e.DO Robot

Product Design Specification

v1.5

Arhum Ahmed
Eric LaBerge
Trevor Kretschmann
Adrian Ionascu

March 2019

Contents

Date	Version	Authors	Description
3/05/2019	v1.0	Arhum Ahmed, Eric LaBerge, Trevor Kretschmann, Adrian Ionascu	First draft
3/07/2019	v1.2	Arhum Ahmed, Eric LaBerge, Trevor Kretschmann, Adrian Ionascu	Added new sequence diagram, revised hardware diagram, added technical specifications for hardware components, and included e.Do topics and messages referenced.
4/15/2019	v1.5	Arhum Ahmed	Revised existing sequence diagrams, added 13 additional sequence diagrams revised hardware limitations revised software architecture revised checkers API updated checkerboard image

1 Introduction

1.1 Purpose

The purpose of this Product Design Document is to outline the design choices of the system and architecture defined for this project. This document serves as a guide for the development team to unambiguously develop the architecture required for this project. In particular, the different software APIs referenced, the hardware included, and the communication between the components are explained in detail.

2 General Overview and Design Guidelines/Approach

The general overview for the e.DO robot will involve one main script that will call a variety of APIs. These APIs will include the AI, the arm movement, and the image detection. The board state will be stored on the main script.

2.1 Assumptions

In regards to the machine itself, first, we must assume that the Robot is powered on and functioning properly. Second, we must assume that the Robot is capable of taking input from the program. This means that the Robot is calibrated, which is a manual process. Finally, we must assume the Robot arm will function without hardware failure, as this would hinder action on the Robot's side, resulting in the game being impossible to complete.

In regards to the setup of the camera and checkerboard, they must be situated in the exact

locations and dimensions specified in the accompanying manual. This calibration is key in ensuring accuracy of detection and manipulation.

In regards to the game logic, we must assume that the human player is aware of the rules of checkers, and will not intentionally break the rules, although there are countermeasures in place if such an event occurs.

In regards to human interaction with the Robot, we must assume the human player will not attempt to hinder the hardware during games. This is not limited to the arm itself, as intentional obstruction of the camera, outside of normal gameplay (moving pieces, removing pieces once captured, etc.), will assuredly cause the program to malfunction.

2.2 Limitations

2.2.1 Hardware Limitations

The technical specification of all elements of hardware are listed in sections 3.1.1 to 3.1.3. The major hardware limitations was that the e.Do Robot provided was a prototype that still had a few bugs and some limitations in it's field of movement.

2.2.2 Software Limitations

The first limitation of the software is that all of its components are to be written in Python 3.6, or reference libraries compatible with Python 3.6. The software must be written to be compatible with the version of ROS that the e.DO robot is using as the software is written exclusively for the e.DO robot and is not expected to have compatibility with other ROS robots. The software must also be tested to ensure that it runs on the current environment set up on the Nvidia Jetson TX2 provided to the team by Comau.

3 Architecture Design

3.1 Hardware Architecture

The project consists of 4 key pieces of hardware. A camera is mounted above a checkers board and takes regular captures of the board state. This feed is sent to the NVIDIA Jetson TX2 AI chip which processes the images using OpenCV. The layout of the board is extracted and converted into an array of values representing each checker piece. The AI program computes the best possible move given this array and outputs another array containing the checker piece that should be moved and its new location. The coordinates are then sent from the Jetson chip into the e.Do Robot which performs the physical movement to execute the desired play onto the checkerboard. The user then makes their move on the checkerboard and the process repeats with this new board state.

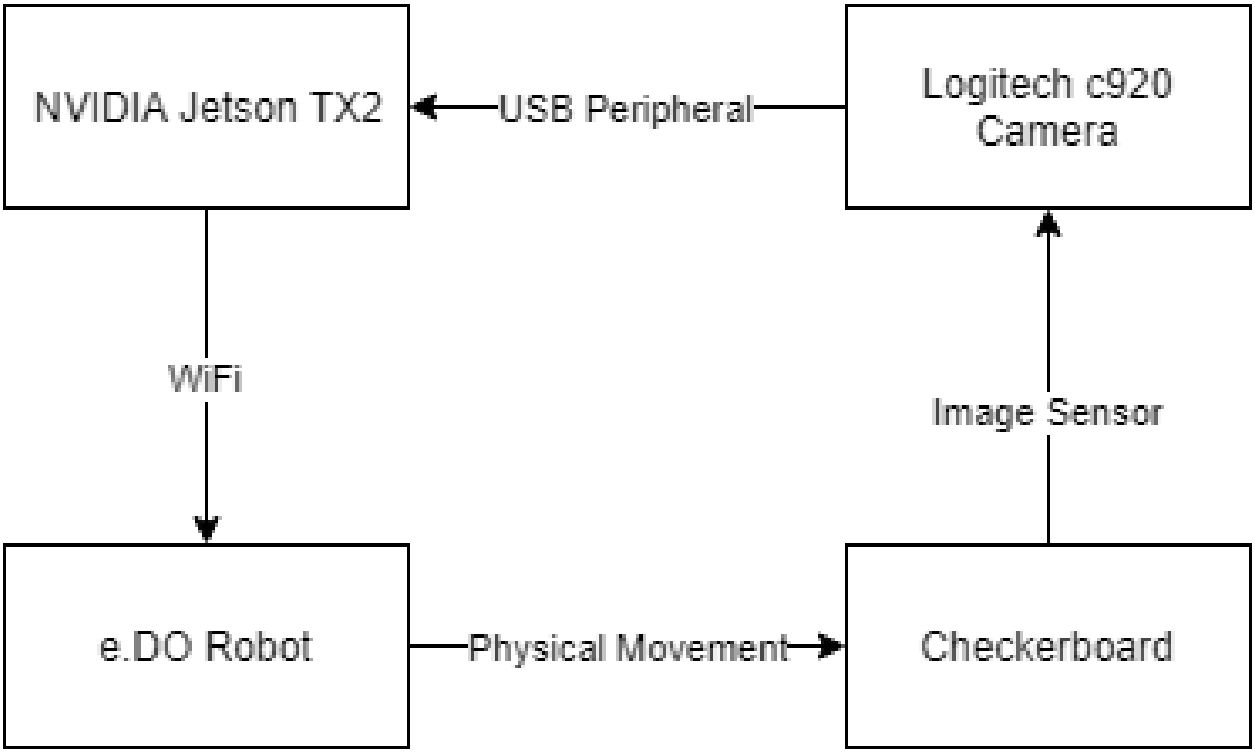

Figure 1: Diagram showing hardware components and connectivity between them

3.1.1 e.DO Robot

The e.DO Robot is a robot designed and made by Comau, with specifications following the table below. The table describes the constraints and limitations of the hardware as given by Comau in the e.DO technical specification documents available at the `https://edo.cloud` website. The table describes details such as the degrees of freedom (or axis) the robot has, and the degrees of motion of each axis along with the speed at which each axis can move. The e.DO robot also contains a seventh motor that is connected to the gripper connected via a flange on the robot to allow for grabbing items.

Specifications		Value
Number of axis		6
Max payload		1 kg (2.2 lbs)
Max reach		478 mm (1.56 ft)
Stroke (speed)	Axis 1	±180°(38°/sec)
	Axis 2	±113°(38°/sec)
	Axis 3	±113°(38°/sec)
	Axis 4	±180°(56°/sec)
	Axis 5	±104°(56°/sec)
	Axis 6	±2700°(56°/sec)
Repeatability		±1 mm
Motherboard		Raspberry Pi running Raspbian Jessie
ROS		Kinetic Kame
Connectivity		1 external USB port 1 RJ45 Ethernet port 1 DSub-9 Serial port

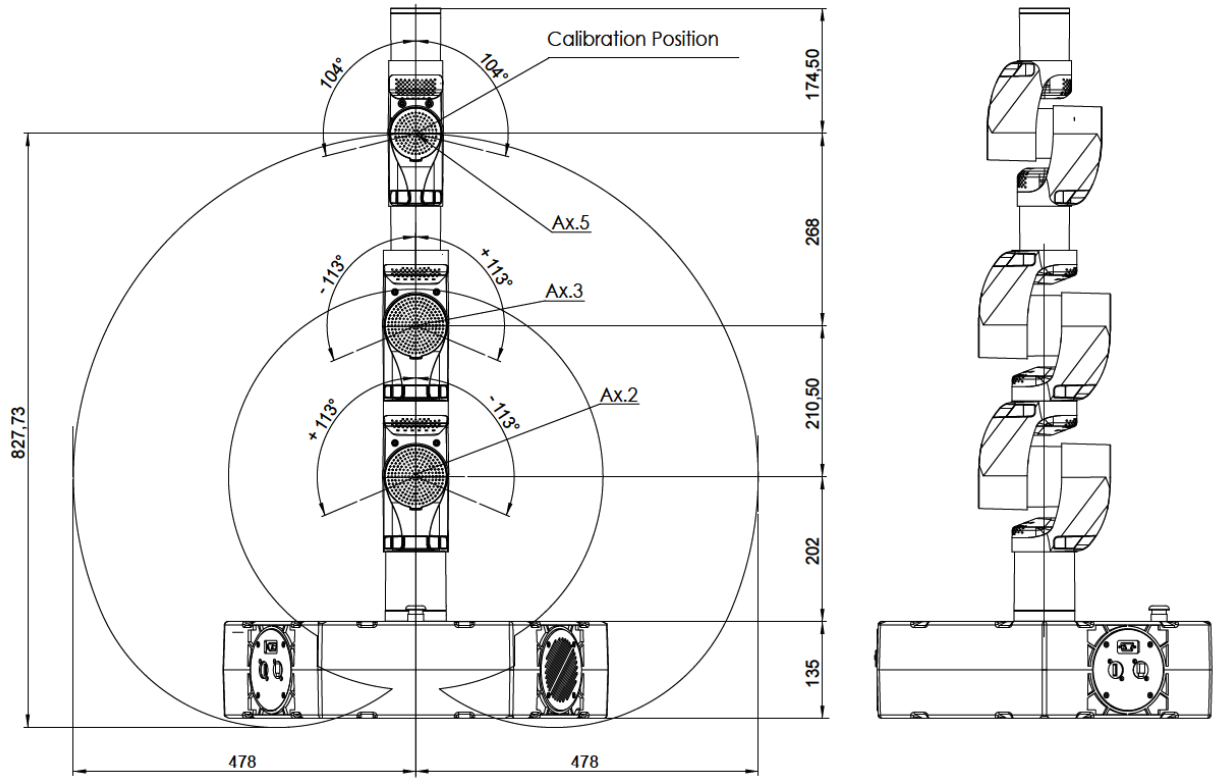

Figure 2: Diagram showing technical specifications of size and movements of the e.DO robot

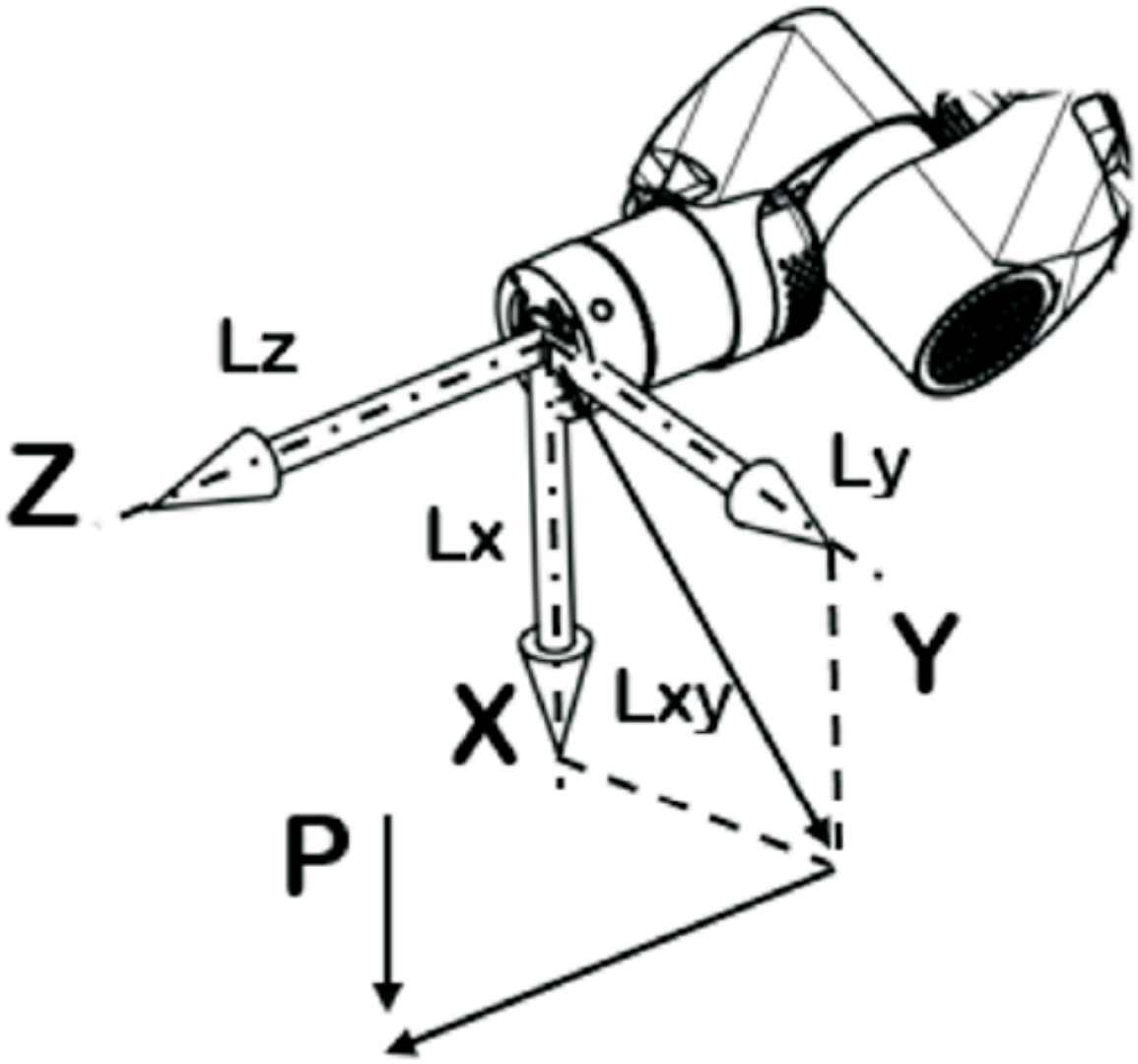

Figure 3: Diagram showing axis in relation to the flange used for attachments

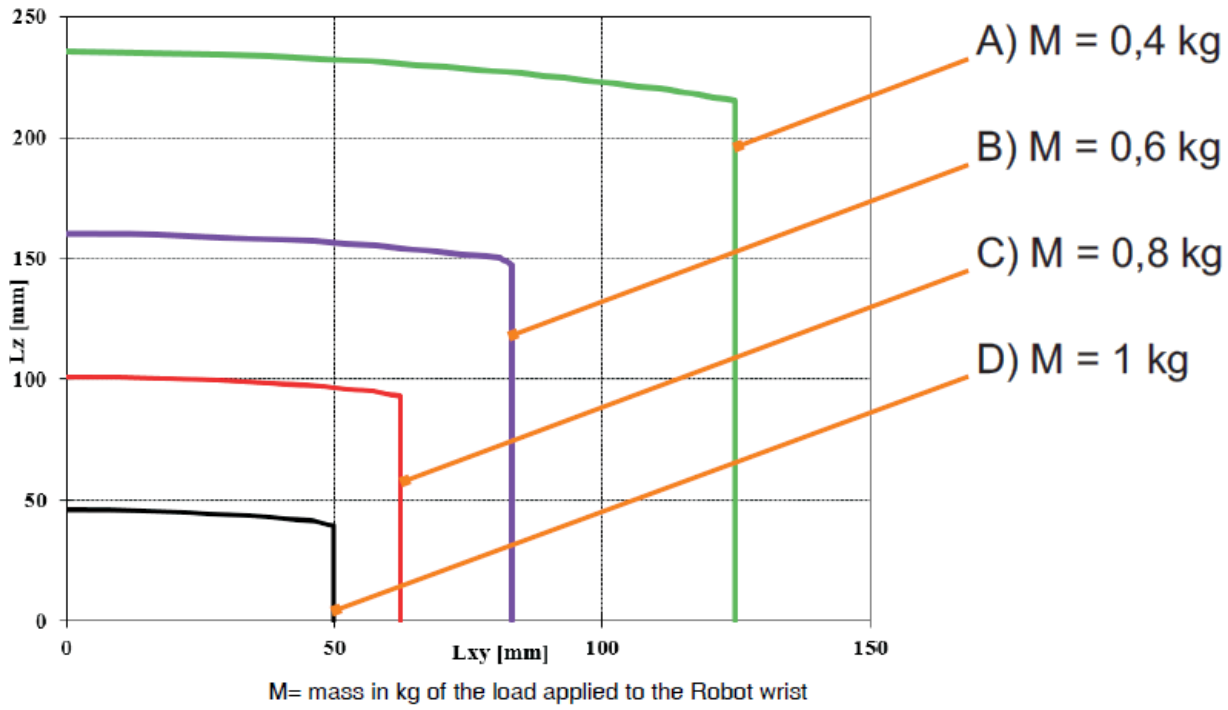

Figure 4: Maximum wrist load based on length of attachment

3.1.2 Camera

The technical specifications of the Logitech c920 are outlined in the following table, which was obtained at: `https://support.logitech.com/en_us/product/hd-pro-webcam-c920/specs`.

Webcam Specifications	
Connection Type	USB
USB Protocol	USB 2.0
USB VID_PID	082D
UVC Support	Yes
Microphone	Yes
Microphone Type	Stereo
Lens and Sensor Type	Glass
Focus Type	Automatic
Optical Resolution	True: 3MP Software Enhanced: 15MP
Diagonal Field of View (FOV)	78°
Horizontal FOV	70.42°
Vertical FOV	43.3°
Focal Length	3.67 mm
Image Capture (4:3 SD)	N/A
Image Capture (16:9 W)	2.0 MP, 3 MP*, 6 MP*, 15 MP*
Video Capture (4:3 SD)	N/A
Video Capture (16:9 W)	360p, 480p, 720p, 1080p
Frame Rate (max)	1080p@30fps
Right Light	RightLight 2
Video Effects (VFX)	N/A
Buttons	N/A
Indicator Lights (LED)	Yes
Privacy Shade	No
Tripod Mounting Option	Yes
Cable Length	5 feet

3.1.3 NVIDIA Jetson

The technical specifications of the NVIDIA Jetson TX2 are outlined in the following table, which was obtained at:
`https://www.nvidia.com/en-us/autonomous-machines/embedded-systems-dev-kits-modules/`.

NVIDIA Jetson TX2 Specifications	
GPU	NVIDIA Pascal™, 256 CUDA cores
CPU	HMP Dual Denver 2/2 MB L2 + Quad ARM® A57/2 MB L2
Video	4K x 2K 60 Hz Encode (HEVC) 4K x 2K 60 Hz Decode (12-Bit Support)
Memory	8 GB 128 bit LPDDR4 59.7 GB/s
Display	2x DSI, 2x DP 1.2 / HDMI 2.0 / eDP 1.4
CSI	Up to 6 Cameras (2 Lane) CSI2 D-PHY 1.2 (2.5 Gbps/Lane)
PCIE	Gen 2 — 1x4 + 1x1 OR 2x1 + 1x2
Data Storage	32 GB eMMC, SDIO, SATA
Other	CAN, UART, SPI, I2C, I2S, GPIOs
USB	USB 3.0 + USB 2.0
Connectivity	1 Gigabit Ethernet, 802.11ac WLAN, Bluetooth
Mechanical	50 mm x 87 mm (400-Pin Compatible Board-to-Board Connector)

3.2 Software Architecture

The user interface is a simple terminal output that prints out the board state and when the software decides it is the robot's turn to make a move. The main script of the program calls on the different APIs to perform the necessary steps. It starts by calling image detection which uses OpenCV to take in an image of the board and return the board state extracted in form of an array consisting of all the pieces and their locations. The main script then passes this array into the AI portion to determine the next best move to be made. Once a move has been computed, the AI will then pass that information along to the main script which calls the manipulation code that executes the move. The AI also can take the current board state and previous board state to determine whether the human has made a valid move and return the result to the main script.

3.3 Communication Architecture

For the setup defined in this project, all communication between the components is local. The camera module captures images of the board state on the checkerboard using its image sensor. The camera is connected to the NVIDIA Jetson TX2 through a USB peripheral and feeds the image data to it. The object detection program on the Jetson then calls the OpenCV API internally and processes the image to extract the board state in the form of an array. This array is then sent to the AI API which computes the best possible move given the current board state and outputs the result as an array containing the checker piece to be moved and its new location. This output is sent to the movement program which makes calls to the ROS (Robot Operating System) API to manipulate the joints of the robot's arm to execute the movement of the piece on the checkerboard.

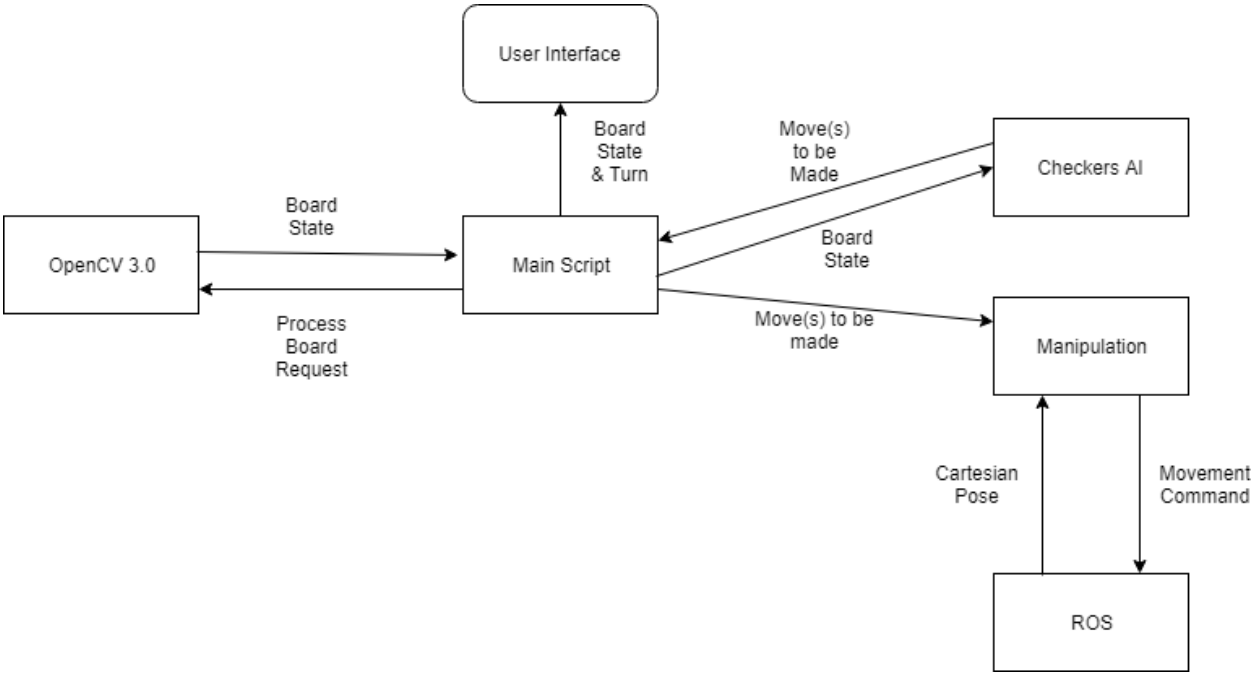

Figure 5: Components used in the software and how they interact with each other

4 System Design

4.1 Sequence Diagrams

4.1.1 Image Detection

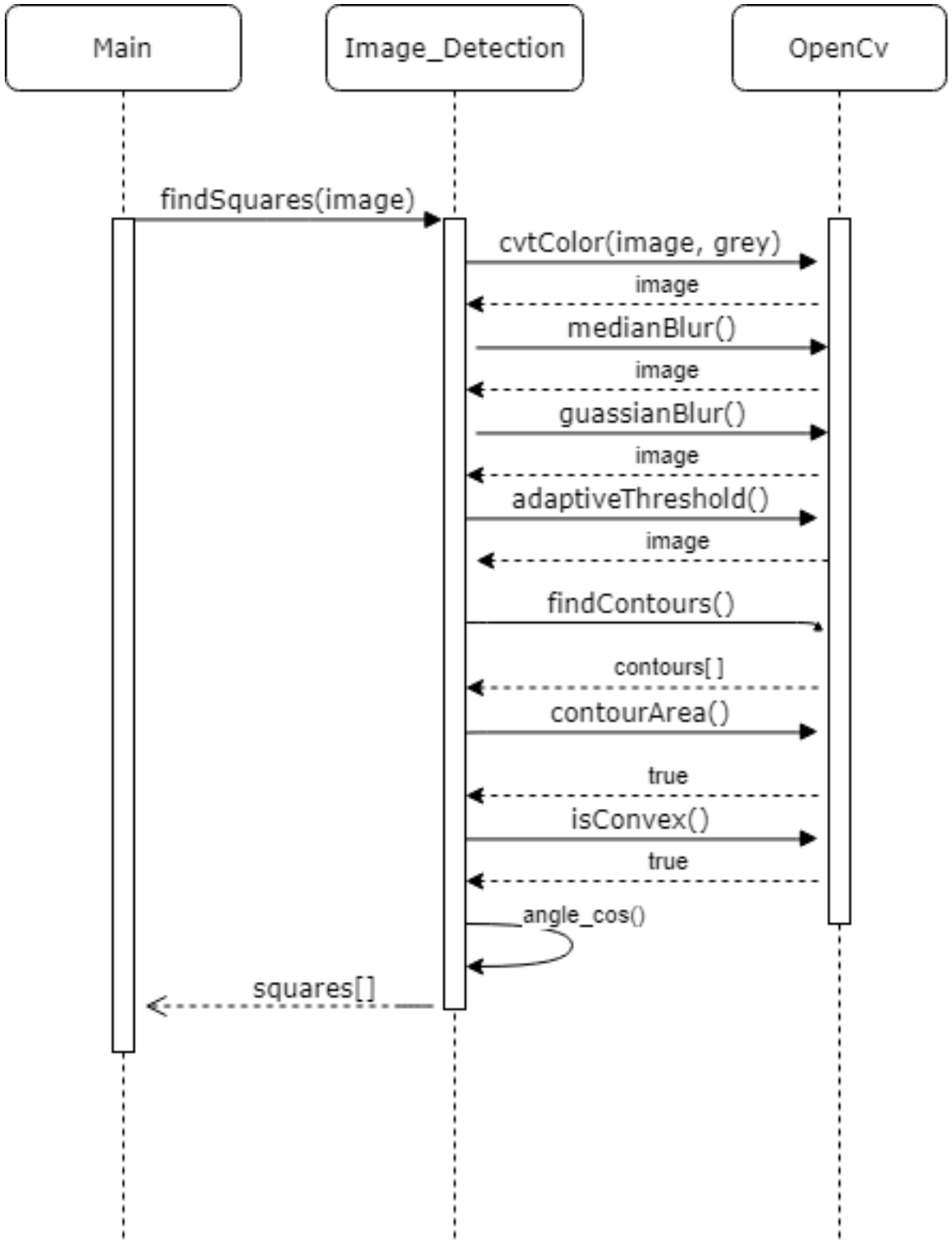

Figure 6: Detecting checkerboard squares

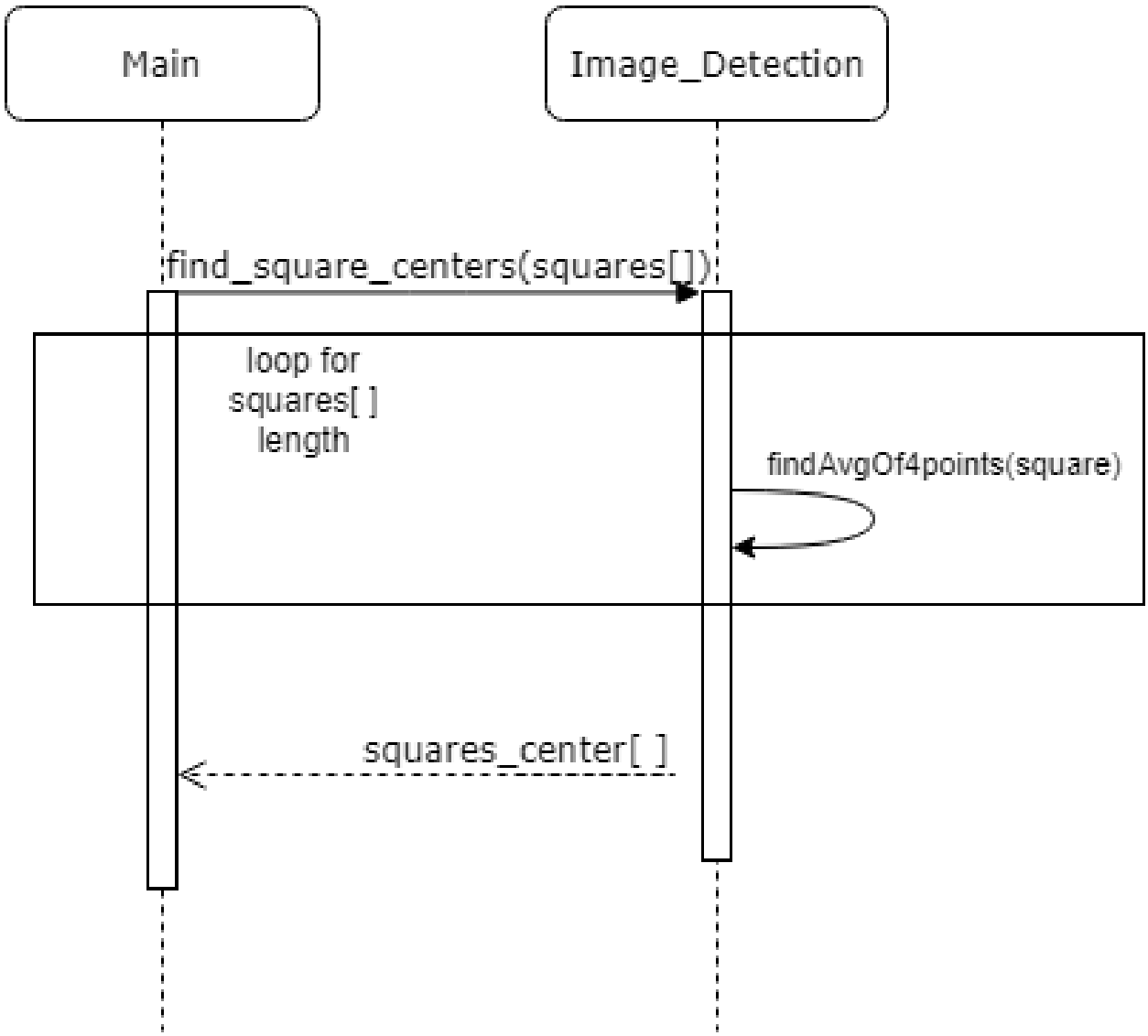

Figure 7: Finding centers of squares

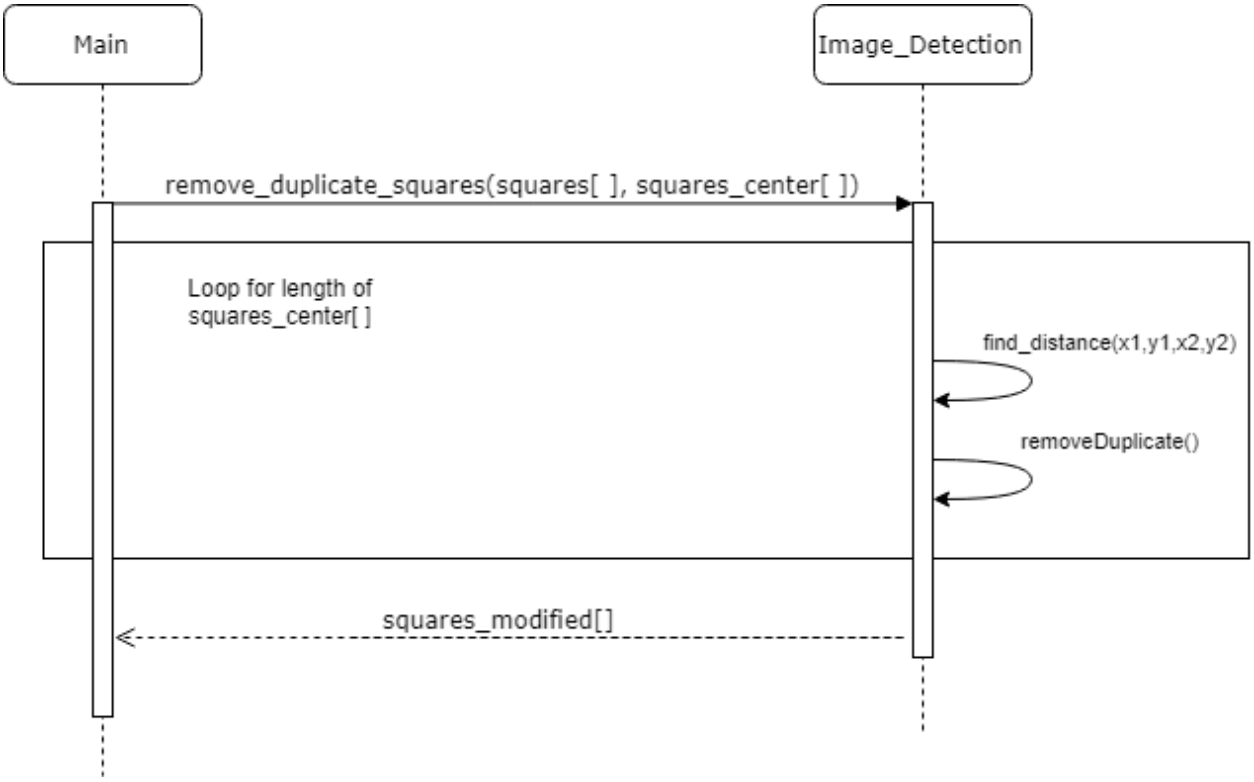

Figure 8: Removing any duplicate squares

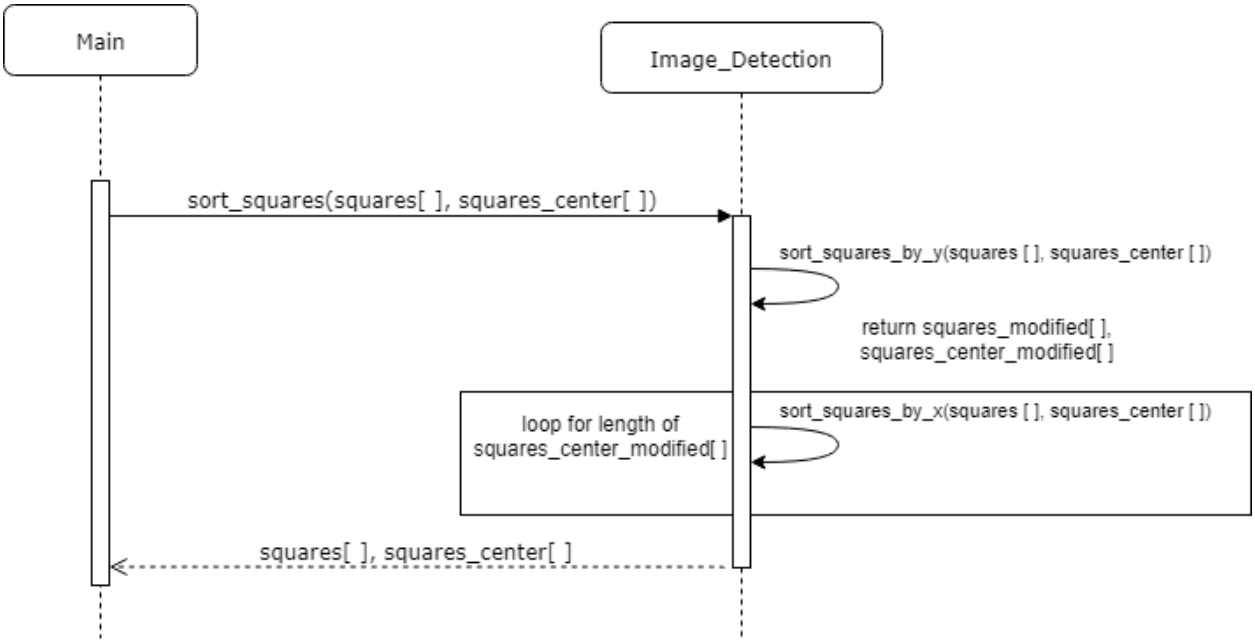

Figure 9: Sorting the squares

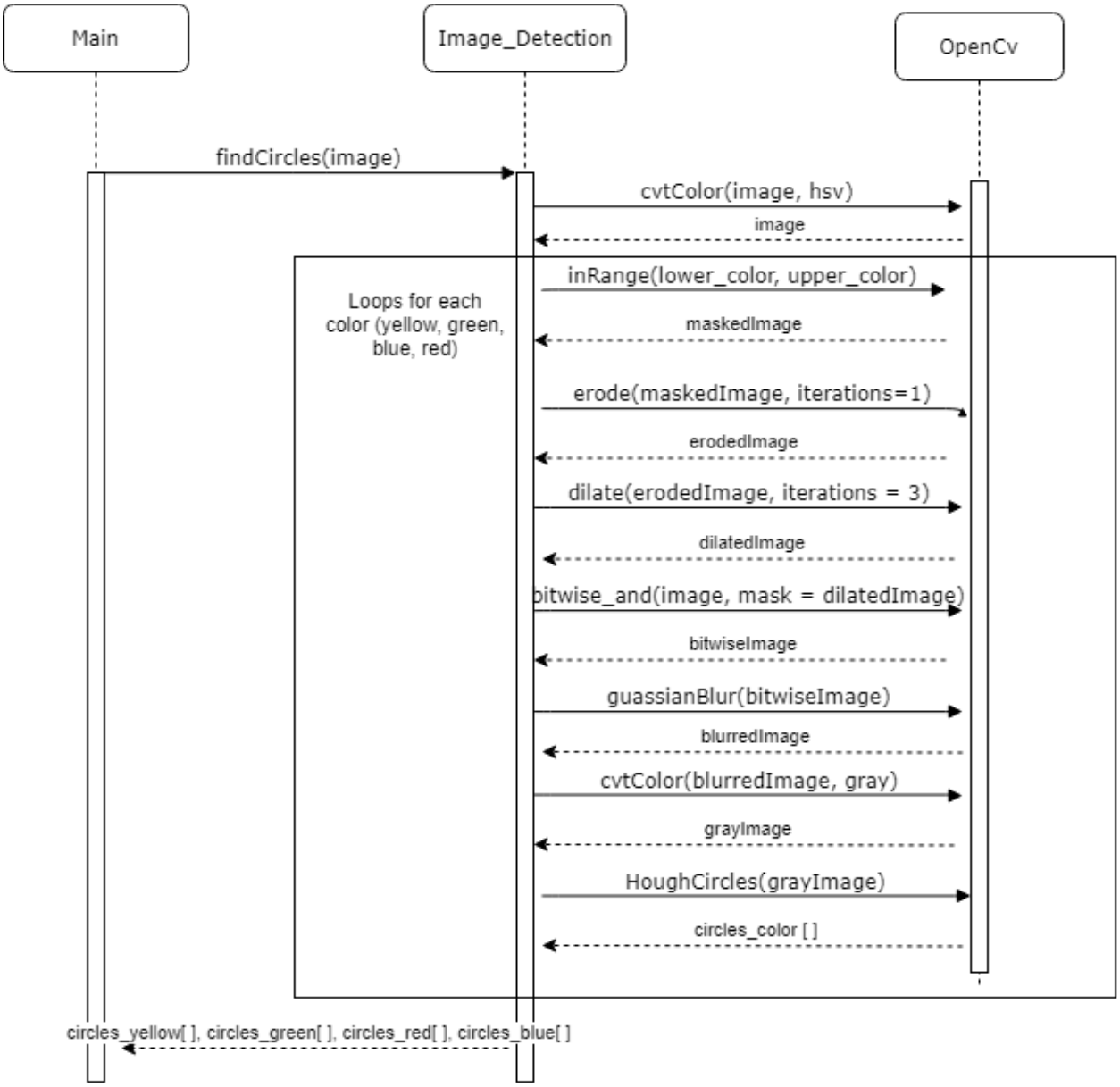

Figure 10: Detecting checker pieces

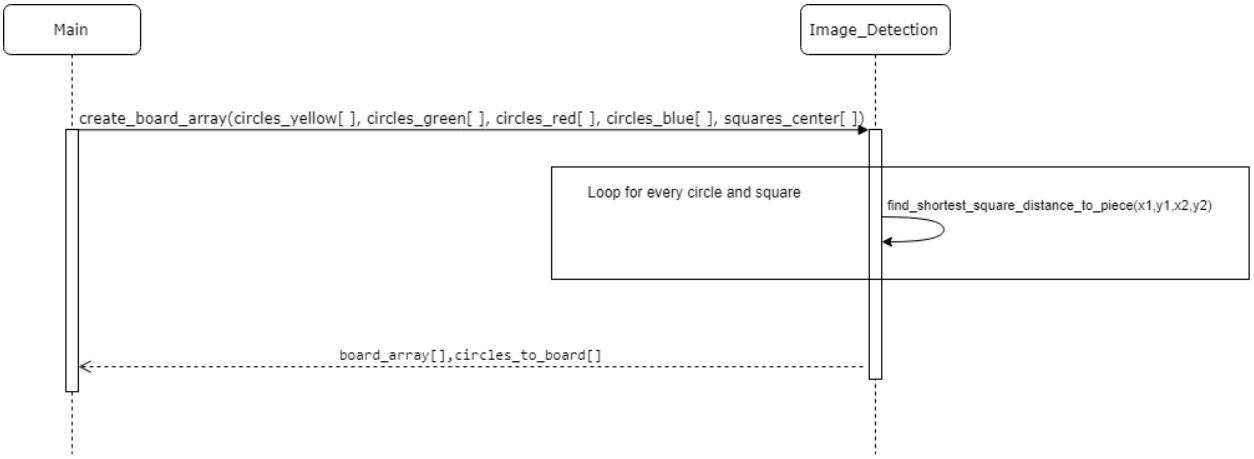

Figure 11: Create board array

4.1.2 Manipulation

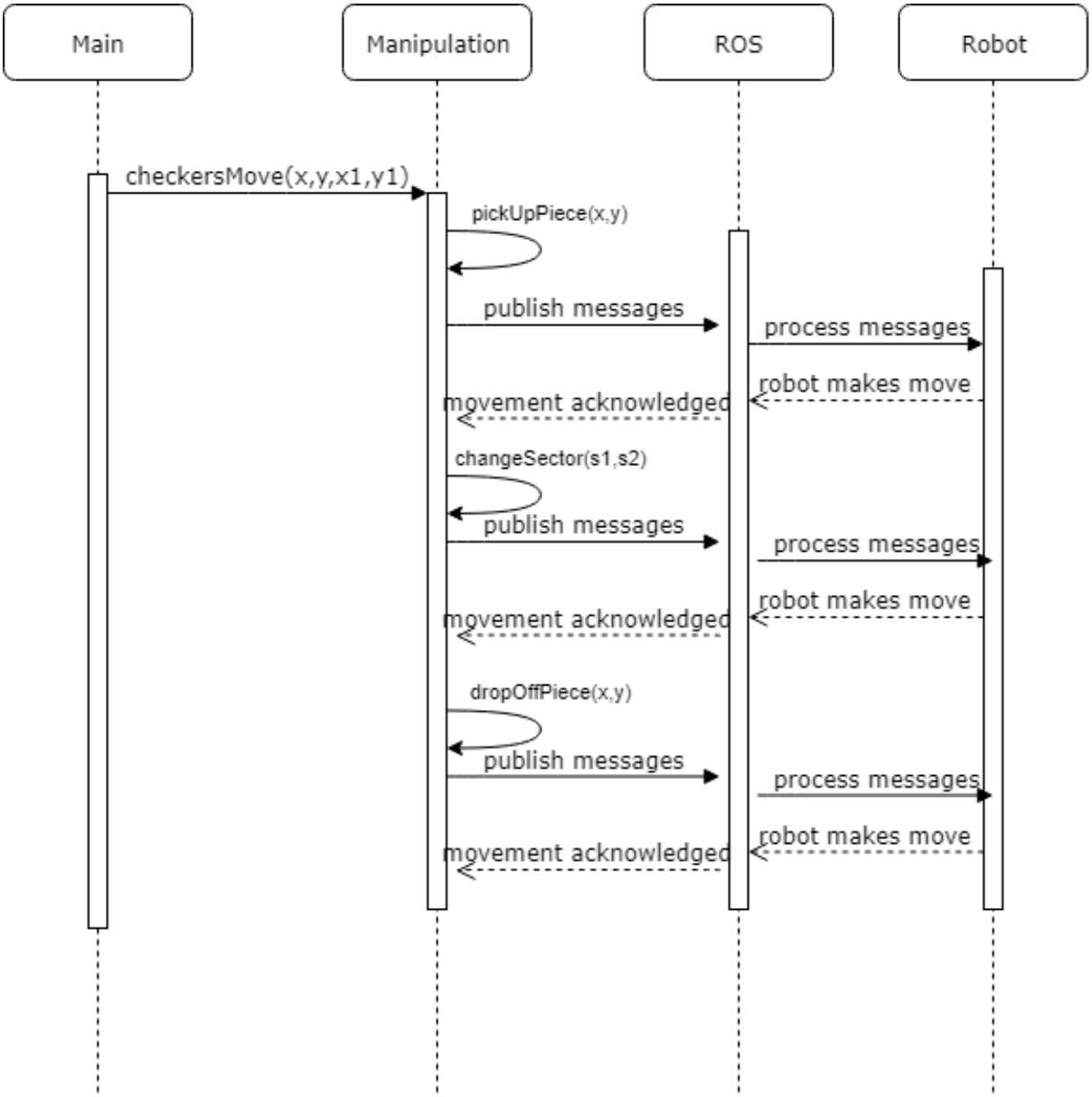

Figure 12: Move checker piece

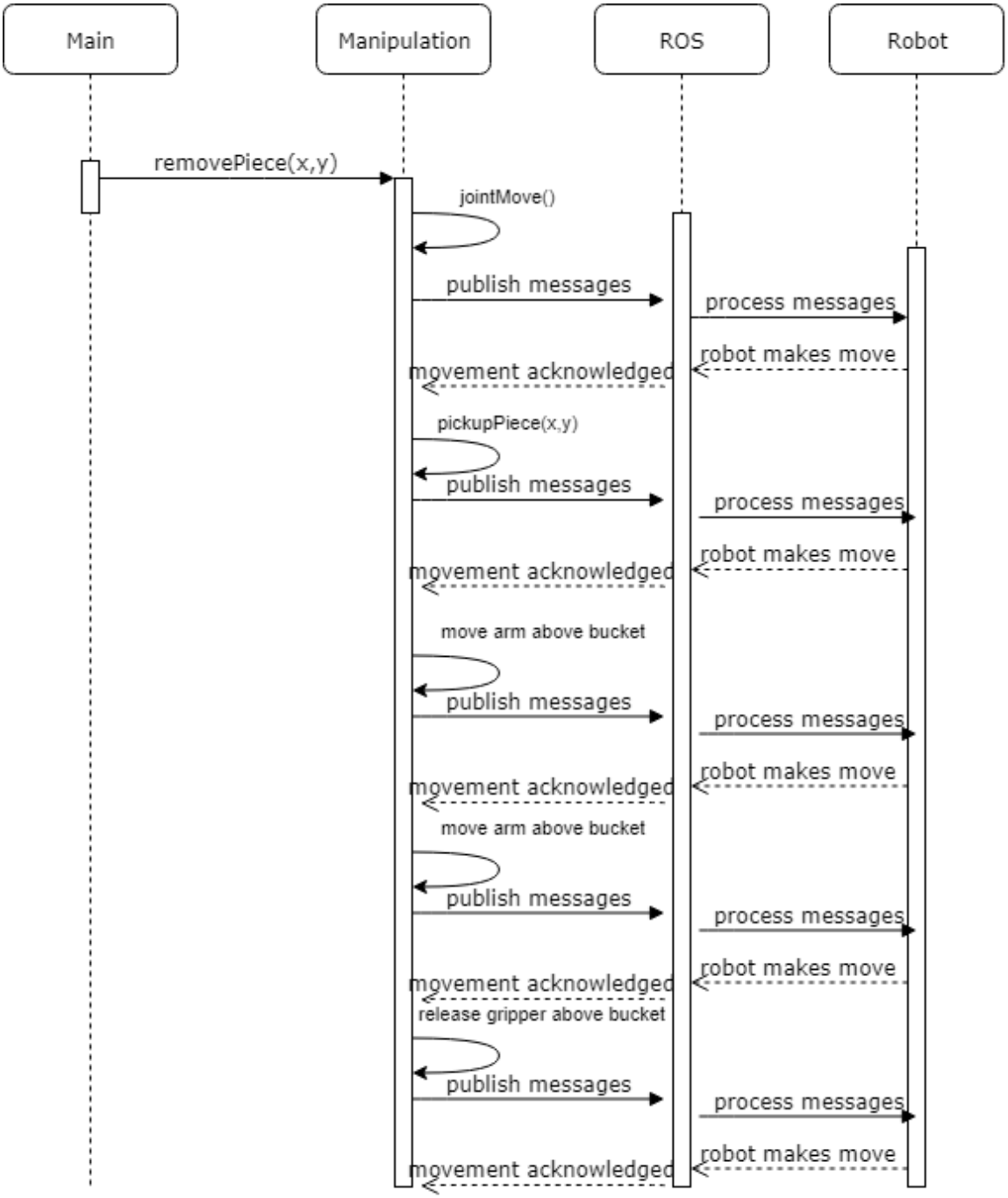

Figure 13: Remove checker piece from board

4.1.3 Integration

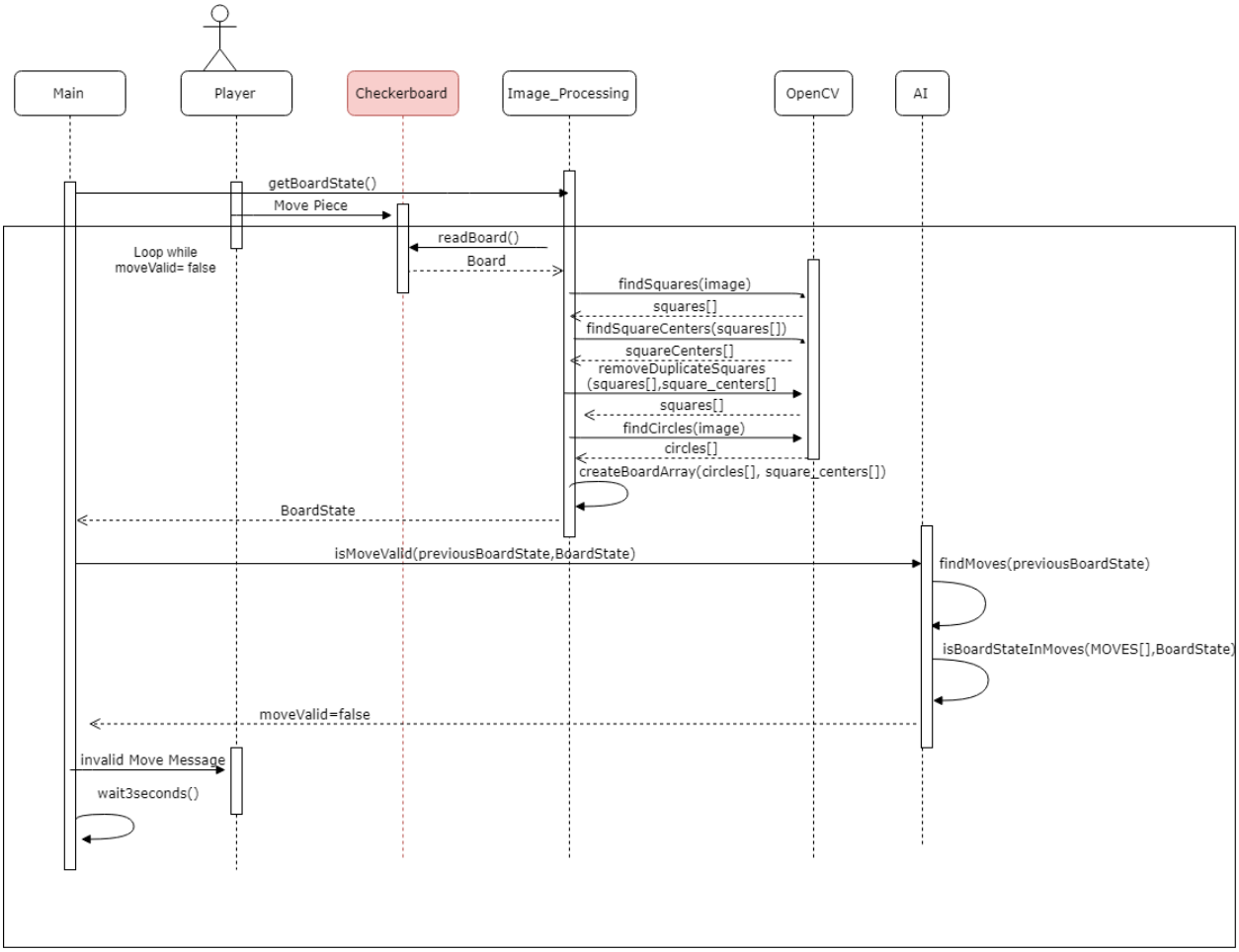

Figure 14: Camera view is obstructed during AI turn

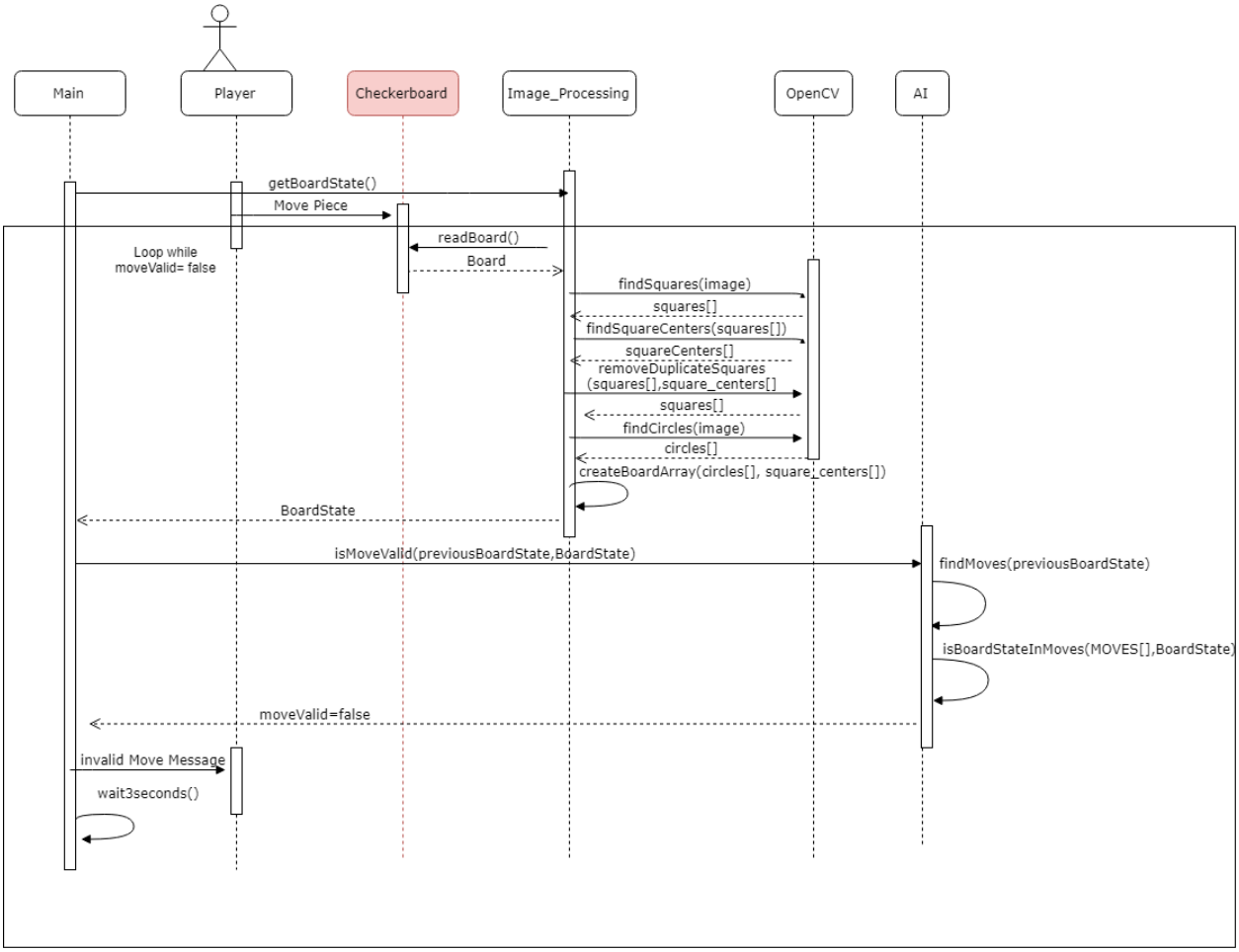

Figure 15: Player has not made their move yet

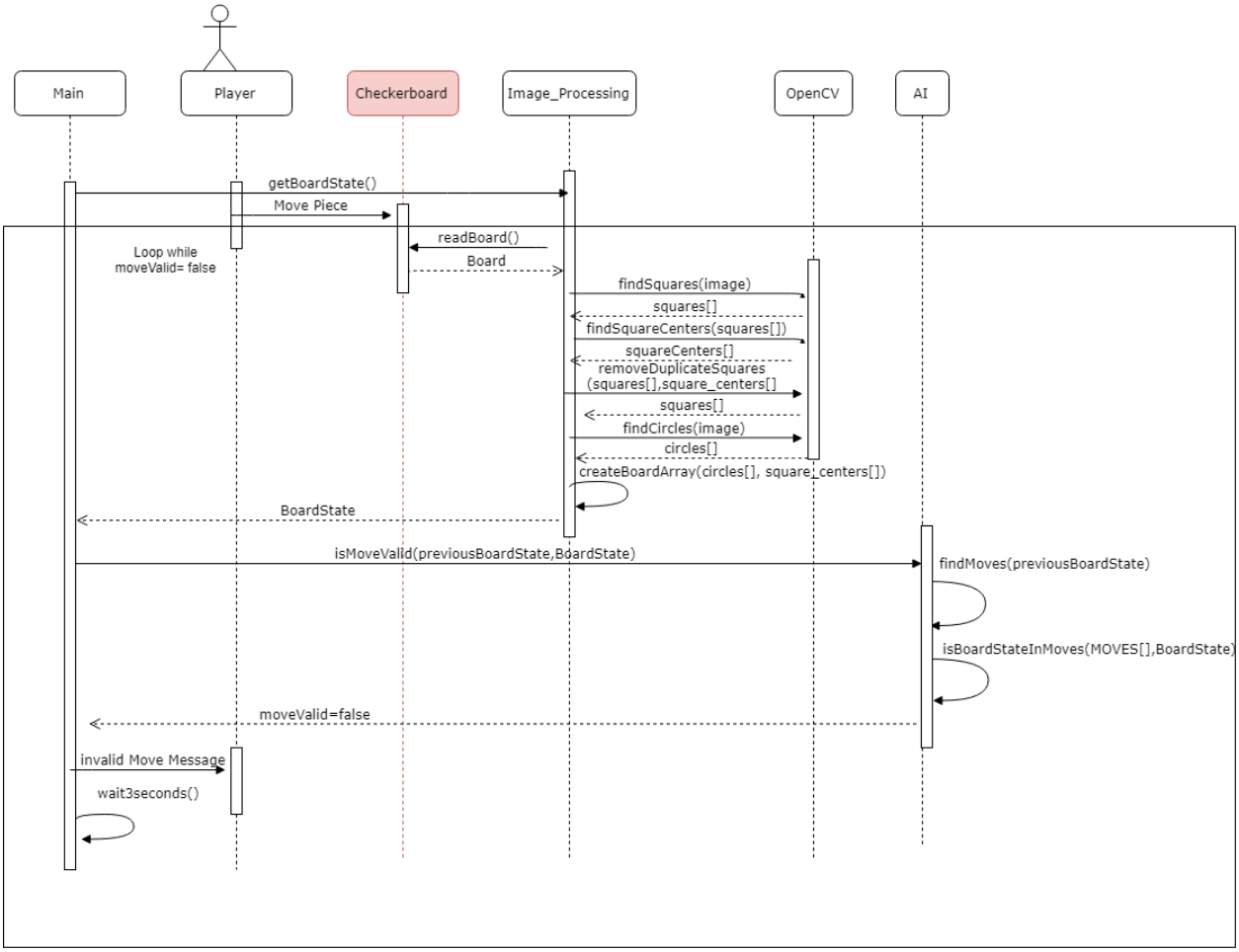

Figure 16: Player has made an invalid move

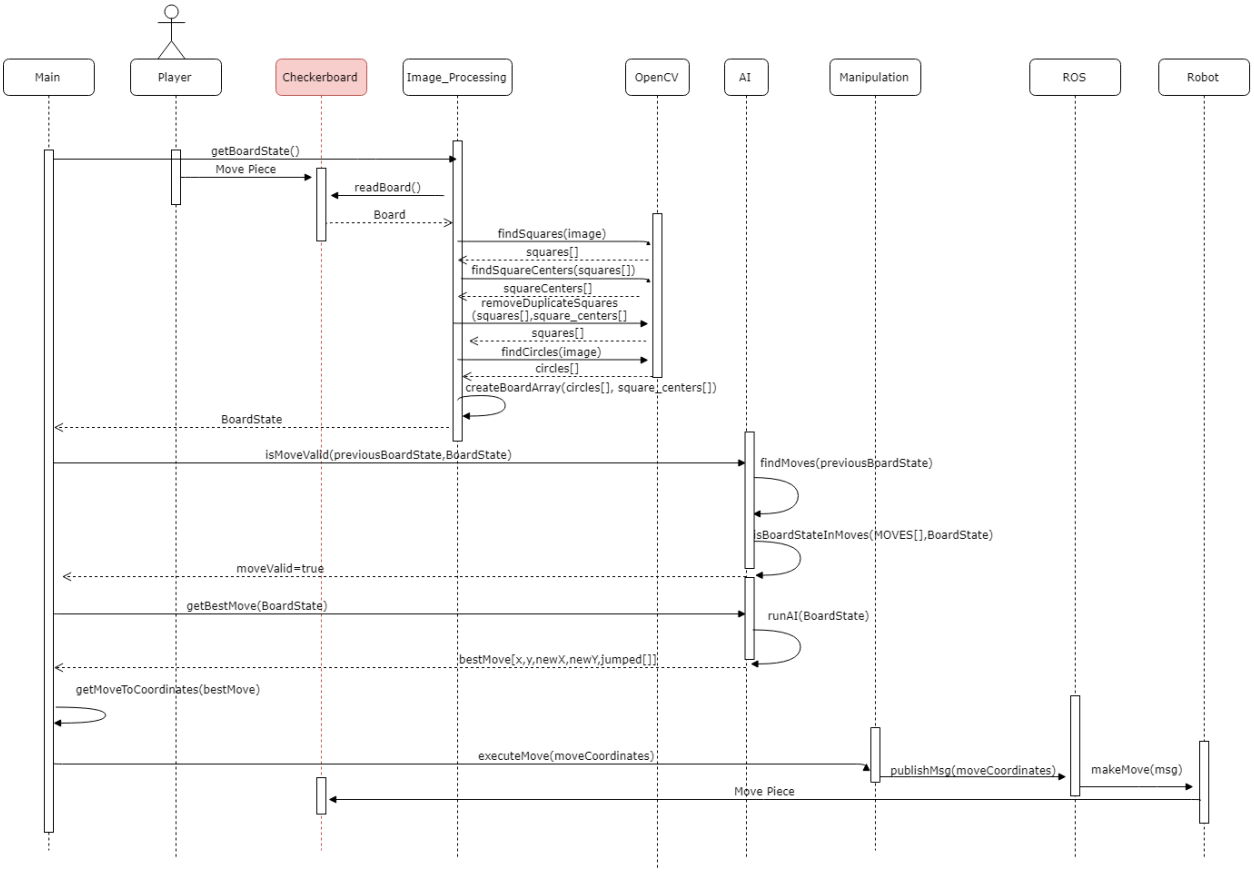

Figure 17: Player has made a valid move

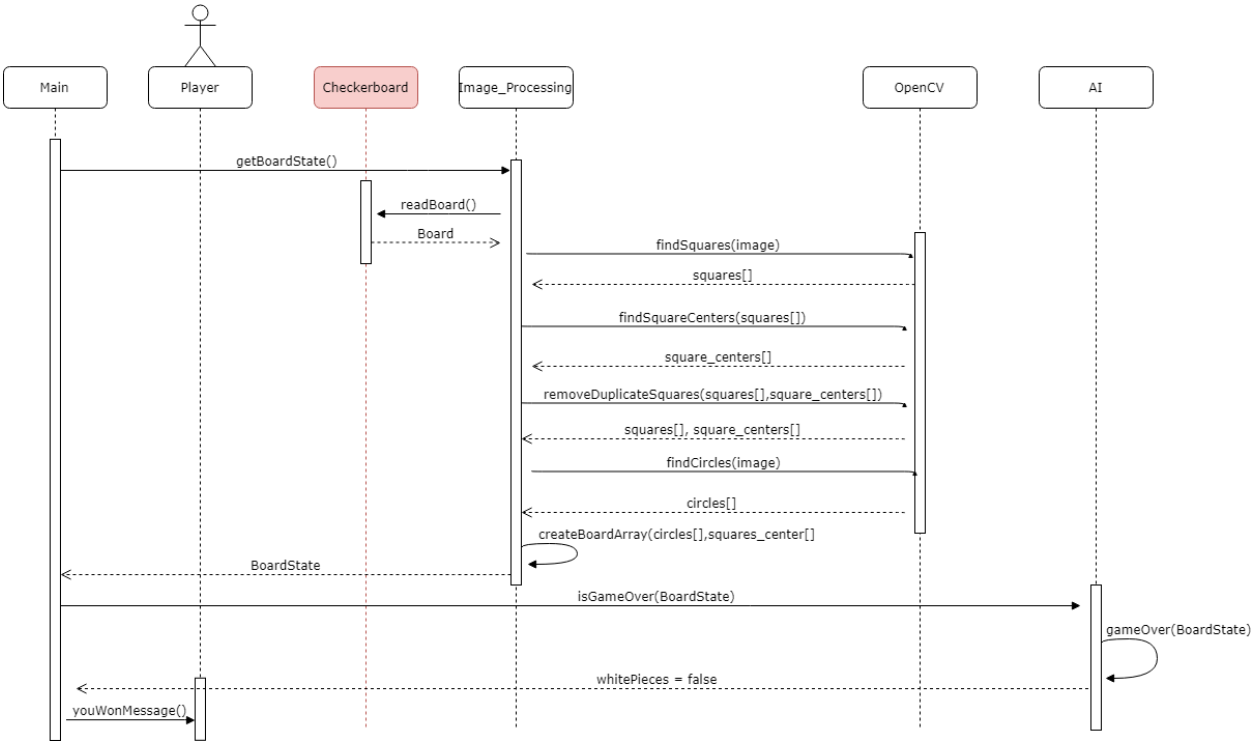

Figure 18: Player has won game

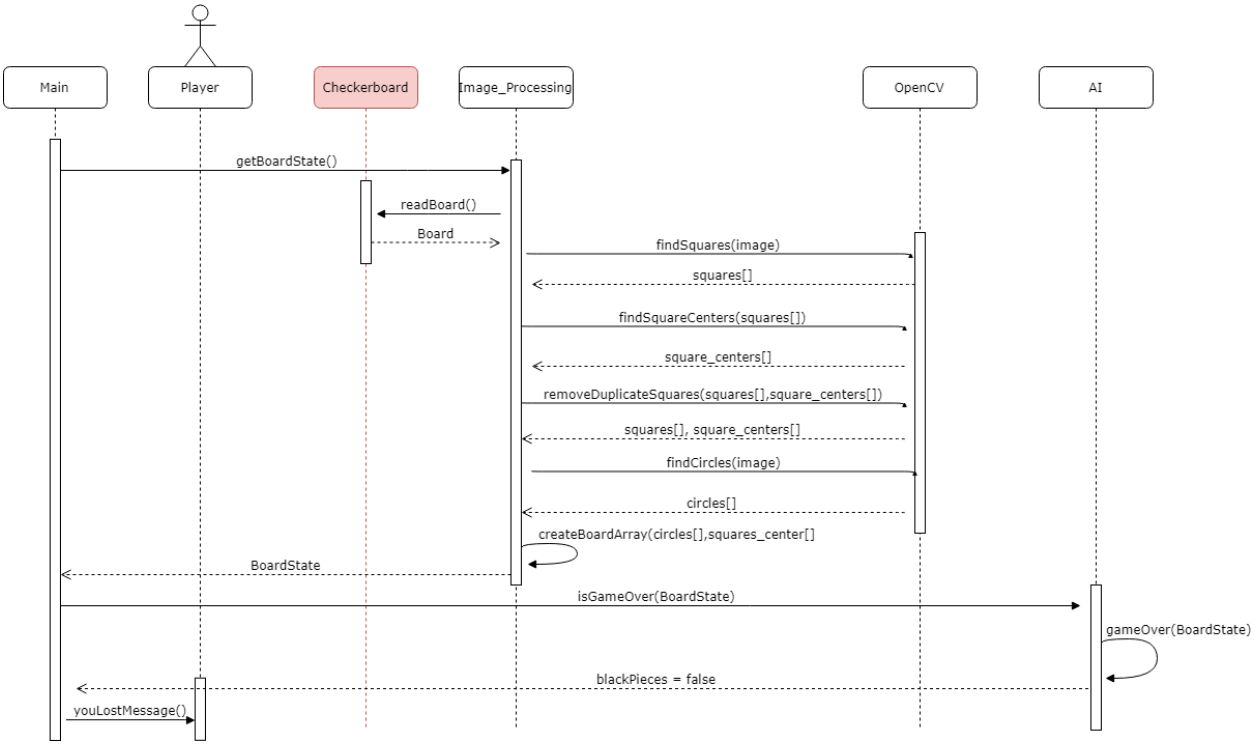

Figure 19: AI has won game

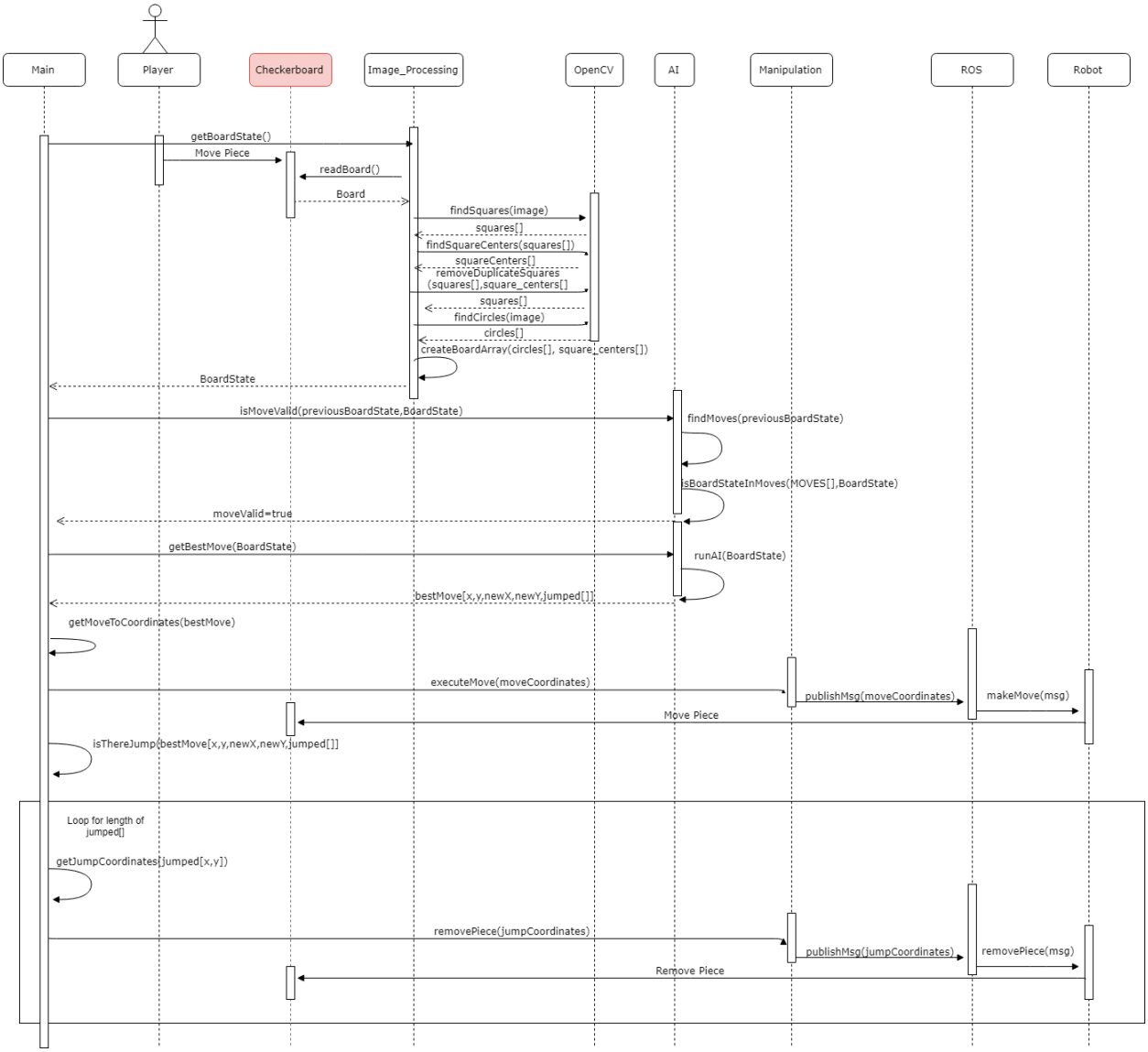

Figure 20: AI kills 1 or more pieces

4.2 Data Flow Diagram

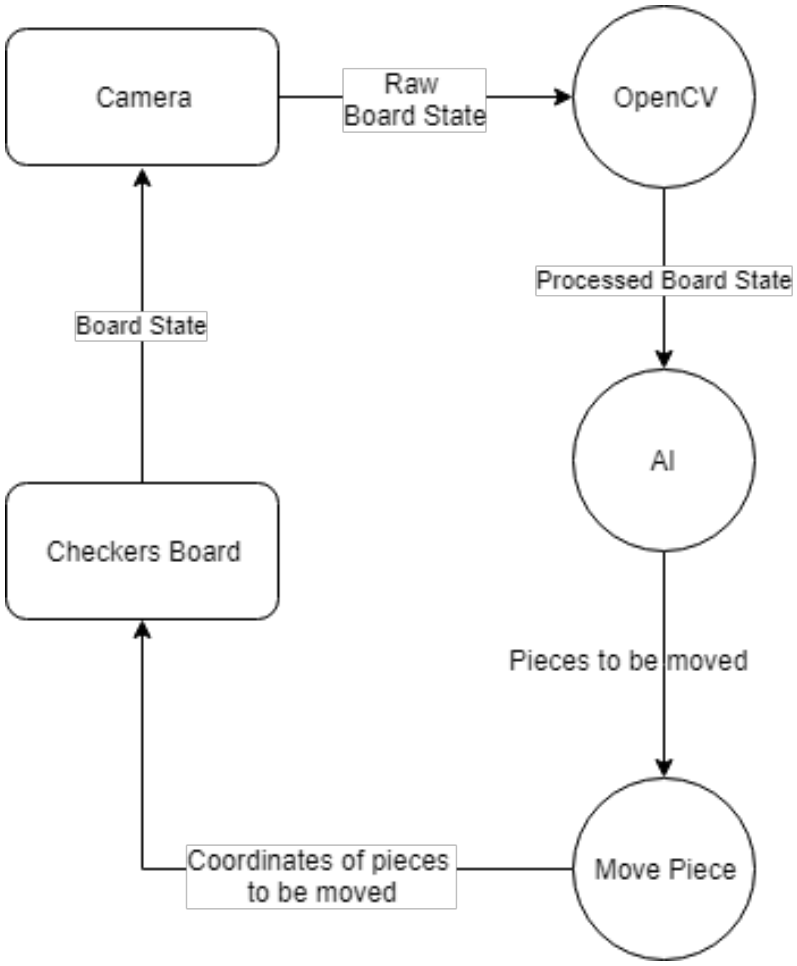

Figure 21: Data flow between the software and hardware components

4.3 Application Program Interfaces

We will be using 3 primary Application Program Interfaces in this program. The first is OpenCV 3, for image recognition. The documentation for OpenCV 3 can be found at: `https://docs.opencv.org/3.0-beta/modules/refman.html`.

4.3.1 OpenCV

Some of the functionality used in OpenCV 3 are:

- cvtColor
- medianBlur
- adaptiveThreshold

- HoughCircles
- findContours
- drawContours
- isContourConvex
- contourArea
- approxPolyDP
- arcLength
- imread
- imshow

4.3.2 ROS/e.DO library

The second API will be ROS. ROS is running on the robot and allows for abstraction of low-level control of the robot. The documentation for general ROS calls as well as the ROS core can be found at: `http://wiki.ros.org/APIs`.

The e.DO Robot is outfitted with its own set of messages and topics found here: `https://github.com/Comau`. The specific repositories referenced in this program are eDO_core and eDO_core_msgs.

e.DO Core:

- ROS Publisher
- ROS Subscriber
- Topic: \bridge_move
- Topic: \cartesian_pose
- Message: MovementCommand
  - uint8 move_command
  - uint8 move_type
  - uint8 ovr
  - uint8 delay
  - uint8 remote_tool
  - float32 cartesian_linear_speed

- edo_core_msgs/Point target
  - uint8 data_type
  - edo_core_msgs/CartesianPose cartesian_data
    - float32 x
    - float32 y
    - float32 z
    - float32 a
    - float32 e
    - float32 r
    - string config_flags
  - uint64 joints_mask
  - float32[] joints_data
- edo_core_msgs/Point via
  - uint8 data_type
  - edo_core_msgs/CartesianPose cartesian_data
    - float32 x
    - float32 y
    - float32 z
    - float32 a
    - float32 e
    - float32 r
    - string config_flags
  - uint64 joints_mask
  - float32[] joints_data
- edo_core_msgs/Frame tool
  - uint8 data_type
  - edo_core_msgs/CartesianPose cartesian_data
    - float32 x
    - float32 y
    - float32 z
    - float32 a
    - float32 e
    - float32 r
  - uint64 joints_mask
  - float32[] joints_data
- edo_core_msgs/Frame frame

* uint8 data_type
* edo_core_msgs/CartesianPose cartesian_data
    - float32 x
    - float32 y
    - float32 z
    - float32 a
    - float32 e
    - float32 r
* uint64 joints_mask
* float32[] joints_data

4.3.3 Checkers AI

The third API will be the checkers AI. The checkers AI is a modified version of an open source checkers AI that can be fond here: `https://github.com/THeK3nger/Cobra-Draughts`. The API will be modified to accept an array input to represent the board state, as well as output what piece should be moved and where it should be moved to.

The input array will consist on numbers ranging from -2 to 2, with -2 representing a black king, -1 a black pawn, 0 an empty spot, 1 a white pawn, and 2 a white king. The output array will fist consist of the place the piece is we want to move followed by the place we want the piece to end up. Any pieces that were jumped, and therefore need removing, will follow. The heuristic will also be modified to incorporate an element of machine learning. Specifically, we will multiply the heuristic by 1 + the win percentage, in decimal form, when this board state is used. We will record the wins and loses of each board state in a separate text file.

4.3.4 Movement API

The fourth and final API is for the Movement of the e.DO arm with respect to the Checkers game, outlined below:

- Move to cartesian position
- Open gripper (Joint 7)
- Pick up piece at position
- Remove piece from play
- Move piece from position to new position
- Turn piece into king
- Get current cartesian pose

4.4 User Interface Design

The User Interface of the application is minimal. It provides the player that is taking their turn, as well as a view of the board as interpreted by the image recognition software. It will display whose turn it is, as well as the state of the board.

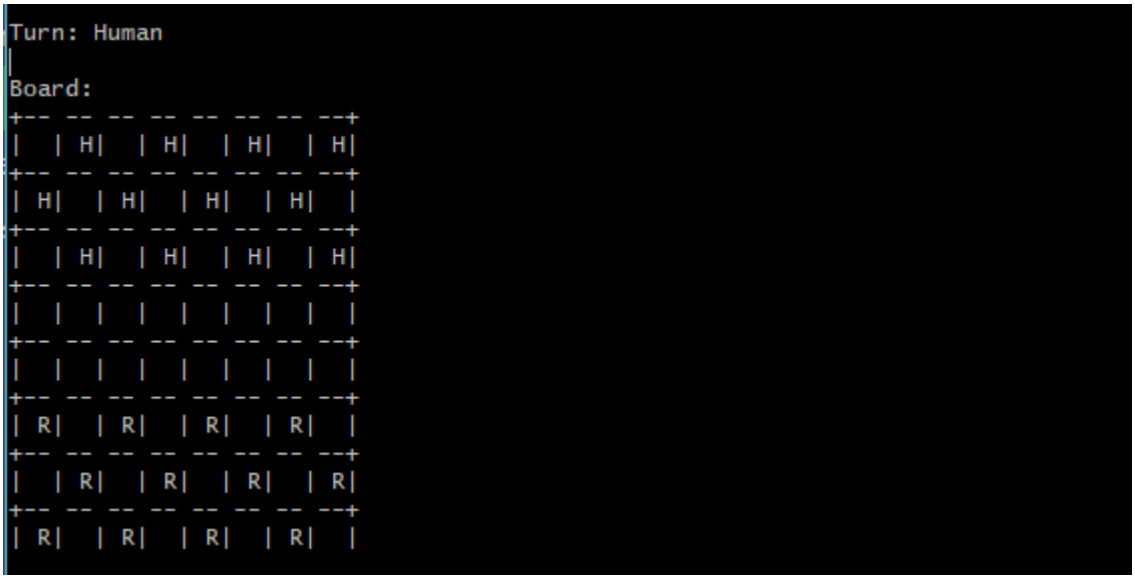

Figure 22: Example of what the terminal output will look like for a simple user interface

The other way through which the User interfaces with the program is by playing checkers on a checkers board, pictured below:

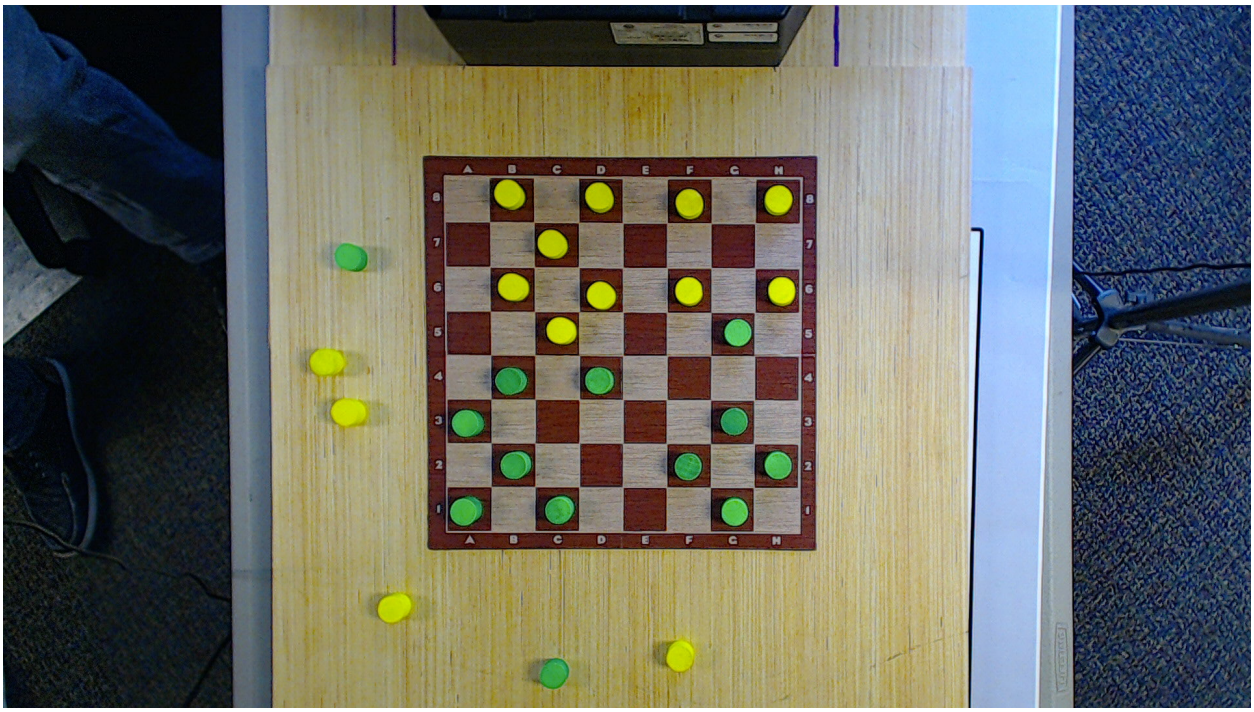

Figure 23: Example of the view the camera will have of the checkers board

5 Product Design Specification Approval

The undersigned acknowledge they have reviewed the e.DO Checkers Product Design Specification document and agree with the approach it presents. Any changes to this Requirements Definition will be coordinated with and approved by the undersigned or their designated representatives.

Signature: _____ Date: _____

Print Name: _____

Title: _____

Role: _____

Signature: _____ Date: _____

Print Name: _____

Title: _____

Role: _____