Air Hockey Robot Visual Calibration and Detection Program

User Manual and Operation Data

Designed by: Roscoe Ambrose, Mason Cannon, Haston LaGrone, and Daniel Pullicar

Table of Contents

[**Introduction 1**](#_9kxzplacp0tb)

[**Library Setup 1**](#_k9nk16cb8nds)

[**Understanding Critical Programs 1**](#_dzdxh449yvp1)

[**Calibration 2**](#_bw8a6h935gpy)

[**Running Calibration 2**](#_ihrttzy2c720)

[**Loading a Calibration File 3**](#_53ff1286se6s)

[**Deleting or Overwriting a Calibration File 3**](#_9mxihgji0tq4)

[**Circle Detection 3**](#_roosm5myjnz0)

[**Graphical User Interface 4**](#_e3j8outwhpgb)

[**Detection Parameters 4**](#_j9zo4avtkzzh)

[**Calibration Settings 5**](#_8r3xfgomskdp)

[**Camera Settings 5**](#_vwygjvxxogcf)

[**Main Interface 6**](#_u5cmx4c2mlvs)

[**Using the Program 6**](#_4f9f9cy2indx)

[**Launching the Program: 6**](#_q26n8f50n5kg)

[**Adjusting Settings for Camera and Circle Detection 6**](#_r3pchhvlj366)

[**Loading Settings for Camera and Circle Detection 7**](#_f8ripdrwswgv)

[**Closing a Webcam View Window 7**](#_zh6ncv8uxj4b)

[**Running Puck Detection 7**](#_c0og46mj8zjh)

[**Multiple Cameras 7**](#_5anldn81ys8t)

[**Modes of Operation 8**](#_64whyi51qvvk)

[**Test Mode 8**](#_amnn136itdlq)

[**Run Mode 8**](#_enb0haxghir3)

[**Known Bugs and Limitations 9**](#_vvbvmr8z2m73)

[**Appendix: 10**](#_ne37zqg3tf4w)

[**FourCC Codes: 10**](#_676uvvarhl9z)

[**ROS Nodes: 11**](#_q0w07ngea26y)

# 

# Introduction

Hello, and thank you for using our program. Its intended purpose is to use a variable number of web cameras to track the real-world position of an air hockey puck within its field of view and provide those coordinates to a robot. Please ensure you read this entire document before beginning the process of setting up this program to run on your computer. It will reduce the amount of time and effort needed to configure and run the program. We hope you enjoy what it offers.

# Library Setup

The visual calibration and detection program is a composite program dependent upon nine files. Six of which are python scripts. One of which is an application icon. Two of which are default variable and calibration files. It is critical that all files are in the same folder so the call functions within can locate their dependent scripts. Pull the following nine files and place them in the same location.

* AirHockeyIcon.ico
* Calibrate.py
* Calibrationfunc.py
* AltCaller.py
* Caller.py
* ConvertToWorldFunc.py
* HoughCirclesClass.py
* CameraArraysDefault.npz
* ProgramVariablesDefault.npz

The names are case sensitive. Ensure the script’s names are ***IDENTICAL*** to those listed above in your directory.

# Understanding Critical Programs

While the overall program utilizes six different python scripts, it can be broken down into three critical components. Calibration, Circle Detection, and the Graphical User Interface (GUI). An overview of what each file does will be discussed, with emphasis on the GUI as it controls much of what the other two programs do.

# Calibration

The calibration program uses the web cameras attached to the computer and a chessboard styled grid to perform intrinsic calibration and then extrinsic calibration (see <https://docs.opencv.org/4.x/dc/dbb/tutorial_py_calibration.html>). Both of these calibration sets are required to output accurate spatial coordinates using a concept called homography (see <https://docs.opencv.org/4.x/d9/dab/tutorial_homography.html>).

## Running Calibration

Calibration is initialized through the GUI using the following steps:

1. Input values for all fields in the calibration settings (see page 6)
2. Input a **Camera Port #** value and a **Camera Name** value.
3. Load any camera settings from the **Load Variables** drop-down menu.
4. Press **Run Calibration**.
5. In the webcam window that appears you will see an overlay appear over the chessboard if it is functioning correctly.
6. Move the calibration chessboard throughout the camera’s field of view, making sure to reach as close to the edges as possible. At each position, press spacebar to capture an image. Repeat at least 9 times making sure the chessboard is completely flat in all images.
7. Move the calibration chessboard so that the starting corner is a position with a known offset from the spatial origin, and orientation is aligned with the axis of your spatial coordinate system (see Figure 1). Calibration works best when this starting corner is at the center of the camera’s view. Press the spacebar to capture this final image.
8. Press escape. The webcam window will update with an undistorted webcam view. This effect is often very subtle. If the webcam view is visibly distorted, you may need to restart calibration from the beginning. A single poor image may cause poor results.
9. Your calibration is saved under “CameraArrays\_\_\_\_\_\_\_.npz”, depending on your **Camera Name** input.

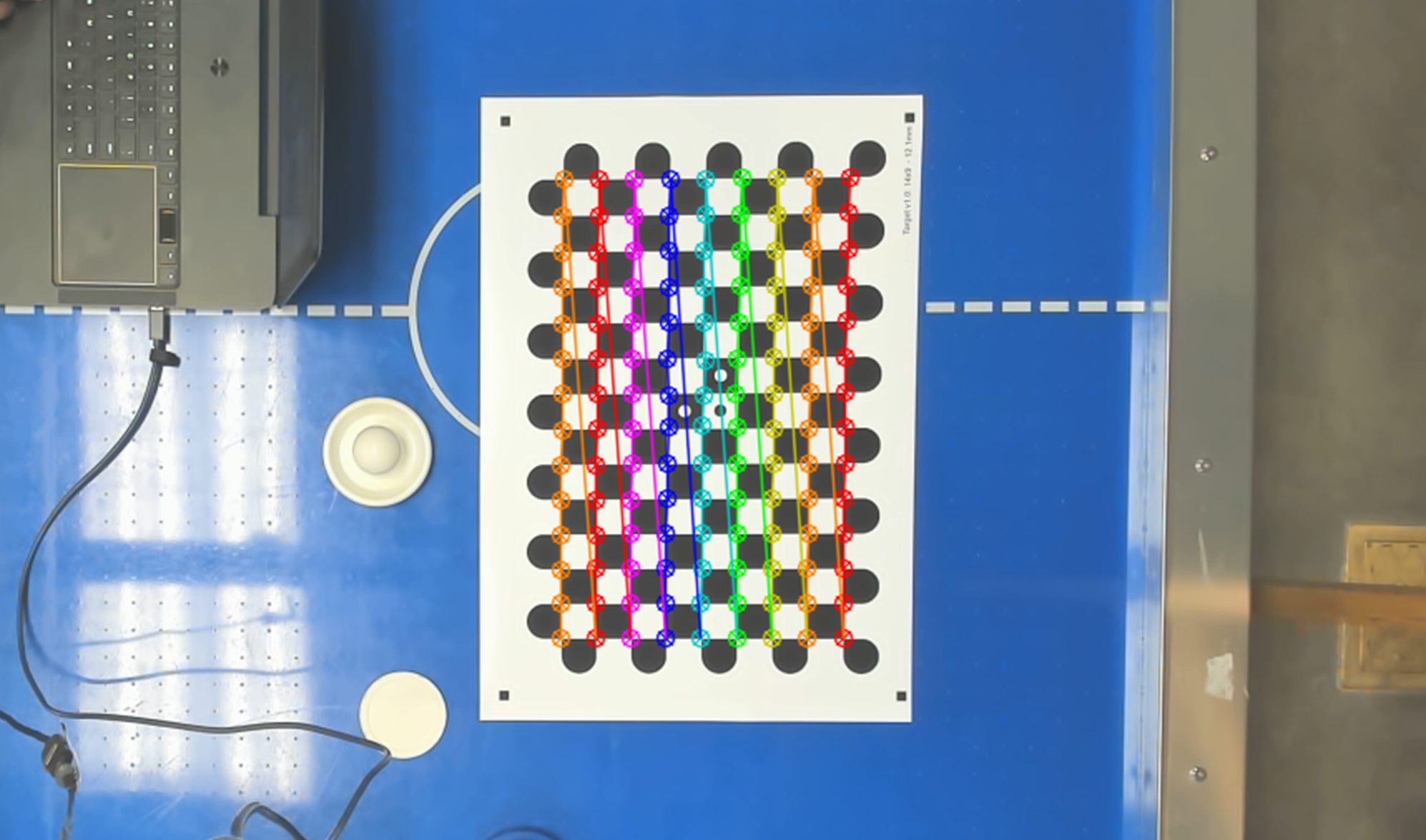


Figure 1: 14 x 9 Calibration Chessboard. The starting corner is in the bottom-right, where the red point does not connect to the orange row.

## Loading a Calibration File

Loading a calibration file is required to properly find the location of the puck. It is accomplished using the following steps:

1. In the **Camera Port** field, input the serial port that the camera is plugged into.
2. In the **Camera Name** field, input the name of the camera’s calibration that you would like to load. For example, if the file saved is called “CameraArraysBrios.npz”, your input should be “Brios”
3. Press **Start Program**

## Deleting or Overwriting a Calibration File

Calibration files are saved into the same folder as your python scripts. They are deleted through your file explorer. To overwrite a calibration file, run a new calibration using the same name as the file you would like to overwrite.

# Circle Detection

The circle detection program uses a list of parameters to “decide” what qualifies as a circle. By tuning these parameters, which is done via the GUI, the operator can identify the puck in the play space and the program will continue to track its position as long as it is in the frame. The program outputs the pixel coordinates of the puck’s centroid to the calibration program, which then converts those coordinates into spatial coordinates. From there that information is published to either the robot or user depending on whether the program is set to run mode or test mode. This is discussed in greater detail in the program initialization section.

# Graphical User Interface

For the majority of users, the GUI will be the only portion of the program that is interacted with. Figure 2 below shows an example of the GUI and is included to aid in its description.

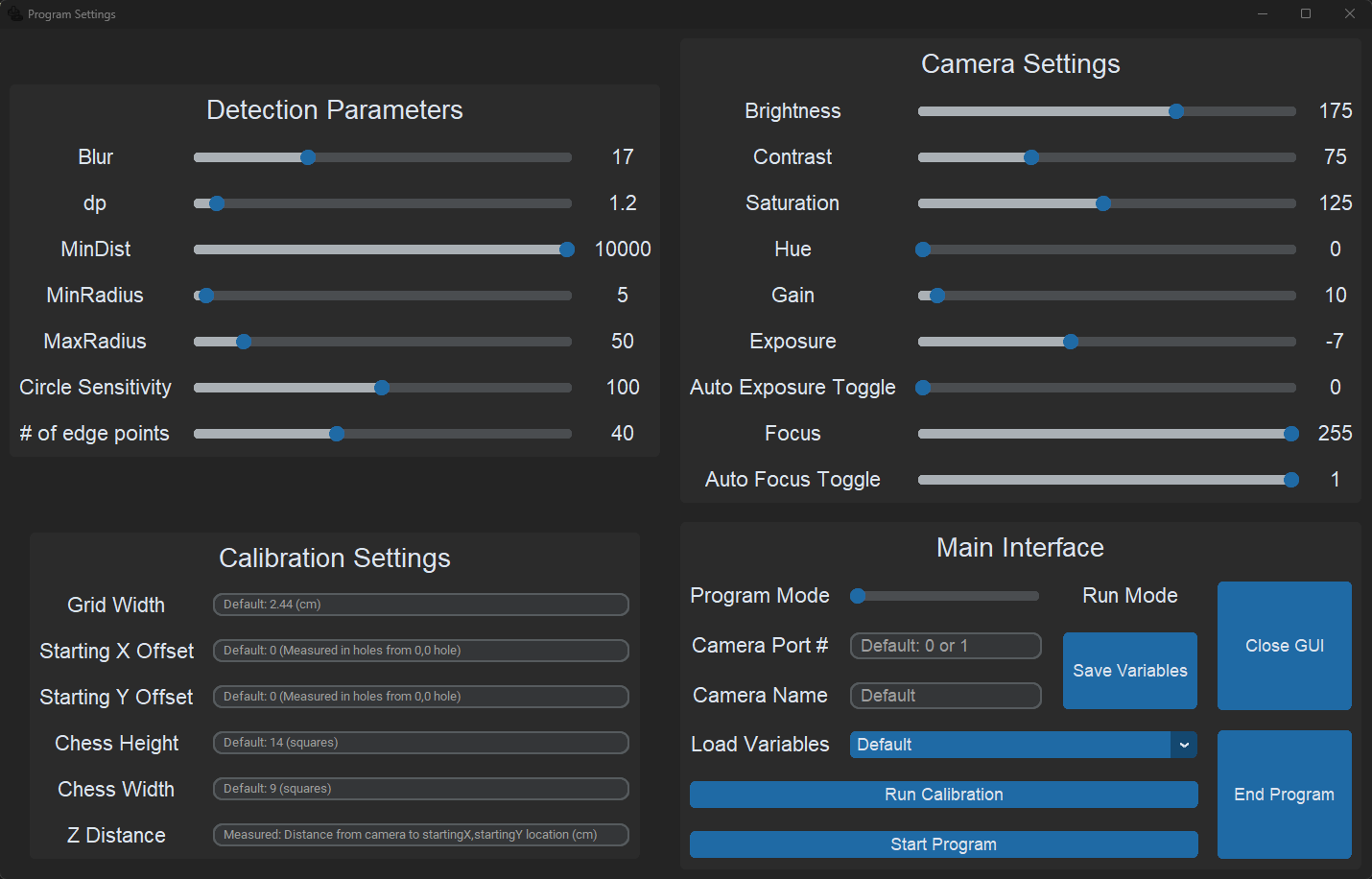


Figure 2: Graphical User Interface

The GUI is broken up into four cells, each of which control a critical function of the program. The top left cell, listed as Detection Parameters, assigns the circle detection program’s parameters. The list below includes a brief description of each.

## Detection Parameters

* **Blur**: Adjusts the amount of Gaussian Blur applied to the frame the program will look for circles in.
* **dp**: This is an inverse ratio of the amount of the frame used for detection. For example, a value of 2 in dp means only half of the frame’s area is used. The region of exclusion begins at the perimeter and moves inwards evenly.
* **MinDist**: This is the minimum distance in pixels between circles. Setting this value greater than the widest dimension of your resolution will ensure only one circle is ever drawn on the frame.
* **MinRadius**: This defines the minimum radius in pixels of a circle to qualify for detection.
* **MaxRadius**: This defines the maximum radius in pixels of a circle to qualify for detection.
* **Circle Sensitivity**: This parameter determines how “circular” an object must be to qualify for detection. The higher the value, the more circular the object must be.
* **# of edge points**: The underlying program the detection program is built on uses drawn points on a perimeter to help determine shapes. This parameter adjusts how many edge points are needed to construct a circle. This ties in closely with the Circle Sensitivity parameter and if desired can be toggled to the point in which noncircular objects can be detected.

This recaps the detection parameters. On the lower left cell are the parameters for the calibration program. Below is a brief description of each field. Note that the measurement of “holes” in the information below is related to the distance between air vents on the table.

## Calibration Settings

* **Grid Width**: Length of an edge of the grid squares in centimeters.
* **Starting X Offset**: Distance in holes of the spatial origin’s X coordinate from some desired real zero coordinate.
* **Starting Y Offset**: Distance in holes of the spatial origin’s Y coordinate from some desired real zero coordinate.
* **Chess Height**: Number of squares on the long axis of the calibration grid.
* **Chess Width**: Number of squares on the short axis of the calibration grid.
* **Z Distance**: Distance in centimeters from the camera aperture to the desired spatial origin. Measured point to point.

This recaps the calibration parameters. Moving to the top right are the parameters for configuring the camera. A brief description will be listed for each parameter below. Note that the way in which cameras interpret these parameters will vary among camera types, even those from the same manufacturer.

## Camera Settings

* **Camera Brightness**: Adjusts the baseline brightness of each pixel in the frame.
* **Camera Contrast**: Adjusts the ratio between color tones within the image.
* **Camera Saturation**: Adjusts the intensity of the colors within the image.
* **Camera Hue**: Adjust the image colors similarity or difference from their primary base color.
* **Camera Gain**: Controls the amplification of the signal from the camera sensor.
* **Camera Exposure**: Controls the time in which the aperture is open to capture a frame. Lower numbers correlate to shorter times and higher frames per second (fps).
* **Camera Auto Exposure**: Allows camera to automatically adjust exposure. Will impact framerate.
* **Camera Focus**: Adjusts the depth of the viewing field.
* **Camera Autofocus**: Allows the camera to automatically adjust the focus. Will impact framerate.

This recaps the Camera Settings. Lastly, the bottom right portion of the GUI is the heart of the program. The main interface’s fields are discussed below.

## Main Interface

* **Camera Port #**: Selects the serial port connected to the camera the user desires to operate.
* **Camera Name**: Provides unique name for calibration and variables file.
* **Save Variables**: Allows users to save variable values adjusted by the GUI for later use.
* **Load Variables**: Selects file to load calibration and parameters from. (File must be in the library).
* **Run Calibration**: Starts calibration program.
* **Start Program**: Begins detection program.
* **Program Mode**: Toggles whether the program is in test mode or run mode. (See ***Modes of Operation*** section for more information.)
* **Close GUI**: Closes Graphical User Interface.
* **End Program**: Closes detection program (If it’s already open).

# Using the Program

With the directory set up the user can begin an instance of the program. It is possible to operate it in two ways. One configuration will be entirely dependent upon the Graphical User Interface (GUI) and is recommended for use when testing the program or when presenting the program to individuals. The second configuration utilizes terminal launches of portions of the program concurrently, allowing for a more dynamic interaction at the cost of ease of use.

## Launching the Program:

The program is initialized by running **Caller.py** but it is important that all files are correctly placed as described in page 1 of this document. Launching the program can be accomplished using the following steps:

1. Launch **Caller.py** either through your command terminal or file explorer. The GUI will open along with a webcam view window.

## Adjusting Settings for Camera and Circle Detection

It is important to get both your camera settings and circle detection settings just right for proper operation. The details of each setting can be found on pages 4-5 of this document. Setting adjustments can be made using the following steps:

1. Launch the program.
2. Do not close the GUI or the webcam view window. If the webcam view is showing you the wrong webcam, disconnect all other webcams to ensure that the intended webcam is using the default port.
3. Enter a name into the **Camera Name** field.
4. Place all objects that may be visible during play in view of the camera.
5. Use the GUI to methodically adjust each setting as you observe its effect on puck detection in real-time in the webcam view window.
6. Once you are ready, click **Save Variables** to save these settings to a file called “ProgramVariables\_\_\_\_\_\_\_.npz” depending on your **Camera Name** input.

## Loading Settings for Camera and Circle Detection

Loading a variables file can save time when initializing your cameras. It is accomplished using the following steps:

1. Launch the program.
2. Click the **Load Variables** drop-down menuto expand it.
3. Select the file you would like to load by the **Camera Name** it was saved under.
4. Once selected, all settings will automatically update.

## Closing a Webcam View Window

Webcam view windows run within a continuous while loop. They can not be exited by pressing the X in the window corner. To close a webcam view window, select it and hold the ‘Q’ key until it closes.

## Running Puck Detection

Once you’ve calibrated the camera and adjusted all your settings, running the program is easy. Follow these steps:

1. Launch the program from the caller.
2. Load the calibration file (see page 3).
3. Load settings (see page 7).
4. Select Run mode/Test Mode from GUI toggle. (See ***Modes of Operation*** section for more information.)
5. Press **Start Program**. A webcam view window will appear if the program is in test mode.
6. Press spacebar while this window is selected to print puck coordinates to the console.
   1. If the program is in run mode, coordinates are published in a constant stream.

## Multiple Cameras

If you desire to utilize multiple cameras, duplicate the **Caller.py** file in your library and assign the duplicate a new name (ex Caller1, Caller2, etc.). Launch the new version of the caller and follow the process detailed above for each camera (important to change the port number of the camera inside the duplicated Caller file). Ensure each variable and calibration file are saved under **UNIQUE** names.

Please see the **Known Bugs and Limitations** section for what not to do when launching.

# Modes of Operation

The program has two modes of operation. Test mode and run mode. Both modes can be run for either configuration method. In both cases the tuning and calibration process will be identical to the methods described in Program Initialization. However, the activities and performance of the program during the detection phase will vary. Initially the user will need to open the Caller.py file and set the testMode variable to either 0 or 1 (0 = run mode, and 1 = test mode). After the program is running, there is a toggle switch, **Program Mode**, on the GUI to set this mode. This change will only take place when relaunching the camera via the **Start Program** button.

## Test Mode

Test mode is a more user centric mode of operation for the program. During the detection phase the input from each camera will be shown on the monitor. The operator can view the circle detection in real time. While tabbed into the viewing window of a camera pressing the spacebar will create a console printout of the real-world coordinates of the puck. The operator can use this data to check the accuracy of the program against a real-world measuring instrument to identify if the calibration was accurate. During multiple camera operation, returned positional data can be compared to further validate if each was calibrated accurately and are also returning the same location. Upon termination of the program, the user will be provided with an accuracy percentage and frame per second readout. This allows for further debugging or tuning if these parameters are below desired values.

It is important to know that displaying the real time tracking can be processor heavy. Depending on the computer's capabilities, a notable reduction in performance can occur. This becomes noticeable with multiple cameras running in parallel. Laptop operation should be relegated to a single camera, with two being the maximum for both unless the laptop is exceptionally capable.

## Run Mode

Run mode is a bare bones version of the program tailored for performance. After configuration the program will no longer return any parameter information or performance data. Coordinates are published in real time to be received by the air hockey robot and are not shown to the user. This eliminates the heavy resource consumption that occurs by providing the user with visuals. This mode is more reliant upon prior testing, so use it when confidence is high in the configuration. Experiments in this mode show improvements in framerate of greater than 15 frames per second (fps) if the camera is not already operating at its cap. If detection parameters are accurate, improved frame rates directly correlate to improved levels of accuracy in the program. It is important to note that in run mode, the program can only be terminated via the **End Program** button on the GUI or by pressing ctrl+c in the terminal that the program is running from.

# Known Bugs and Limitations

The following list will discuss some currently known bugs in the program.

* Frozen interface:
  + If the detection program is launched from the GUI rather than the terminal it will launch a sub thread within the GUI. This results in the parameters in the GUI not being able to be adjusted during operation. Must press and hold ‘Q’ while on the camera window to close it- then the GUI will become responsive again.
  + ***IF*** an attempt is made to adjust these parameters during this event, the program ***WILL*** crash.
* Program ***WILL*** crash if initial tuning window is not closed before launching calibration:
  + If the window used to initially tune the camera is not closed using the escape key before the calibration is launched using the GUI, the program will become indefinitely frozen. Requires task manager to quit.
* Program ***WILL*** crash if a new variable set is selected from **Load Variables** dropdown menu if **Program Mode** switch is toggled to run mode when the program was started in test mode.
* Program ***WILL NOT*** inform you if the camera parameters passed are invalid for the connected camera:
  + Many web cameras use proprietary software to configure their many parameters. Because of this, the way in which they interpret values passed to them can vary significantly. Due to limitations within the OpenCV library itself, the program will not inform you if a parameter is not accepted. The camera will simply use its last configuration for that camera and the program will continue executing. This can only effectively be discerned by viewing the detection window and seeing if the camera responds to the input.
  + Also note, if the camera appears to be completely unresponsive to passed parameters, reference the ***fourcc*** subsection of the **Appendix**. It is possible the problem stems from this value.
* API backend incompatibilities:
  + The default OpenCV API can typically interface with any web camera. However, they cannot utilize them to their maximum potential. Within the detection and calibration program a parameter known as the ***fourcc*** (four-character code) dictates the API backend that is being interfaced with. If the correct one is passed for the operating system/camera interface the user can utilize it to its full potential. However, this code can in some cases be different than the one within the program. MJPG is typical for most Linux and windows devices, but the full library is over 300 codes, and some cameras may need a different value passed. The program ***CANNOT*** identify this for you.
* Hardware limit on performance:
  + As touched upon before, the processing power of the computer will be directly related to the accuracy of the program. If the cameras being utilized cannot run at 60 fps or greater, it is possible for the program to lose track of the puck as its velocity will cause its shape to no longer be circular but become elliptical. This degrades the positional data output by the program and reduces the overall accuracy of the tracking.
* Significant delay during calibration when a grid is not visible.
  + If the calibration grid is not in full view of the camera while the calibration program is running, the program frantically tries to identify a candidate grid. This consumes a significant amount of resources and will slow the camera feed down. If this occurs, it is easiest to identify the center of the camera's field of view from the calibration window and then place the grid in that location. Wait for it to catch up then move the grid once the color overlay is visible. Try to not obscure the grid while you move it across the play space to take calibration pictures. It runs significantly faster that way.

# Appendix:

## FourCC Codes:

There is a possibility that certain cameras and computers will not be able to interface using the ***fourcc*** code within the parameters of the program. This will be evident if a camera’s resolution and frame rate do not match those passed to the camera (if those parameters are known to be valid operating specifications for the camera). If this occurs open the **HoughCirclesClass.py** and navigate to line 47. You will see this line: ***camera.fourcc = cv.VideoWriter\_fourcc('M','J','P','G').*** For Linux devices, there are a handful of alternate ***fourcc*** codes that may resolve the issue. They are listed below.

* V4L2
* H264
* MP4V

Change the MJPG letters to match those of the ***fourcc*** code you wish to test, then proceed through the **launch program** steps.

If none of these variations work, a full list of ***fourcc*** codes can be found here: (<https://softron.zendesk.com/hc/en-us/articles/207695697-List-of-FourCC-codes-for-video-codecs>)

## ROS Nodes:

For publishing coordinate data from the detection program to a ROS Node, first open **ConvertToWorldFunc.py.** Navigate to line 45 where you will find the following code: ***return [objpos,imgMtx].*** The matrix objpos contains the real-world coordinates of the puck. By extracting objpos[0] and objpos[1] you will have the respective x and y coordinates of the puck to publish to ROS.