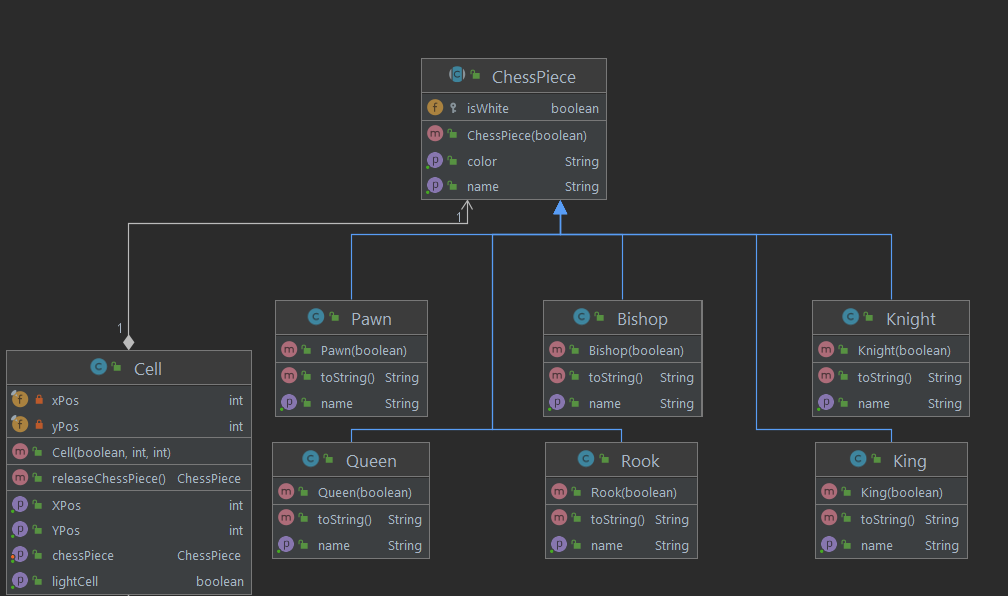


Figure 4.22 ChessPieces and Cell UML Class Diagram

#### ChessEngine, ChessBoard and IRCClient

Figure 4.23 ChessEngine, CHessBoard and IRCClient UML Class diagram

#### User, ROSSystem and ProtocolObject

Figure 4.24 User, ROSSystem and ProtcolObject UML CLass DIagram

The three UML class diagrams depicted in Figures 4.21, 4.23 and 4.24 show the types of classes that were created to implement the execution of commands for a user interface.

The classes User, ChessEngine and ROSSystem utilitze the IRCClient class to send and execute commands on the ROS robotic system. Each of them holds their own reference to the commands and errors that they deal with from the protocol.

The IRCClient holds the socket to communicate to the ROS system.

The ProtocolObject represents the definition of the protocol as seen in section 4.2.1.

The classes ChessBoard, Cell and chess pieces enable the user interface an easy inclusion of a chessboard representation and a way to utilize player moves, chess engine moves and such.

# Testing

Detailed descriptions of every test case are definitely not what is required in this section; the place for detailed lists of tests cases is in an appendix. In this section, it is more important to show that you adopted a sensible strategy that was, in principle, capable of testing the system adequately even if you did not have the time to test the system fully.

Provide information in the body of your report and the appendix to explain the testing that has been performed. How does this testing address the requirements and design for the project?

How comprehensive is the testing within the constraints of the project? Are you testing the normal working behaviour? Are you testing the exceptional behaviour, e.g. error conditions? Are you testing security issues if they are relevant for your project?

Have you tested your system on “real users”? For example, if your system is supposed to solve a problem for a business, then it would be appropriate to present your approach to involve the users in the testing process and to record the results that you obtained. Depending on the level of detail, it is likely that you would put any detailed results in an appendix.

Whilst testing with “real users” can be useful, don't see it as a way to shortcut detailed testing of your own. Think about issues discussed in the lectures about until testing, integration testing, etc. User testing without sensible testing of your own is not a useful activity.

The following sections indicate some areas you might include. Other sections may be more appropriate to your project.

## Overall Approach to Testing

## Automated Testing

### Unit Tests

### User Interface Testing

### Stress Testing

### Other Types of Testing

## Integration Testing

## User Testing

# Critical Evaluation

Examiners expect to find a section addressing questions such as:

* Were the requirements correctly identified?
* Were the design decisions correct?
* Could a more suitable set of tools have been chosen?
* How well did the software meet the needs of those who were expecting to use it?
* How well were any other project aims achieved?
* If you were starting again, what would you do differently?

Other questions can be addressed as appropriate for a project.

The questions are an indication of issues you should consider. They are not intended as a specification of a list of sections.

The evaluation is regarded as an important part of the project report; it should demonstrate that you are capable not only of carrying out a piece of work but also of thinking critically about how you did it and how you might have done it better. This is seen as an important part of an honours degree.

There will be good things in the work and aspects of the work that could be improved. As you write this section, identify and discuss the parts of the work that went well and also consider ways in which the work could be improved.

In the latter stages of the module, we will discuss the evaluation. That will probably be around week 9, although that differs each year.

# Figures

[Figure 2.1 General User- User Stories 14](#_Toc134086758)

[Figure 2.2 User and Admin - User Stories 15](file:///C:\Users\Omar\Desktop\Interactive%20Robot%20Chess.docx#_Toc134086759)

[Figure 2.3 Admin - User Stories 15](#_Toc134086760)

[Figure 4.1 User Interface - ROS Robotic System Communication 20](#_Toc134086761)

[Figure 4.2 Initial Architecture of the ROS robotic system 23](#_Toc134086762)

[Figure 4.3 Current Architecture of the ROS robotic system 25](#_Toc134086763)

[Figure 4.4 UML Class Diagramm ChessWrapperNode 28](#_Toc134086764)

[Figure 4.5 Current System State Machine 30](file:///C:\Users\Omar\Desktop\Interactive%20Robot%20Chess.docx#_Toc134086765)

[Figure 4.6 Chess Engine and Player move in Full Simulation Flow Chart 32](file:///C:\Users\Omar\Desktop\Interactive%20Robot%20Chess.docx#_Toc134086766)

[Figure 4.7 Set Target Example 1 33](#_Toc134086767)

[Figure 4.8 Set Target Example1 different point of view 34](#_Toc134086768)

[Figure 4.9 Current Overall Robot Arm State Machine 35](file:///C:\Users\Omar\Desktop\Interactive%20Robot%20Chess.docx#_Toc134086769)

[Figure 4.10 Current Pick and Drop State Machine 37](file:///C:\Users\Omar\Desktop\Interactive%20Robot%20Chess.docx#_Toc134086770)

[Figure 4.11 contour\_moments of the hough\_lines from Figure 4.11 40](file:///C:\Users\Omar\Desktop\Interactive%20Robot%20Chess.docx#_Toc134086771)

[Figure 4.12 hough\_lines image empty chess board initial design 40](file:///C:\Users\Omar\Desktop\Interactive%20Robot%20Chess.docx#_Toc134086772)

[Figure 4.13 hough\_lines with a chess piece initial design 41](file:///C:\Users\Omar\Desktop\Interactive%20Robot%20Chess.docx#_Toc134086773)

[Figure 4.14 contour\_moments of the hough\_lines from Figure 4.13 42](file:///C:\Users\Omar\Desktop\Interactive%20Robot%20Chess.docx#_Toc134086774)

[Figure 4.15 goodfeature\_track with chess pieces 43](file:///C:\Users\Omar\Desktop\Interactive%20Robot%20Chess.docx#_Toc134086775)

[Figure 4.16 extracted information from Figure 4.15 43](file:///C:\Users\Omar\Desktop\Interactive%20Robot%20Chess.docx#_Toc134086776)

# References

|  |  |
| --- | --- |
| [1] | NASA, "NASA Science MARS EXPLORATION," NASA, 2019. [Online]. Available: https://www.mybib.com/tools/ieee-citation-generator. [Accessed 1 May 2023]. |
| [2] | ABB, "ABB Robotics solutions for the Automotive industry," ABB, 2023. [Online]. Available: https://new.abb.com/products/robotics/industries/automotive?gad=1&gclid=Cj0KCQjw6cKiBhD5ARIsAKXUdyYJZJKC300oQEn37bAUx4mqF-km8smXIauCIvN8SO7X-YBFLBVNT\_gaAl1cEALw\_wcB. [Accessed 1 May 2023]. |
| [3] | "Mechanical Turk," Wikipedia, 21 July 2021. [Online]. Available: https://en.wikipedia.org/wiki/Mechanical\_Turk. |
| [4] | A. Silver, "Chess News," 7 July 2022. [Online]. Available: https://en.chessbase.com/post/robots-and-chess. [Accessed 1 May 2023]. |
| [5] | "Deep Blue (chess computer)," Wikipedia, 2019. [Online]. Available: https://en.wikipedia.org/wiki/Deep\_Blue\_(chess\_computer). [Accessed 1 May 2023]. |
| [6] | J. Henley, "Chess robot grabs and breaks finger of seven-year-old opponent," The Guardian, 24 July 2022. [Online]. Available: https://www.theguardian.com/sport/2022/jul/24/chess-robot-grabs-and-breaks-finger-of-seven-year-old-opponent-moscow. [Accessed 1 May 2023]. |
| [7] | L. P.-A. Jan Rosell, "ROS Final project," Universitat POLITECHNICA DE CATALUNYA BARCELONA TECH, 2023. [Online]. Available: https://sir.upc.edu/projects/rostutorials/final\_work/index.html. [Accessed 1 May 2023]. |
| [8] | ROS, "ROS.org | Powering the world's robots," ROS, 2020. [Online]. Available: https://www.ros.org/. [Accessed 1 May 2023]. |
| [9] | EEDGE-tronics, "Chess-Robot Github," GitHub, 2023. [Online]. Available: https://github.com/EDGE-tronics/Chess-Robot. [Accessed 1 May 2023]. |
| [10] | A. E. Soltis, "Chess," Britannica, 3 Apri 2023. [Online]. Available: https://www.britannica.com/topic/chess. [Accessed 1 May 2023]. |
| [11] | B. SIG, "Bluetooth Technology Overview," Bluetooth SIG, 2023. [Online]. Available: https://www.bluetooth.com/learn-about-bluetooth/tech-overview/. [Accessed 1 May 2023]. |
| [12] | B. SIG, "2022 Market Update," Bluetooth SIG, [Online]. Available: https://www.bluetooth.com/2022-market-update/. [Accessed 1 May 2023]. |
| [13] | A. Wykes, "Why is Bluetooth still so unreliable?," SOUNDGUYS, 12 September 2021. [Online]. [Accessed 1 May 2023]. |
| [14] | "RS-232," Wikipedia, 2020. [Online]. Available: https://en.wikipedia.org/wiki/RS-232. [Accessed 1 May 2023]. |
| [15] | "USB," Wikipedia, 2019. [Online]. Available: https://en.wikipedia.org/wiki/USB. [Accessed 1 May 2023]. |
| [16] | "Internet Protocol," Wikipedia, 2019. [Online]. Available: https://en.wikipedia.org/wiki/Internet\_Protocol. [Accessed 1 May 2023]. |
| [17] | K. Koishigawa, "TCP vs. UDP Wath's the difference and which protocol is faster?," freeCodeCamp, 28 June 2021. [Online]. Available: https://www.freecodecamp.org/news/tcp-vs-udp/. [Accessed 1 May 2023]. |
| [18] | Python, "What is Python? Executive Summary," Python.org, 2019. [Online]. Available: https://www.python.org/doc/essays/blurb/. [Accessed 1 May 2023]. |
| [19] | Java, "What is Java technology and why do I need it?," Java.com, 2022. [Online]. Available: https://www.java.com/en/download/help/whatis\_java.html. [Accessed 1 May 2023]. |
| [20] | Kotlin, "Kotlin Homepage," Kotlin Foundation, [Online]. Available: https://kotlinlang.org/. [Accessed 1 May 2023]. |
| [21] | Gluon, "JavaFX," Gluon OpenJFX, [Online]. Available: https://openjfx.io/. [Accessed 1 May 2023]. |
| [22] | U. Robots, "Collaborative robots from universal robots," Universal Robots 2023, 2023. [Online]. Available: https://www.universal-robots.com/products/#:~:text=UR3e,6.6%20lbs%20(3%20kg).. [Accessed 1 May 2023]. |
| [23] | Kinova, "Imagine the possibilities," Kinova inc., [Online]. Available: https://www.kinovarobotics.com/product/gen3-robots. [Accessed 1 May 2023]. |
| [24] | O. S. R. Foundation, "Gazebo," Open Source Robotics Foundation, 2014. [Online]. Available: https://classic.gazebosim.org/. [Accessed 1 May 2023]. |
| [25] | C. Belshe, "Chess Piece Detection," California Polytechnic State university, 2021. [Online]. Available: https://digitalcommons.calpoly.edu/cgi/viewcontent.cgi?article=1617&context=eesp. [Accessed 1 May 2023]. |
| [26] | Shredder, "UCI," Shredder, 2023. [Online]. Available: https://www.shredderchess.com/chess-features/uci-universal-chess-interface.html. [Accessed 1 May 2023]. |
| [27] | official-stockfish, "Stockfish," GitHub, 2020. [Online]. Available: https://github.com/official-stockfish/Stockfish. [Accessed 1 May 2023]. |
| [28] | komodochess.com, "README.TXT," 2020, [Online]. Available: https://komodochess.com/store/pages.php?cmsid=14. [Accessed 1 May 2023]. |
| [29] | S. C. Foundation, "Standard C++," 2019. [Online]. Available: https://isocpp.org/. [Accessed 1 May 2023]. |
| [30] | Figma, "The Collaborative interface design tool," [Online]. Available: https://www.figma.com/. [Accessed 1 May 2023]. |
| [31] | J. S. Ken Schwaber, "The 2020 Scrum Guide," 2020. [Online]. Available: The iterative approach of Scrum provides the opportunity to test and validate new ideas. [Accessed 1 May 2023]. |
| [32] | kanbanize, "Kanban explained in 10 minutes," 2021. [Online]. Available: https://kanbanize.com/kanban-resources/getting-started/what-is-kanban. [Accessed 1 May 2023]. |
| [33] | GitHub, "GitHub Homepage," GitHub, [Online]. Available: https://github.com/. [Accessed May 1 2023]. |
| [34] | G. Docs, "About Projects," GitHub, [Online]. Available: https://docs.github.com/en/issues/planning-and-tracking-with-projects/learning-about-projects/about-projects. [Accessed 1 May 2023]. |
| [35] | sandlube2, "Pawn Promotion Popularity," Reddit, March 2023. [Online]. Available: https://old.reddit.com/r/chess/comments/10ue8gq/pawn\_promotion\_popularity/. [Accessed 1 May 2023]. |
| [36] | wintermute93, "The pawn with the highest percentage of promotion, statistically," Reddit, February 2023. [Online]. Available: https://www.reddit.com/r/chess/comments/10vinkz/the\_pawn\_with\_the\_highest\_percentage\_of\_promotion/. [Accessed 1 May 2023]. |
| [37] | HabibOladepo, "Nodes," ROS.org, 4 December 2018. [Online]. Available: http://wiki.ros.org/Nodes. [Accessed 1 May 2023]. |
| [38] | TullyFoote, "Topics," ROS.org, 20 February 2019. [Online]. Available: http://wiki.ros.org/Topics. [Accessed 1 May 2023]. |
| [39] | FIDE, "FIDE Handbook," FIDE, 04 August 2022. [Online]. Available: https://handbook.fide.com/chapter/StandardsOfChessEquipment2022. [Accessed 1 May 2023]. |
| [40] | OpenCV, "OpenCV Homepage," OpenCV, 2019. [Online]. Available: https://opencv.org/. [Accessed 1 May 2023]. |
| [41] | iory, "opencv\_apps," ROS Wiki, 15 February 2022. [Online]. Available: http://wiki.ros.org/opencv\_apps#hough\_lines. [Accessed 1 May 2023]. |
| [42] | OpenCV, "Feature Detection," OpenCV, [Online]. Available: https://docs.opencv.org/2.4/modules/imgproc/doc/feature\_detection.html#goodfeaturestotrack. [Accessed 1 May 2023]. |

# Appendices

The appendices are for additional content that is useful to support the discussion in the report. It is material that is not necessarily needed in the body of the report, but its inclusion in the appendices makes it easy to access.

If you have used any 3rd party code, i.e. code that you have not written yourself such as libraries, then you must include Appendix A. In that appendix, you will provide details of the 3rd party code that you have used.

For most other items, it would be better to include them in your technical submission instead of including them as an appendix. For example:

* If you have developed a Design Specification document as part of a plan-driven approach for the project, then it would be appropriate to include that document in the technical work. In this report, you would highlight the most interesting aspects of the design, referring your reader to the full specification for further detail.
* If you have taken an agile approach to developing the project, then you may be less likely to have developed a full requirements specification at the start of the project. Perhaps you used stories to keep track of the functionality and the ‘future conversations.’ If it isn’t relevant to include all those stories in the body of your report, you could detail those stores in a document in the technical work.
* If you have used manual testing, then include a document in the technical work that records the tests that have been done. In this report, you would talk about the use of those tests.

Documents included in the technical work or in the appendices are supporting evidence of the work done. Where you include documents, this report should refer to the documents. You should not be relying on detailed study of those documents in order to understand what is written in this report.

Speak to your supervisor or the module coordinator if you have questions about this.

* 1. Third-Party Code and Libraries
     1. Code
        1. C++ Socket Creation on Linux
        2. C++ bytes to integer conversion
        3. C++ ChessBoard class
        4. C++ SubProcessHandler class

If you have made use of any third-party code or software libraries, i.e. any code that you have not designed and written yourself, then you must include this appendix.

As has been said in lectures, it is acceptable and likely that you will make use of third-party code and software libraries. If third-party code or libraries are used, your work will build on that to produce notable new work. The key requirement is that we understand what your original work is and what work is based on that of other people.

Therefore, you need to clearly state what you have used and where the original material can be found. Also, if you have made any changes to the original versions, you must explain what you have changed.

The following is an example of what you might say.

**Apache POI library** – The project has been used to read and write Microsoft Excel files (XLS) as part of the interaction with the client’s existing system for processing data. Version 3.10-FINAL was used. The library is open source and it is available from the Apache Software Foundation [5]. The library is released using the Apache License [6]. This library was used without modification.

Include as many declarations as appropriate for your work. The specific wording is less important than the fact that you are declaring the relevant work.

* 1. Code Samples

This is an example appendix. Include as many appendices as you need. The appendices do not count towards the overall word count for the report.

For some projects, it might be relevant to include some code extracts in an appendix. You are not expected to put all of your code here - the correct place for all of your code is in the technical submission that is made in addition to the Project Report. However, if there are some notable aspects of the code that you discuss, including that in an appendix might be useful to make it easier for your readers to access.

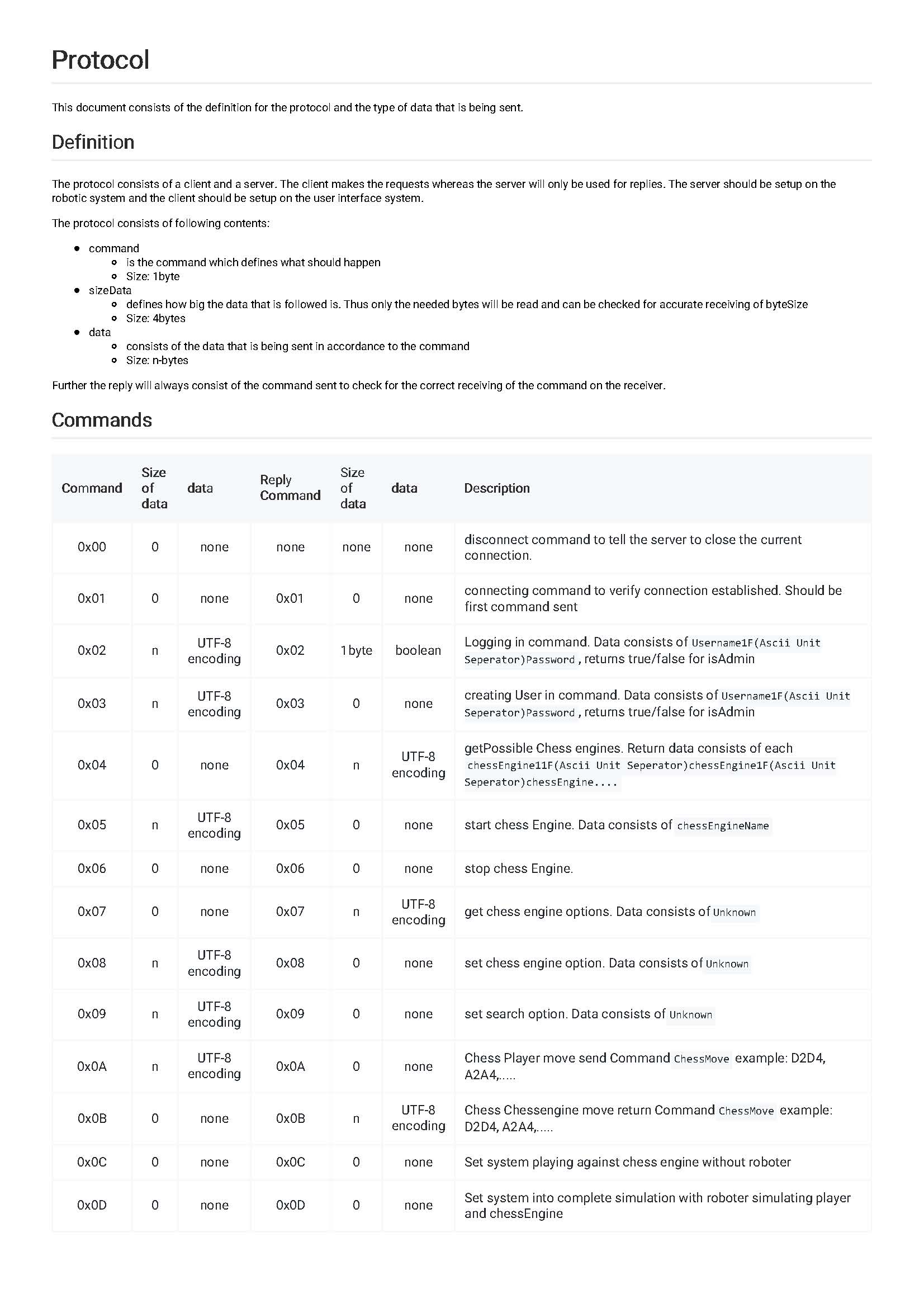
As a general guide, if you are discussing short extracts of code then you are advised to include such code in the body of the report. If there is a longer extract that is relevant, then you might include it as shown in the following section.

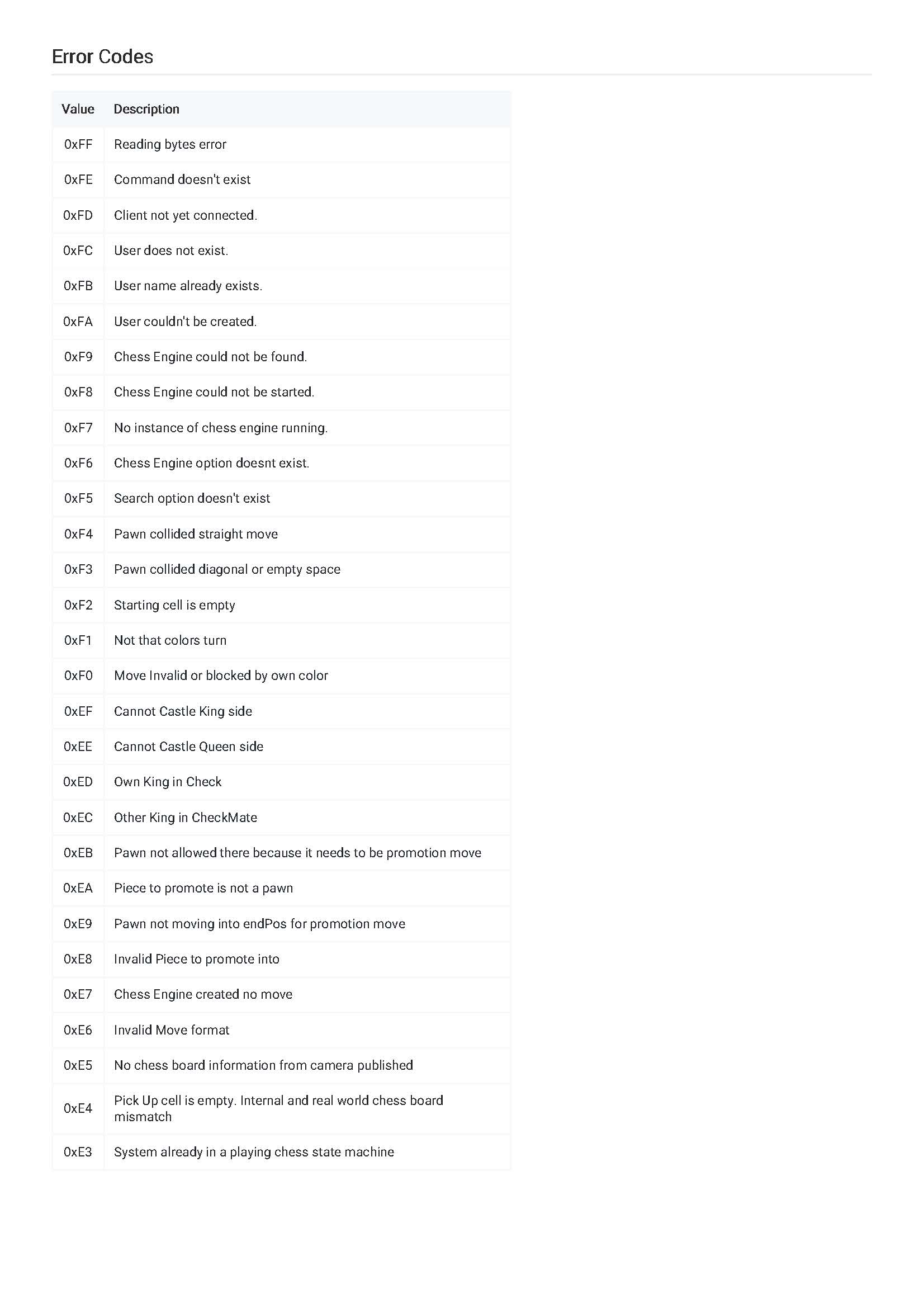
Only include code in the appendix if that code is discussed and referred to in the body of the report.

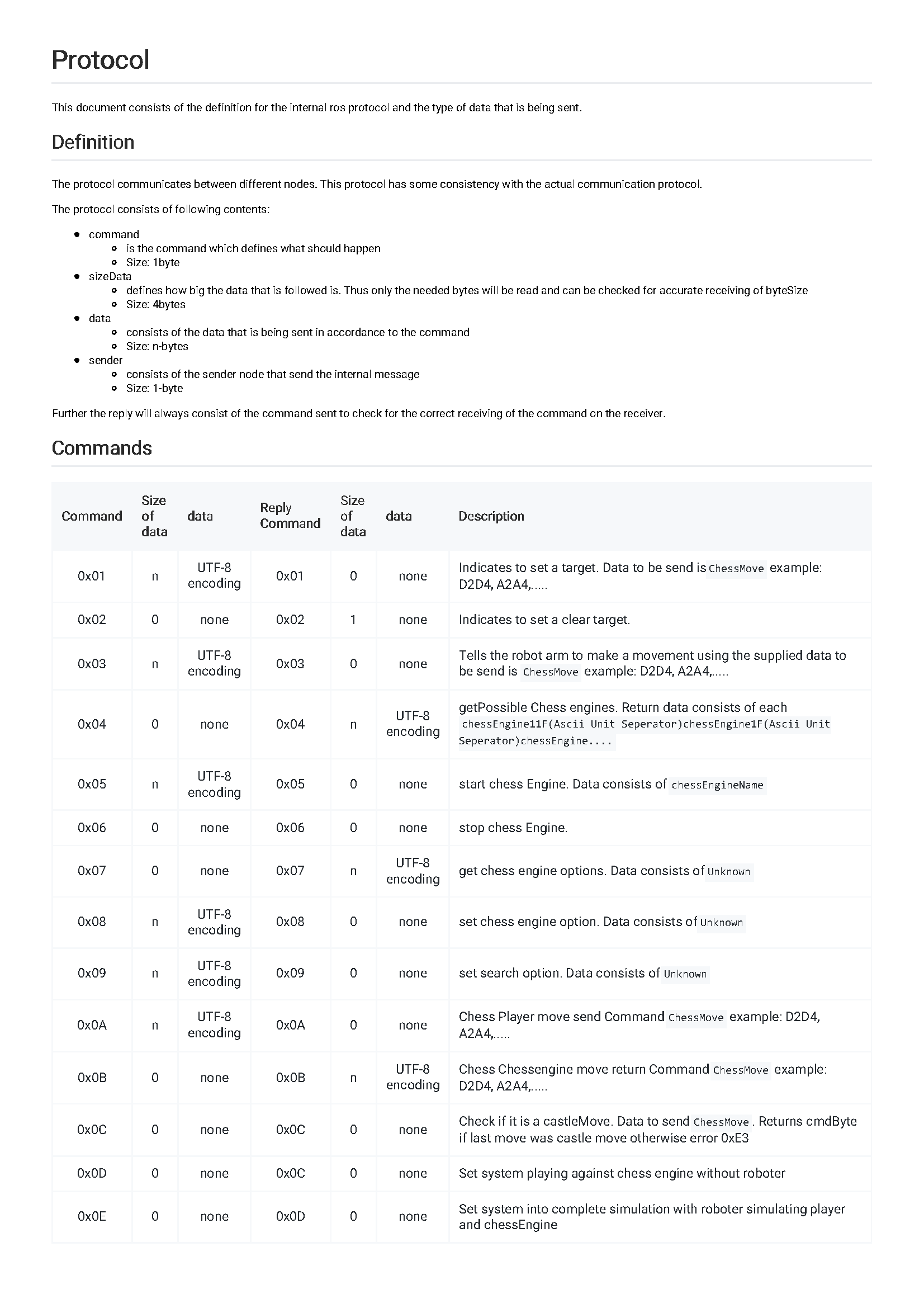
Random Number Generator

The Bayes Durham Shuffle ensures that the pseudo random numbers used in the simulation are further shuffled, ensuring minimal correlation between subsequent random outputs.

// Some example code here…

* 1. Communication Protocol



* 1. Internal Communication Protocol

