CHESSTER README word doc

play

BLUF: The play function encompasses all of the functions that enable CHESSTER to play with its human opponents through computer vision and an exchange of information between MatLab and Arduino.

- Establishes variables for connection to Arduino serial port. (sets baud rate, loads default camera positions from given files).

- Prompts user in MatLab to recalibrate camera over the chess board

    - Reads response and calls calibrateCamera function with the default camera positions as inputs or displays blank image.

- Calls startGame function and sets turn to 0

- Establishes serial connection to Arduino and waits at most 10 minutes before moving to next turn.

    - While turn is 0, MatLab communicates to Arduino and displays a standby message. Arduino receives the communication and calls the startPlay function, which changes the standby message on the LCD to a read to play message and send a ready signal back to MatLab.

    - MatLab receives start signal and calls the captureImage function, which sets the default image of the chess board, and displays its own ready message while also advancing the turn.

    - Arduino receives ready signal from MatLab and prompts the user to begin their move, as well as to press the button on the LCD after their move.

    - When the button is pressed, Arduino runs the signalMove function which displays a loading message on the LCD screen and sends a play signal to MatLab.

    - When MatLab receives the play signal, it causes the camera to take another image of the board with captureImage, then it runs computer vision by calling the camPassForward function which takes the original image of the board, new image of the board, and startGame functions to run. MatLab then runs the engine for Chesster using the engPassForward function. Moves made are translated and sent to Arduino using json and sent to Arduino.

    - Move is received by Arduino, and displays a confirmation message on the LCD, while communicating with MatLab as well.

    - Arduino makes a decision based on kill values sent with the move data and displays that the move was received and whether a piece was taken with its move, finally displaying a move complete message (sending another message to MatLab) and prompting the user to make another move and advancing the turn, otherwise an error message will display.

    - As MatLab receives confirmation, it displays the acknowledgement and the move made using the showOff function, as well as advancing the turn. When the move complete message is received, MatLab runs the updateGame function and takes another picture of the board using the captureImage function.

    - As the player makes another move, the process repeats itself.

Computer Vision

captureImage

BLUF: The captureImage starts the camera and saves images taken to the landing directory, as well as identifying the image for use by other functions later.

Inputs: camDir, landingDir, camKIllBatDir (Directories for starting and stopping the camera, as well as storing the images.

Outputs: Images

- Runs the initiateCamera function with the camera directory as the input.
- Identifies the open directory folder as the original directory, then sets a stat variable as 1.
- While loop that runs when stat = 1, changes open folder to the landing directory folder (where pictures go), sets a search variable “a” to identify any objects in the directory with a .bmp file identifier. Sets variable “n” as the number of objects identified by the “a” variable.
    - If statement, that initiates when the number of objects in the landing directory is greater than 9, which in turn runs the kill function to stop the camera from taking photos. If successful, then a script will print stating it executed the kill, otherwise it will display an error message.
    - Stat is set to 0, ending the while loop.
    - Switches open folder to the landing directory in case it is not, sets the image to the 6th image taken (higher resolution)
    - Removes images that it did not identify in the previous step from the landing directory.
- Sets open folder back to the original directory.

initiateCamera

BLUF: Starts camera

Inputs: camDir (Directory that contains the command to start the camera)

Outputs: Camera starts

- Sets open folder as the original directory.
- Try (Executes code)
    - Sets open folder as the input directory from **captureImage**.
    - Sets the command variable to the code that starts the camera
    - Tells Command Prompt to run the start command.
    - Sets open folder to the original directory
    - Displays a message when command is sent.
- Catch (Catches errors)
    - Sets open folder to the original directory
    - Displays error message.

calibrateCamera

BLUF: Captures image of blank board and ensures camera is working properly.

Inputs: camDir, landingDir, camKIllBatDir (Directories for starting and stopping the camera, as well as storing the images.

Outputs: Calibrated image of the blank board

- Sets confirmation variable to a default, while default user will be prompted to “press any button when ready to calibrate”
    - When a button is pressed, and the ready variable is changed, an if statement is executed, setting the blank image as a variable, that uses the output of the **captureImage** function.
        - squareOffTestFig function is executed with the blank image as the input
        - User is prompted to press “Y” to confirm calibration or “N” if a new image needs to be taken. This input replaces the previous confirmation variable.
        - A full file specification using the landing directory folder and the 6.bmp image from the folder is set to a current image name variable.
            - A second if statement is executed if the user answers the calibration prompt with “Y”.
            - A blank.bmp filename is created for the landing directory folder and set as the new image name variable.
            - The current image name is changed to the new image name “blank.bmp”
            - The blank image variable is reset to the blank.bmp image, and a message is sent confirming the name change.
    - All files excluding the newly named .bmp are cleared from the landing directory folder.

squareOffTestFig

BLUF: Takes blank board image, squares it off and makes it greyscale.

Inputs: Initial image of the blank board

Outputs: Calibrated image of the blank board

- Calls sobelMask function, with the blank board image as an input and with a very high threshold in order to outline the entirety of the board.
- Calls the squareOff function with the blank board as the input and isolates the board.
- Displays the three new images in a figure for the user to see.

sobelMask

BLUF: Finds edges and generates an edge map image of the board and squares within.

Inputs: Initial blank image of the board, color threshold fo identifying pixels as black or white.

Outputs: Image of the identified white areas on the board.

- Calls greyScale function in input image, normalizing color values between 0 and 1
- Defines the “Mask” matrix to be used in the Sobel.
- Convolutes the mask matrix over the image matrix and returns a convolution matrix, then calculates the magnitude of the convolutions.
- Stores the magnitudes as 1-bit integers as the output of the function.

greyScale

BLUF: Normalizes an image (0 - white, 1 – black) *Figure out numbers in function

Inputs: Initial blank image of the board

Outputs: Grayscale image of the blank board.

- The input image's colors are reassigned as either black or white depending on which they are closer to in color, and assigned to the output variable.

squareOff

BLUF: Square off input image and isolate Board based on calculated bounds.
Inputs: Initial blank image of the board
Outputs: Image of the board after identifying and applying the boundaries of the board in the image.

- Sets the bounds to a variable bound that calls the boardBounds function using the input image to the squareOff function as the same input.
- Sets the x and y bounds of the new squared image as the bounds found in the boards function.

boardBounds

BLUF: ID edges of the board, starting at the center of the sobelmasked image and finds the coordinate of the first edge in each direction with a high threshold, then trims the image to be square.

Inputs: Initial blank image of the board

Outputs: Greyscale image of the board with boundaries applied after identification through use of the Sobel mask.

- Calls the sobelMask function, with the input image and a high threshold.
- Determines the size of the sobelmasked image, then finds the center of it.
- Initiates four while loops that move in each direction until an edge is found
- Coordinates of the edges are put in to a matrix as the output.

camPassForward

BLUF: Passes forward the updated game matrix to the engine, using the blank board image, the board image before the move, an image of the board after the move, and a matrix that represents the original image as inputs.

Inputs: Calibrated image of the blank board, an image of the set board before a move, an image of the set board after a move, and the game matrix that represents the board state before a move.

Outputs: Updated game matrix that represents the board after a move.

- Calls the findMove function to calculate what on the board has changed.
- Calls the updateGame function to update the game matrix.

findMove

BLUF: Takes blank image of board, the image of the board before the move, and the image of the board after the move and determines what move was made. *Change variables around, check functionality.

Inputs: Calibrated image of the blank board, an image of the board before a move, and an image of the board after a move.

Outputs: Coordinates of the piece before and after it has moved.

- Calls the moveLoc function to find the starting location of the piece and the ending location of the piece, then assigns them to a matrix as the output.

  moveLoc

  BLUF: Analyzes the input images of the original board and the board with the move made, and determines what move occurred.

  Inputs: Calibrated image of the blank board, an image of the board before a move, and an image of the board after a move.

  Outputs: Coordinates of the piece being moved in both images and an associated confidence.

  - Finds the area in the images where the numerical values in the matrixes differ
  - The difference is run through the sobelMask function
  - The sobelmasked image is run through the squareOff2 function using the blank board bounds.
  - The square bounds of the blank board are then found using the squareOff function, and the sqaureBounds function, which is then assigned as an 8x8 matrix.
  - A score is then calculated that gauges the confidence of the move detected by the series of functions.

    squareOff2

    BLUF: Square off image and isolate Board based on calculated bounds from a blank board

    Inputs: Initial blank image of the board, and the Sobel mask of the difference between the images of the board before and after a move.

    Outputs: Image of the board after identifying and applying the boundaries of the board in the image.

      - Sets the bounds to a variable bound that calls the boardBounds function using the input image to the squareOff function as the same input.
      - Sets the x and y bounds of the new squared image as the bounds found in the boardBounds function.

    sqaureBounds

    BLUF: Creates bounds for the squared image input.

    Inputs: Squared image of the blank board

    Outputs: Bounds for each square on the chess board.

      - Creates dimensional increments of 8 from left to right and top to bottom.
      - Creates vectors of these bounds, then corrects them.
      - Creates an array that stores the squares in the vicinity of each other.

showOff

BLUF: Plots images of the board before the move and after the move, as well as the results of the computer vision stage.

Inputs: Calibrated image of the blank board, an image of the set board before a move, an image of the set board after a move, and the game matrix that represents the board state before a move.

Outputs: Figure that shows side by side images of the board before and after a move, and a digital version of the change the computer vision recognized on the board.

- Calls the findMove function with images of the blank board, board before the move and the board after the move as inputs.
- Assigns the pieces on the board coordinates for both the original board and the moved board using the mapToChess function.
- Calls the updateGame function using the game matrix and the move found.
- Displays a plot of the original board before the move and another of the board after the move, as well as the original and new game matrixes.

mapToChess

BLUF: Converts the matrix coordinates of the board to chess locations.

Inputs: Coordinates of the piece moved in the game matrix

Outputs: Location of the piece on the chess board in corresponding terms

  - Assigns the matrix coordinates a chess board position (i.e. knight to a4)

displayBoard

BLUF: Creates a figure of the digital representation of the current board give the game matrix. *Does not use the output in the function*

Inputs: The game matrix

Outputs: Figure that represents the current state of the board

- Calls the checkerBoard function and assigns it to a variable.
- Creates a vector of 8 evenly spaced points from 1 to 64 and an 8x8x2 position matrix of zeros.
- Assigns positions to four quadrants of the chess board individually, then compiles them into an image.

checkerBoard

BLUF: Creates a blank checkerboard pattern for plotting.

Inputs: None

Outputs: Blank checkerboard patter to be used to plot a digital representation of the chess board.

  - Creates a matrix of ones to represent white, and a matrix of zeros to represent black, and assigns them to an overall matrix for the output.

updateGame

BLUF: Updates the game matrix based on the move made.

Inputs: The current game matrix and the coordinates of a piece before and after a move.

Outputs: Updated game matrix for the board after a move has been made.

- Assigns the original position of the piece to x and y variables as well as the new location of the piece.
- Uses the position of the piece and identifies both its type and color using the game matrix as the reference.
- Overwrites the original position of the piece in the game matrix, and sets the new location of the piece in the game matrix.

startGame

BLUF: Creates the game matrix, with pieces at starting locations identified by both color and type.

Inputs: None

Outputs: Original game matrix of a set board before any move has been made.

- Assigns a color and piece matrix for the starting board state.

Engine

engPassForward

BLUF: Generates a recommended move and translates it to steps for the motor using the game matrix and turn number as inputs and outputting piece and move locations, as well as info on whether a piece has been taken.

Inputs: Current game matrix and the current turn number.

Outputs: Positions of a piece before and after a move as well as information regarding whether a piece is taken.

- Calls the engine function with the game matrix and turn number as inputs.
- Takes the recommended move and separates it into the coordinates of the piece before and after the move.
- Size parameters are then set for the engine, in reference to the board as well as for the motor's rotation.
- Sets kill conditions for when one player's piece overtakes another.

engine

BLUF: Takes in the game matrix and turn, puts them through several layers of a decision tree, returning an array with the information for a recommended move.

Inputs: Current game matrix and the current turn number.

Outputs: A cell array with all relevant information for a recommended move.

- Creates the first layer of the decision tree using the firstLayer function, and a second using the secondLayer function with the first layer as an input.
- Creates a weighted move matrix using the firstLayerOutputWeighted with both layers.
- Attributes a score to the weighted matrix and creates a set of recommended moves.

firstLayer

BLUF: Uses the game matrix and turn as inputs, outputting an 8x8 array, where each cell contains the moves the piece in that position can perform. *Nested 8x8 in each part of the output array.*

Inputs: Current game matrix and the current turn number.

Outputs: Cell array with all available moves for the pieces on the board.

- Determines the color of the piece based on the current turn.
- Calls the initFirstLayer function to create an empty move matrix as a shell for the first layer.
- Populates the first layer by running a for loop through the entire game matrix, and assigning moves from the pawnMoves, rookMoves, bishopMoves, knightMoves, queenMoves, and kingMoves based on the piece in each cell.

initFirstLayer

BLUF: Creates the first layer for the engine.

Inputs: None

Outputs: Empty cell array for use in the firstLayer function.

- Assigns the initialization as an 8x8 cell array
- Sets the values of the array as zeros.

secondLayer

BLUF: Creates the second layer of the decision tree for the engine, using the game matrix, turn, and the first layer as inputs.

Inputs: Current game matrix, the current turn number, and the first layer cell array.

Outputs: Cell array of all possible moves for the user to make in response to any move possible to CHESSTER in the first layer.

- Determines the color of the piece based on the current turn.
- Calls the movedGame function with the first layer and the game matrix as inputs in order to create an input layer of all possible moved games that can occur as a result of the first layer.
- Calls the initSecondLayer function to create an empty move matrix as a shell for the second layer.
- Populates the second layer by running a for loop through the entire game matrix, and assigning moves from the pawnMoves, rookMoves, bishopMoves, knightMoves, queenMoves, and kingMoves based on the piece in each cell.

movedGame

BLUF: Creates a cell array of all moved games that can occur as a result of first layer moves to be used as inputs for the second layer, uses the first layer and the game matrix as inputs.

Inputs: Current game matrix and the first layer cell array.

Outputs: Cell array of all moved games that can occur as a result of first layer moves.

- Creates an empty cell array and populates it with data from the game matrix and first layer, showing all possible game movements from the game matrix to the first layer.

initSecondLayer

BLUF: Initializes the second layer for the engine using the output from the movedGame function as its input.

Inputs: movedGame cell array

Outputs: An empty input layer cell array.

- Assigns the initialization as an 8x8 cell array
- Sets the values of the array as zeros.

pawnMoves

BLUF: Uses the game matrix, and the position of the piece in the matrix, as inputs to create an 8x8 showing possible moves for pawns.

Inputs: Current game matrix and the coordinates of pawns in the given cell array.

Outputs: Coordinates in the cell array of where each pawn could move legally on the board.

- Checks the color of the piece.
- Creates an 8x8 blank matrix for possible moves.
- Assigns if statements for the conditions of the board, to calculate what legal moves can be made for a pawn.

rookMoves

BLUF: Uses the game matrix, and the position of the piece in the matrix, as inputs to create an 8x8 showing possible moves for rooks.

Inputs: Current game matrix and the coordinates of rooks in the given cell array.

Outputs: Coordinates in the cell array of where each rook could move legally on the board.

- Checks the color of the piece.
- Creates an 8x8 blank matrix for possible moves.

- Assigns if statements for the conditions of the board, to calculate what legal moves can be made for a rook.

bishopMoves

BLUF: Uses the game matrix, and the position of the piece in the matrix, as inputs to create an 8x8 showing possible moves for bishops.

Inputs: Current game matrix and the coordinates of bishops in the given cell array.

Outputs: Coordinates in the cell array of where each bishop could move legally on the board.

- Checks the color of the piece.
- Creates an 8x8 blank matrix for possible moves.
- Assigns if statements for the conditions of the board, to calculate what legal moves can be made for a bishop.

knightMoves

BLUF: Uses the game matrix, and the position of the piece in the matrix, as inputs to create an 8x8 showing possible moves for knights.

Inputs: Current game matrix and the coordinates of knights in the given cell array.

Outputs: Coordinates in the cell array of where each knight could move legally on the board.

- Checks the color of the piece.
- Creates an 8x8 blank matrix for possible moves.
- Assigns if statements for the conditions of the board, to calculate what legal moves can be made for a knight.

queenMoves

BLUF: Uses the game matrix, and the position of the piece in the matrix, as inputs to create an 8x8 showing possible moves for queens.

Inputs: Current game matrix and the coordinates of queens in the given cell array.

Outputs: Coordinates in the cell array of where each queen could move legally on the board.

- Checks the color of the piece.
- Creates an 8x8 blank matrix for possible moves.
- Assigns if statements for the conditions of the board, to calculate what legal moves can be made for a queen.

kingMoves

BLUF: Uses the game matrix, and the position of the piece in the matrix, as inputs to create an 8x8 showing possible moves for kings.

Inputs: Current game matrix and the coordinates of kings in the given cell array.

Outputs: Coordinates in the cell array of where each king could move legally on the board.

- Checks the color of the piece.
- Creates an 8x8 blank matrix for possible moves.
- Assigns if statements for the conditions of the board, to calculate what legal moves can be made for a king.

firstLayerOutputWeighted

BLUF: Uses the game matrix, first layer, second layer, and turn number as inputs to weigh the decisions in the first and second layer, takes the difference, and applies the move to a "dictionary" where the overall maximum difference is chosen as the best move.

Inputs: Current game matrix, the first layer cell array, second layer cell array, and the current turn number.

Outputs: Score key associated with every possible move on the board.

- Determines the color of the piece based on the turn.
- Creates a weight dictionary for the value of each piece.

firstLayerOutput

BLUF: Checks the second layer and determines whether the move would result in the opponent being able to take CHESSTER's king.

Inputs: Current game matrix, the first layer cell array, second layer cell array, and the current turn number.

Outputs: Weight of every move in the first layer cell array.

- Creates a cell array with all of the information about possible moves and their attributed score.
- Determines the location of the friendly king on the board.
- Creates an empty matrix the size of the second layer.
- Analyzes the moves available to CHESSTER using the listMoves function.
- Uses the second layer to determine which of the moves in the move list leave the opportunity for the user to take CHESSTER's king.
- Determines if the move made by CHESSTER would be legal or would result in the loss of its king.
- Outputs a cell array with all relevant information pertaining to moves that could be made including piece identifying information, locations, scores related to moves, and the enemy's best response.

listMoves

BLUF: Lists all moves available to CHESSTER from the first layer

Inputs: Current game matrix and the first layer cell array.

Outputs: List of all moves available from the first layer.

- Creates an empty string array

- Populates the array with information from the first layer, using the location from the first layer as it relates to the game matrix and calling the mapToChessPlot function to convert the game matrix coordinates to locations on the chess board.

  mapToChessPlot

  BLUF: Converts game matrix coordinates to chess locations.

  Inputs: Game matrix coordinates

  Outputs: Corresponding chess board locations

secondLayerOutput

BLUF: Translates the output of the second layer into a 7-cell array format.

Inputs: Current game matrix and the second layer cell array.

Outputs: 7-cell array identifying each piece and its possible moves in response to the first layer

- For every possible move found in the second layer, it is assigned as part of the output array that identifies it by piece type, color, locations, score, and corresponding first layer move.