2016

MECH 423 Final Report

AIR HOCKEY PLAYER

JOHNNY LI – 37553112 EMAIL: johnnyli8@hotmail.com

BRIAN WANG – 35401108 EMAIL: brian.ck.wang@gmail.com

Table of Contents

Abstract 1

1. Objectives 2

2. Rationale 2

3. Summary of Functional Requirements 2

3.1 Hardware mechanism construction 3

3.2 Vision system for object position detection 3

3.3 Computer central control interface 3

3.4 Motor control hardware 3

3.5 Stepper motor control software 3

4. Functional Requirement #1 – Mechanical Hardware Construction 4

4.1 Approach and Design 4

4.1.1 Motor Support 5

4.1.2 Freewheel Support 6

4.2.3 Mallet Holder 6

4.1.4 Camera Holder and Lighting Assembly 7

4.2 Inputs and Outputs 8

4.3 Parameters 8

4.4 Testing and Results 8

5. Functional Requirement #2 – Vision System 9

5.1 Approach and Design 9

5.1.1 Camera 9

5.1.2 Vision Library 9

5.1.3 Vision Processing Parameters and Interface 9

5.2 Inputs and Outputs 9

5.3 Parameters 10

5.4 Testing and Results 10

5.4.1 Camera Frame Rate 11

5.4.2 Filter Performance 11

6. Functional Requirement #3 – Communication/Vision Interface 12

6.1 Approach and Design 12

6.1.1 Communication Interface 12
6.1.2 Color Detection Parameters 13
6.1.3 Puck and Mallet Related Parameters 13
6.1.4 Camera Setting, Camera Start, and Data Recording 14
6.1.5 Position Detection Filtering 14
6.1.6 Puck Path Detection and Time Elapsed Display Boxes 15
6.1.7 Image Frames and Filtering 16
6.1.8 Serial Communication Settings 17
6.1.9 Path Prediction Tester 18
6.1.10 Color Filter Examples 18
6.2 Inputs and Outputs 19
6.2.1 Inputs 19
6.2.2 Outputs 19
6.3 Parameters 19
6.4 Testing and Results 19
1. Changeable Bounce Boundary Numeric Control 19
2. Constant Target Position 19
3. Packet Transmit Mode 19
7. Functional Requirement #4 - Motor control hardware 20
7.1 Approach and Design 20
7.2 Inputs and Outputs 22
7.3 Parameters 22
7.4 Testing and Results 22
Testing Functionality by Manually Stepping 22
Testing Functionality with Varying Frequency Signal from Arduino 22
8. Functional Requirement #5 - Stepper Motor Control Software 23
8.1 Approach and Design 23
8.1.1 Controller Feedback 23
8.1.2 Code 24
8.2 Inputs and Outputs 25
8.3 Parameters 25

8.4 Testing and Results 25

Setting Kp and KI 25

9. System Evaluation 26

10. Reflections.. 26

Most Useful Lessons from MECH 423.. 26

Things to learn going forward.. 26

11. Appendix A – Motor Control Software 27

Arduino Code 27

12. Appendix B – C# Software.. 33

Abstract

The Air Hockey Player is an autonomous robot that plays air hockey on a mini table top air hockey table. It reacts to the incoming puck sent by a human player and move its mallet accordingly to defend its goal.

The robot is actuated by a stepper motor controlled in vision based closed loop position control. The vision system utilizes the PS3EYE camera for its inexpensive and capable hardware to meet the requirements and the algorithm is developed in Microsoft Visual Studio in C# with EMGU V3.1.0 vision library which is an openCV wrapper library for C# and many other languages.

The air hockey table and the camera were purchased from Amazon.ca. All the custom designed components are 3D printed with a Prusa i3 RepRap printer while other commercial parts are purchased from McMaster Carr and various other sources. An Arduino and a stepper driving breakout board are used to accept information from the vision system and drive the stepper motor to the desired location.

The project development took roughly two and a half weeks and it is functional while having rooms for improvement.

1. Objectives

The goal of the project was to design a vision system and motor control system to mount on an existing miniature air hockey table to defend the goal. The design was limited to 1-axis and therefore could only defend against the player's shots at the goal. For the final product, we were able to build the mechanical system, implement a computer vision algorithm and control a stepper motor to defend the goal.

2. Rationale

The aspect of the project that is interesting is developing machine vision to control and actuate an air hockey mallet to defend the goal. The challenge is processing the input quickly enough to produce a response able to defend the hockey puck. It is worthwhile to do this project because vision based robot control is a growing and promising field and very important in the robotics industry. The project is different from other air hockey table robots in that we are limiting the system to 1-axis control and using a smaller sized air hockey table.

3. Summary of Functional Requirements

Table 1 – Functional Requirements and Assigned Tasks

Functional Requirements	**% Effort**	**Responsible Person**
Hardware mechanism construction (physical structure to support components)	10	Brian & Johnny
Vision system for object position detection	25	Brian
Computer central control interface	20	Brian
Motor control hardware	20	Johnny
Stepper motor control software	25	Johnny

3.1 Hardware mechanism construction

Design, produce, and purchase parts to construct an air hockey table with stepper motor driving mechanism and a camera holder.

3.2 Vision system for object position detection

Develop an algorithm that processes captured incoming image frames for object position detection and outputs necessary information for motor control.

3.3 Computer central control interface

Develop a C# interface for user adjustable variables and serial transmission control.

3.4 Motor control hardware

Build a stepper motor driving hardware that runs the motor control firmware.

3.5 Stepper motor control software

Develop a stepper motor driving algorithm that takes packet information from the vision system and outputs signal pulses for stepper motor control.

4. Functional Requirement #1 – Mechanical Hardware Construction

Hardware mechanism construction (physical structure to support components).

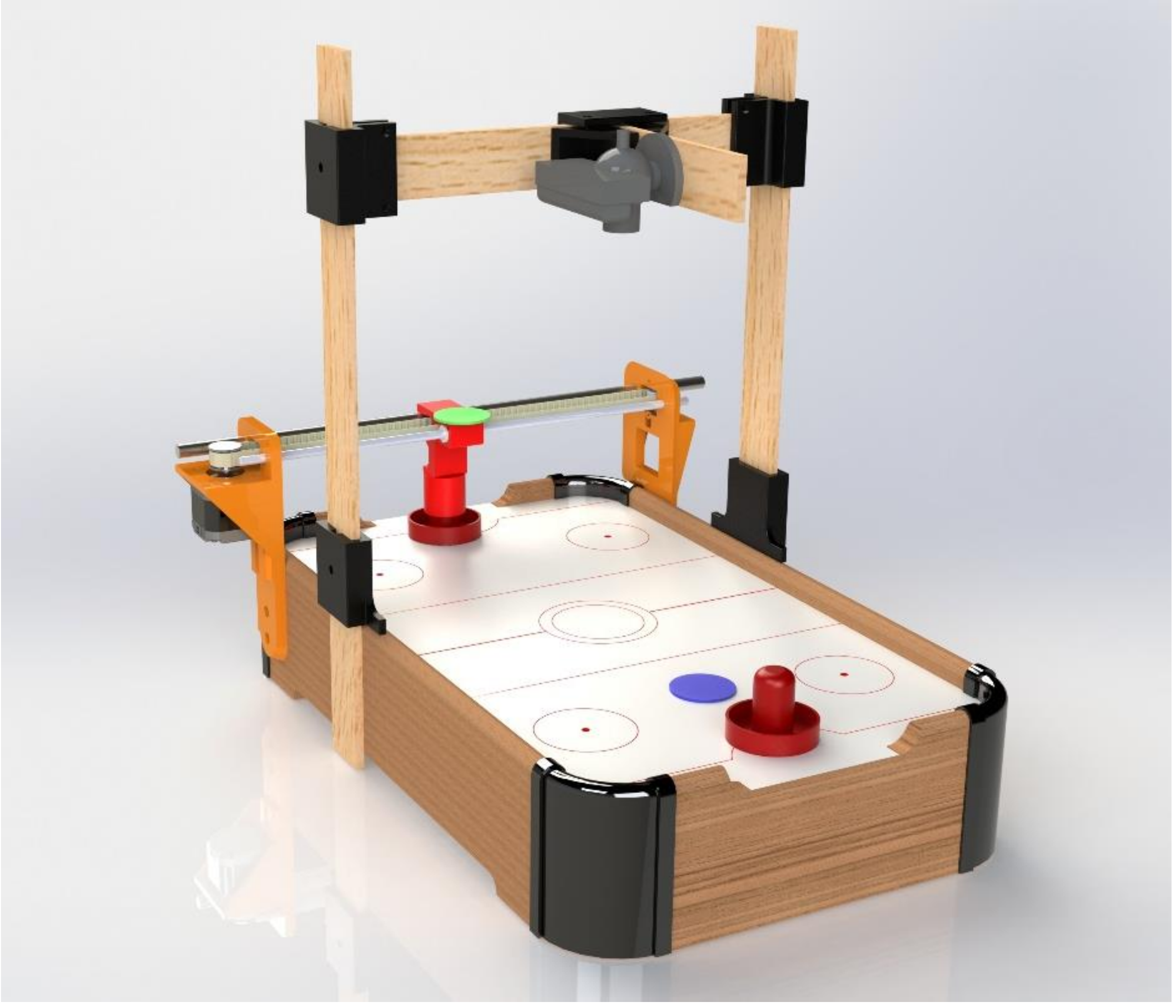


Figure 1 – Final Design Rendering of Assembly

4.1 Approach and Design

The objective is to construct a physical device that is capable of performing the following functions.

- Interface and connect with air hockey table
- Supports a camera for vision input
- Supports a 1-axis motor driven mechanism for manipulating the air hockey mallet

For simplicity, an existing air hockey table is used for the base of the device and is being modified for the additional required functions. Most of the components used to mount all the hardware were printed using a 3D printer with ABS plastic.

4.1.1 Motor Support

One side of the supports is fitted with mounting holes to attach a NEMA 17 stepper motor. The holes for supporting the rod are positioned to allow for the timing belt and timing belt pulley clearance to pass between the rods. The support is 3D printed using ABS plastic.

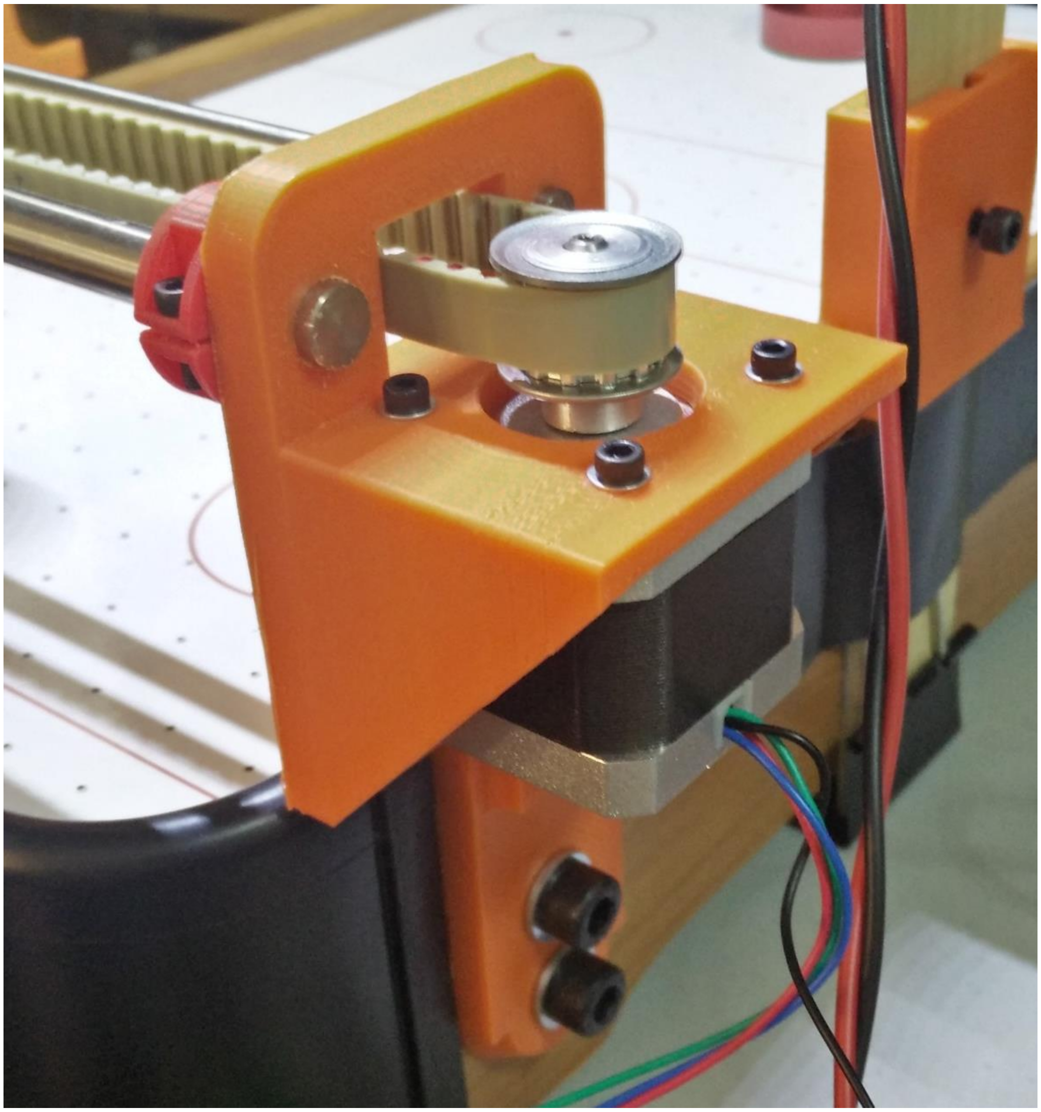


Figure 2 - Motor support 3D printed

4.1.2 Freewheel Support

The freewheel consists of a M5 screw with a ball bearing in between spacers and a nut. It is used to guide the timing belt pulley as well as provide tension. In our design, we created a slot to increase the tension of the timing belt. However, our initial placement of the freewheel hole provided sufficient tension for the project.

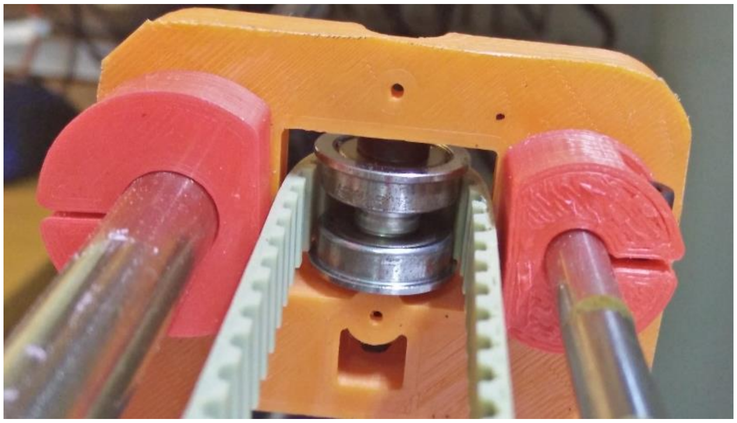


Figure 3 - Freewheel support 3D printed

4.2.3 Mallet Holder

The mallet holder is constrained to the front rod with a PTFE bushing on both sides. For the rear rod, a C-shaped opening was used to prevent rotation and allow for some tolerance to any misalignment of the rods. The mallet holder has a matching timing belt profile that clips onto the timing belt. The green circle is used for vision tracking the position of the mallet.

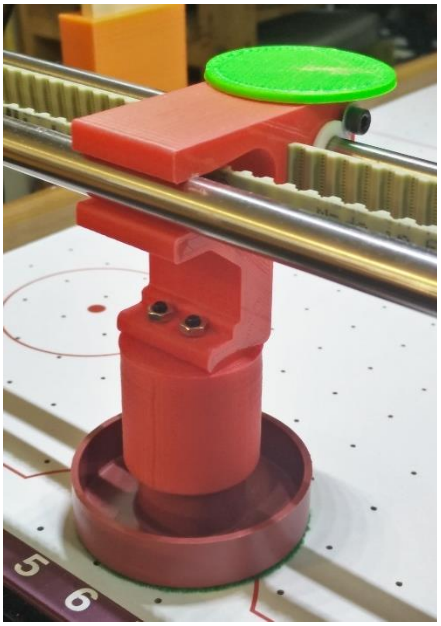


Figure 4 - Mallet holder assembly

4.1.4 Camera Holder and Lighting Assembly

The camera required designing clamps that could be adjusted to move the camera in all three axes (X, Y, Z). This was accomplished using two wooden supports to guide the clamps vertically, one wooden support to guide the clamp laterally (width of table), and one wooden support to adjust the position of the camera longitudinally (length of table). The clamps were secured in place with M5 screws. A strip of LED light is used to illuminate the table in order to eliminate shadows for better vision detection.

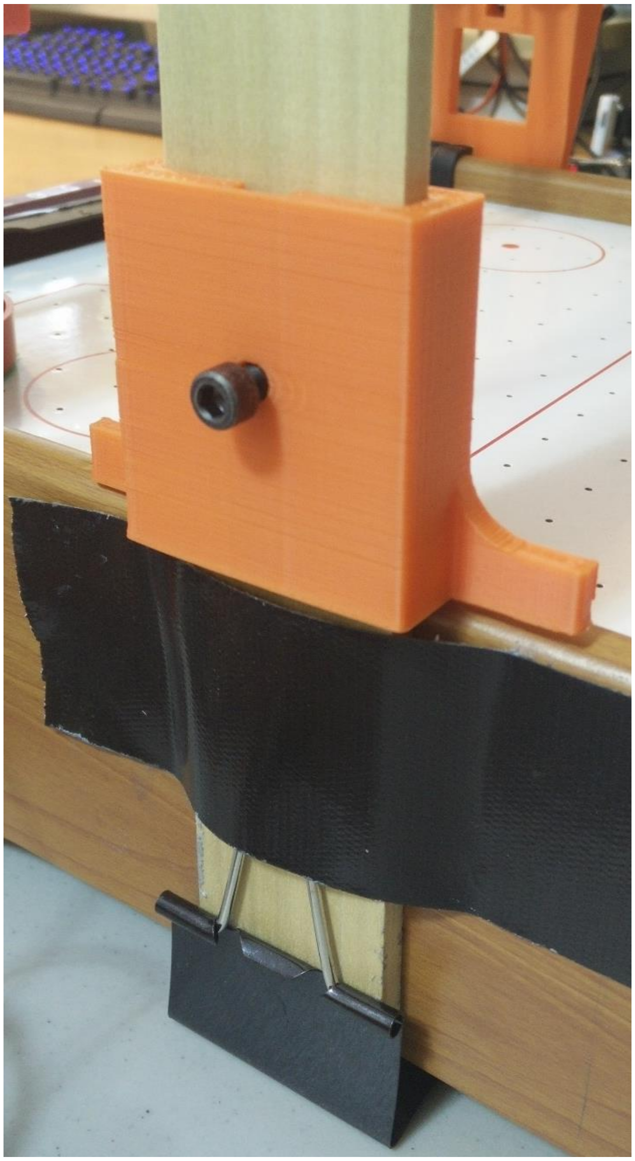


Figure 5 - Clamp design

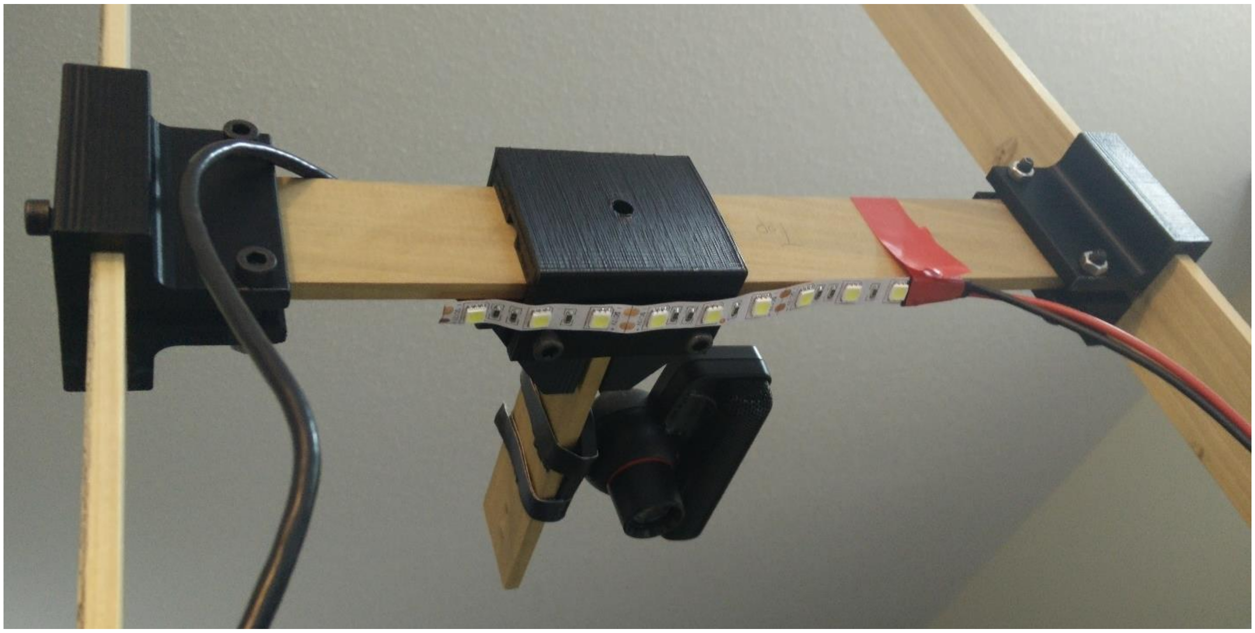


Figure 6 - Camera and LED Strip

4.2 Inputs and Outputs

<u>Input</u>

1. Motor control signal and power sent by the motor control circuitry.

<u>Output</u>

2. Picture frames.

4.3 Parameters

Motor operating speed	The control command update rate
Mallet position	Design relative position on the air hockey table
Motor sizing	Choose appropriate motor size for desired driving performance
Camera distance to surface	The hardware needs to be adjustable to accommodate varying the camera distance to playing surface

4.4 Testing and Results

The main design effort was for the 3D printed parts as they were constrained by the metal and wooden materials we scavenged or purchased with our provided funds of $30. The hardware was verified by physically mounting it to the air hockey table. The design was made with clearance for moving parts in mind.

5. Functional Requirement #2 – Vision System

Vision system for object position detection.

5.1 Approach and Design

The objective is to develop an algorithm that takes the frames output by the camera and output the 2D position of the tracked puck, its predicted path of travel, and mallet destination position. The processing will be done by the host desktop PC with a script to determine necessary information.

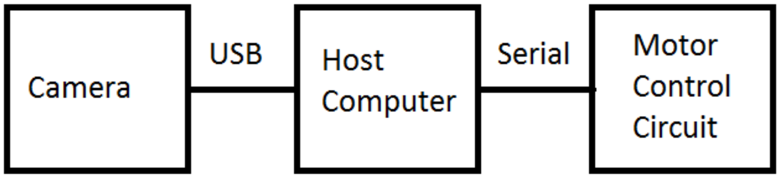


Figure 7 -Vision Block Diagram

5.1.1 Camera

A PS3EYE was used for obtaining image frames in reasonable frame rate. The camera was chosen sue to its high frame rate output capability as well as its image resolution. After several trials, the frame rate was set to be 60Hz and resolution of 320x240 for the best shape detection stability.

5.1.2 Vision Library

The algorithm was developed in C# form application using EMGU 3.1.0 which is a C# wrapper that provides openCV (open Computer Vision) function compatibility in C# and many other languages. The library provides most of the functions that openCV provides and is adequate for the scope of this project.

5.1.3 Vision Processing Parameters and Interface

Vision processing parameter controls are integrated with communication interface and are further described in 6.1 Approach and Design under 6. Functional Requirement #3.

5.2 Inputs and Outputs

Input

1. Picture frames from camera at a frame rate of 60Hz and 320x240 in resolution.

Output

1. Puck filtered current position.
2. Path of travel and bounce direction.
3. Estimated intersection point with axis of motion for mallet.

5.3 Parameters

Table 1 - Vision System Parameters

Frame resolution	The resolution of each frame will affect the processing time required as it takes time to send more information. Counterintuitively, when the resolution was in 640x480, the shape detection result was jittery and once resolution was set to 320x240, the detection was stable. This rate is also the data output rate to the motor control software because it is processed every time a new frame gets quarried.
Frame rate	This will affect how close to 'real time' the frames are when they are analyzed. The high frame rate also eliminates motion blur which makes shape detection more difficult.
Object filter method	Visual inspection for result. HSV color filter, Gaussian smoothing and eroding methods were used to filter the input image. Hough circle transformation was used to detect puck and mallet position after filtering. Alpha smoothing is then used to filter the jittering position detection for reliable post-processing operations like path prediction.
HSV	H, S, and V values are set for a range for the color detection.
Alpha	Alpha is a percentage value for the exponential smoothening low pass filter. It is ranged from 0 to 1 and the higher the number is the more weight new values affects the output.
Range	Range values are the minimum change of values for it to be considered changed. This is to eliminate the small change in position values that affects vector calculation. Small change in position can result in very different resulted direction.
Image Filter Gauss Settings	Several parameters for the Gaussian smoothing for the image. The values are chosen at default because they already work well.
Hough Circle Settings - Canny Threshold	The threshold affects how the circle is detected in the Hough circle transformation.
Hough Circle Settings – Accumulator Threshold	This threshold affects the chance of false positive for a circle to be detected.
Hough Circle Settings – Min/Max Radius	The min/max radius values are to limit the size of the circles detected to prevent false positive from color disturbance.

5.4 Testing and Results

The system is capable of detecting the position of puck and mallet holder as well as calculate the predicted path of puck movement and mallet destination point for collision at adequate speed of the camera frame rate.

The position detected is precise to ±1 pixel and, by using some filtering technique, it is enough for the closed loop position control for the motor.

The following testing procedures were done to ensure the correct operation of the system.

5.4.1 Camera Frame Rate

The camera frame rate is displayed in the image box property panel in C#. The intended frame is 60Hz but the real output rate was roughly 54Hz. The dropped frame rate did not pose any significant system performance degradation because the magnitude of the puck velocity was not important and it is the direction vector of the velocity that affects the follow up operations.

5.4.2 Filter Performance

A variety of filtering technique was tried and compared in effectiveness and performance. Performance matrix including the clearness of the object edge as well as the stability of the shapes in black and white images for object detection.

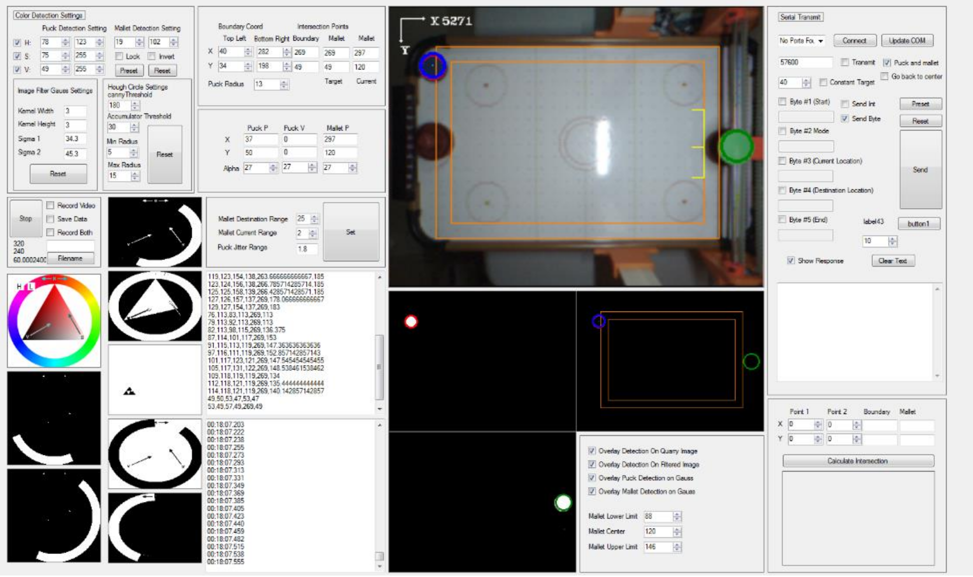
6. Functional Requirement #3 – Communication/Vision Interface

Computer central control interface.

6.1 Approach and Design

The objective is to develop a control interface that displays system state as well as tunable parameters of the vision system and motor control system.

6.1.1 Communication Interface



Color Detection Settings

	Puck Detection Setting		Mallet Detection Setting	
☑ H:	78	123	19	102
☑ S:	75	255	□ Lock	□ Invert
☑ V:	49	255	Preset	Reset

Image Filter Gauss Settings

Kernel Width 3
Kernel Height 3
Sigma 1 34.3
Sigma 2 45.3

Hough Circle Settings
cannyThreshold 180
Accumulator Threshold 30
Min Radius 5
Max Radius 15

	Boundary Coord Top Left	Bottom Right	Intersection Points Boundary	Mallet	Mallet
X	40	282	269	269	297
Y	34	198	49	49	120
				Target	Current

Puck Radius 13

	Puck P	Puck V	Mallet P
X	37	0	297
Y	50	0	120
Alpha	27	27	27

□ Record Video
□ Save Data
□ Record Both
320
240
60.0002400

Mallet Destination Range 25
Mallet Current Range 2
Puck Jitter Range 1.8

```
119,123,154,138,263.666666666667,185
123,124,156,138,266.785714285714,185
125,125,158,139,266.428571428571,185
127,126,157,137,269,178.066666666667
129,127,154,137,269,183
76,113,83,113,269,113
79,113,92,113,269,113
82,113,98,115,269,136.375
87,114,101,117,269,153
91,115,113,119,269,147.363636363636
97,116,111,119,269,152.857142857143
101,117,123,121,269,147.545454545455
105,117,131,122,269,148.538461538462
109,118,119,119,269,134
112,118,121,119,269,135.444444444444
114,118,121,119,269,140.142857142857
49,50,53,47,53,47
53,49,57,49,269,49
```

```
00:18:07.203
00:18:07.222
00:18:07.238
00:18:07.255
00:18:07.273
00:18:07.293
00:18:07.313
00:18:07.331
00:18:07.349
00:18:07.369
00:18:07.385
00:18:07.405
00:18:07.423
00:18:07.440
00:18:07.459
00:18:07.482
00:18:07.515
00:18:07.538
00:18:07.555
```

Serial Tranamit

No Ports Fou
57600 □ Tranamit ☑ Puck and mallet
40 □ Constant Target □ Go back to center
□ Byte #1 (Start) □ Send Int ☑ Send Byte
□ Byte #2 Mode
□ Byte #3 (Current Location)
□ Byte #4 (Destination Location)
□ Byte #5 (End) label43 10
☑ Show Response

	Point 1	Point 2	Boundary	Mallet
X	0	0		
Y	0	0		

☑ Overlay Detection On Quary Image
☑ Overlay Detection On Filtered Image
☑ Overlay Puck Detection on Gauss
☑ Overlay Mallet Detection on Gauss

Mallet Lower Limit 88
Mallet Center 120
Mallet Upper Limit 146

Figure 8 -Final Software Interface

The interface separates function controls using panels. There are 9 main sections indicated in Figure 9.

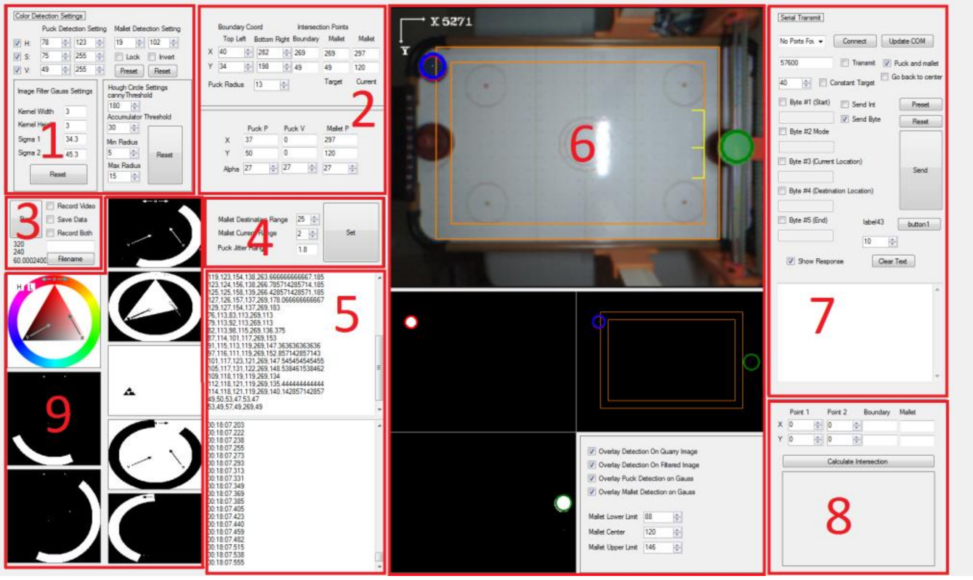
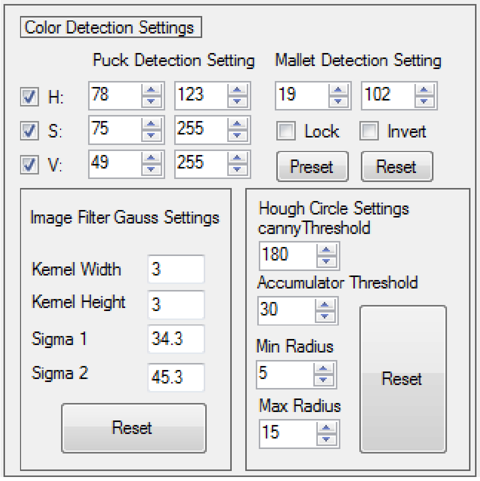


Figure 9 - Communication Interface Section Numbered

6.1.2 Color Detection Parameters



Color Detection Settings

	Puck Detection Setting		Mallet Detection Setting	
☑ H:	78	123	19	102
☑ S:	75	255	☐ Lock	☐ Invert
☑ V:	49	255	Preset	Reset

Image Filter Gauss Settings

Kernel Width	3
Kernel Height	3
Sigma 1	34.3
Sigma 2	45.3

Reset

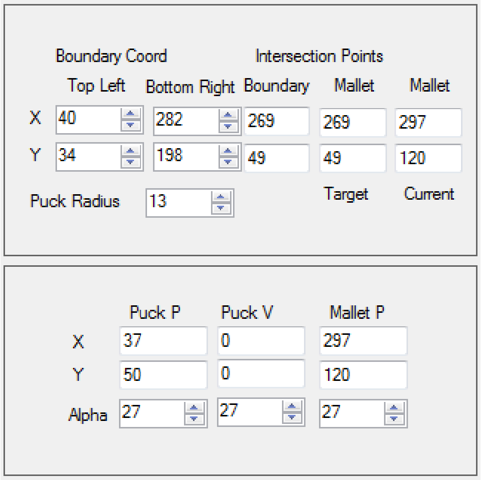
Hough Circle Settings
cannyThreshold 180
Accumulator Threshold 30
Min Radius 5
Max Radius 15

Reset

Figure 10 - Communication Interface Section 1 - Color Filter Parameters

In panel 1, there are many numeric controls for the color detection. The filter uses HSV for color, brightness, and saturation level for filtering out the colors desired under different lighting conditions.

6.1.3 Puck and Mallet Related Parameters



	Boundary Coord		Intersection Points		
	Top Left	Bottom Right	Boundary	Mallet	Mallet
X	40	282	269	269	297
Y	34	198	49	49	120
				Target	Current

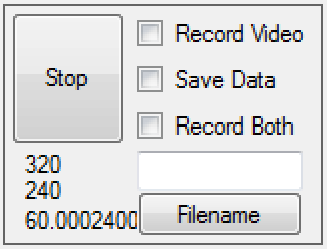
Puck Radius 13

	Puck P	Puck V	Mallet P
X	37	0	297
Y	50	0	120
Alpha	27	27	27

Figure 11 - Communication Interface Section 2 - User Set Puck and Mallet Related Parameters

In section 2, the boundary coordinates numeric boxes display the user set boundary points. The user can later on change the values on the fly for suitable boundary in case the condition changes, i.e., if camera shifted. The current position and predicted mallet position are displayed in text boxes for reference. On the bottom panel, the puck position, puck direction vector, and mallet current position are displayed for user reference. The alpha values are for exponential smoothening for the puck and mallet position values for better prediction.

6.1.4 Camera Setting, Camera Start, and Data Recording



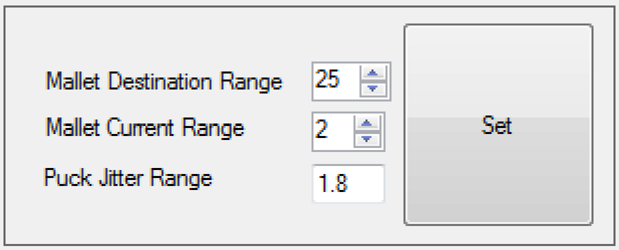
□ Record Video
□ Save Data
□ Record Both

320
240
60.0002400

Figure 12 - - Communication Interface Section 3 - Camera Settings, Start, Recording Settings

The camera section contains a start/stop button for camera operation. The Record Video check boxes, once enabled, records the image frames with overlays into a video in avi format. The Save Data check box, once enabled, starts the recording of important data at the instant for debugging purposes. The numbers under the button show the width and height of image frame resolution and also the frame rate.

6.1.5 Position Detection Filtering

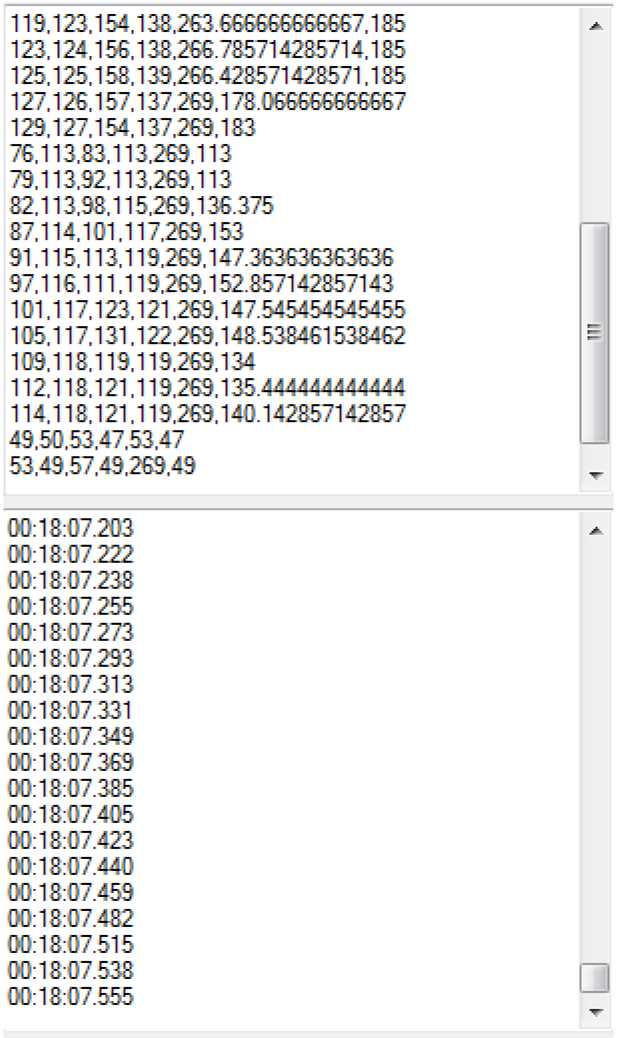


Mallet Destination Range 25
Mallet Current Range 2
Puck Jitter Range 1.8

Figure 13 - Communication Interface Section 4 - Position Detection Filtering

This section set some additional filter settings for puck and mallet. This is the range boundary to eliminate small change in values that causes high frequency jitters in order to smooth out motor control.

6.1.6 Puck Path Detection and Time Elapsed Display Boxes



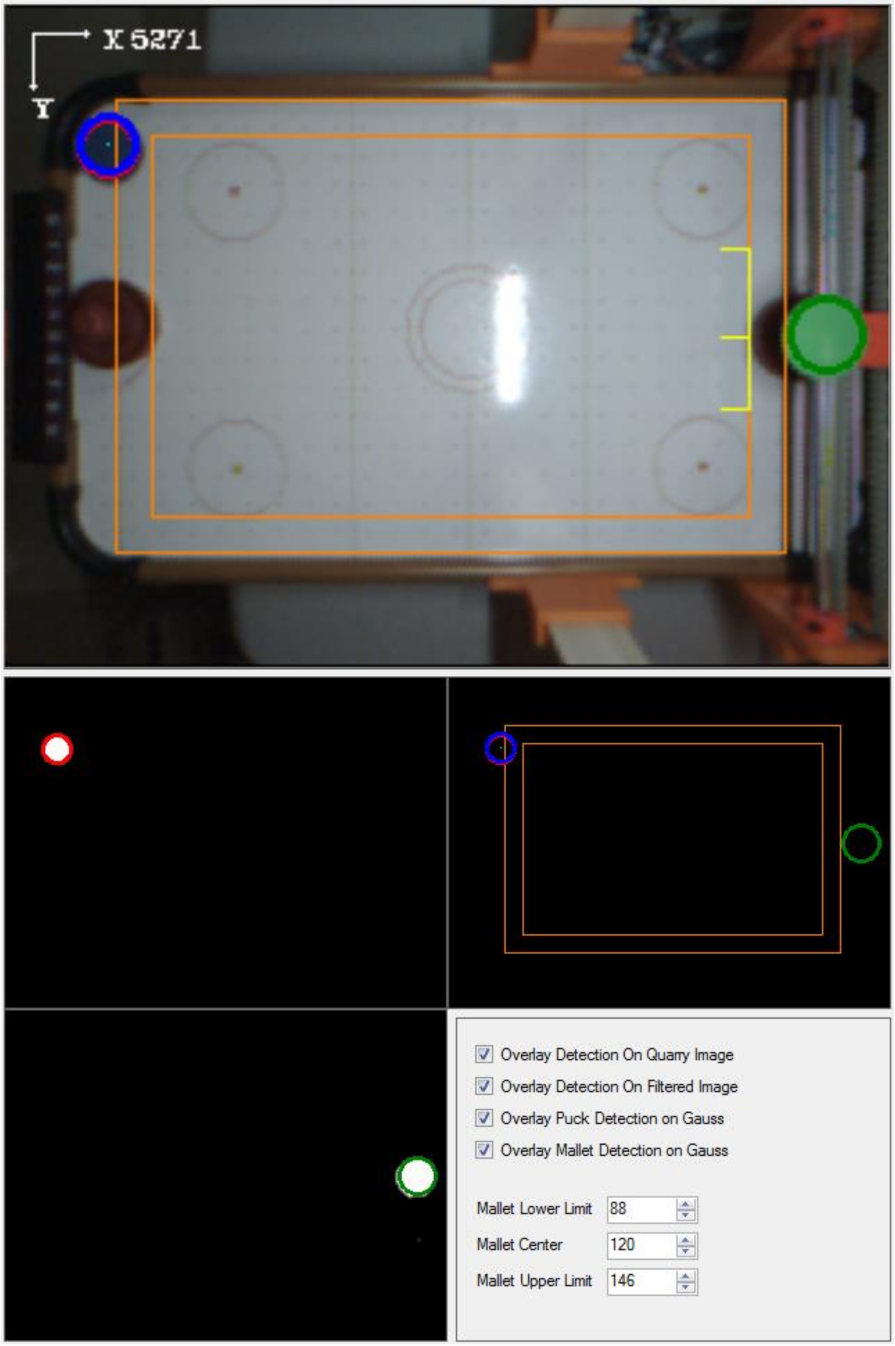
```
119,123,154,138,263.666666666667,185
123,124,156,138,266.785714285714,185
125,125,158,139,266.428571428571,185
127,126,157,137,269,178.066666666667
129,127,154,137,269,183
76,113,83,113,269,113
79,113,92,113,269,113
82,113,98,115,269,136.375
87,114,101,117,269,153
91,115,113,119,269,147.363636363636
97,116,111,119,269,152.857142857143
101,117,123,121,269,147.545454545455
105,117,131,122,269,148.538461538462
109,118,119,119,269,134
112,118,121,119,269,135.444444444444
114,118,121,119,269,140.142857142857
49,50,53,47,53,47
53,49,57,49,269,49
```

```
00:18:07.203
00:18:07.222
00:18:07.238
00:18:07.255
00:18:07.273
00:18:07.293
00:18:07.313
00:18:07.331
00:18:07.349
00:18:07.369
00:18:07.385
00:18:07.405
00:18:07.423
00:18:07.440
00:18:07.459
00:18:07.482
00:18:07.515
00:18:07.538
00:18:07.555
```

Figure 14 - Communication Interface Section 5 - Puck Path Detection and Time Elapsed

These two text boxes display some additional information. The top box shows the path prediction using last position and current position to determine the velocity vector. The magnitude is meaningless as the frame rate sometimes changes so time interval is only an estimate. The bottom box displays time elapsed. It is used to determine whether there is a significant lag from any new algorithm change.

6.1.7 Image Frames and Filtering



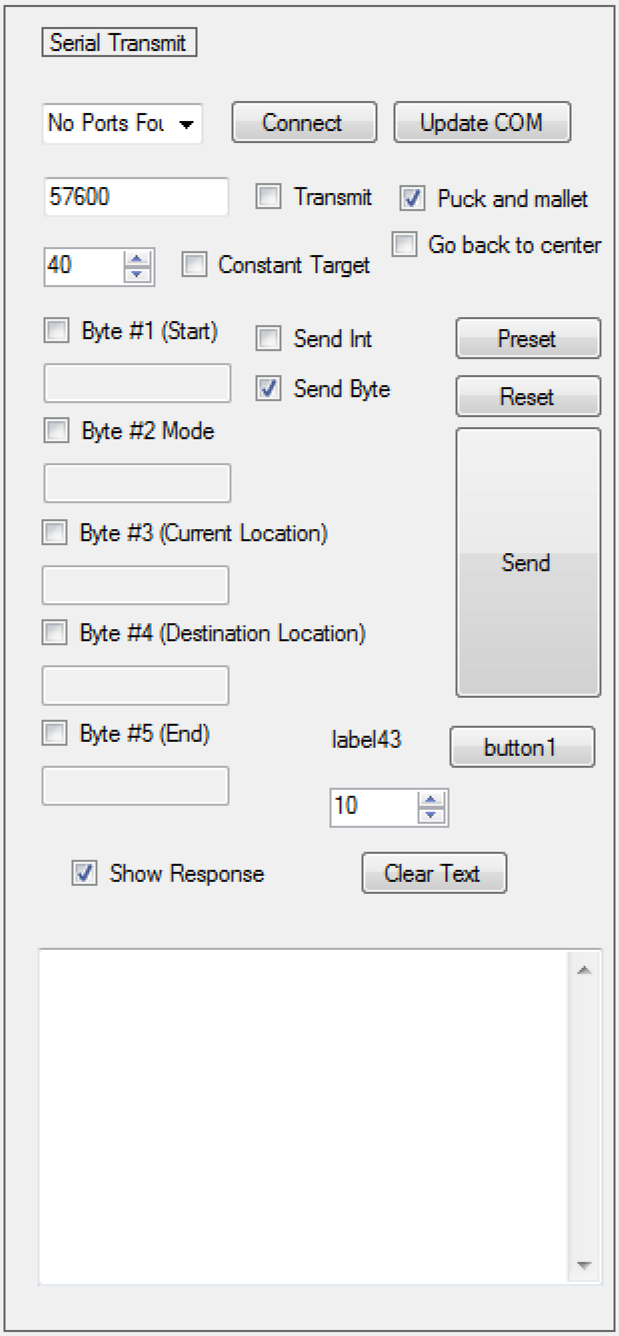
☑ Overlay Detection On Quarry Image
☑ Overlay Detection On Filtered Image
☑ Overlay Puck Detection on Gauss
☑ Overlay Mallet Detection on Gauss

Mallet Lower Limit 88
Mallet Center 120
Mallet Upper Limit 146

Figure 15 - Communication Interface Section 6 - Image Frame and Filtering

The top image box displays the current captured image frame from the camera. Bottom three are filtered images with shape detection overlay and boundary overlay. The numeric controls are for changing the circle detection limits. The boundary is set by click on the top image box twice where the two diagonal points locate. The software will automatically calculate the four boundary lines and use a default value of puck radius for the inner boundary lines. The inner boundary is used to calculate the puck bounce resultant direction.

6.1.8 Serial Communication Settings



Serial Transmit

No Ports Fou ▾

57600

□ Transmit

☑ Puck and mallet

40

□ Constant Target

□ Go back to center

□ Byte #1 (Start)

□ Send Int

☑ Send Byte

□ Byte #2 Mode

□ Byte #3 (Current Location)

□ Byte #4 (Destination Location)

□ Byte #5 (End)

label43

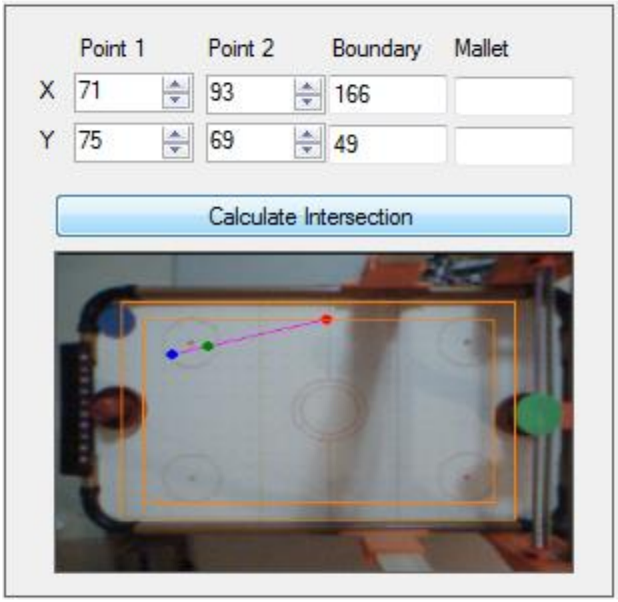
10

☑ Show Response

Figure 16 - Communication Interface Section 7 - Serial Transmit Settings

The serial transmit section controls the port connection setting and packet sent to the motor control hardware.

6.1.9 Path Prediction Tester



	Point 1	Point 2	Boundary	Mallet
X	71	93	166	
Y	75	69	49	

Figure 17 - Communication Interface Section 8 - Path Detection Algorithm Tester

This panel is designed for testing the path prediction algorithm.

6.1.10 Color Filter Examples

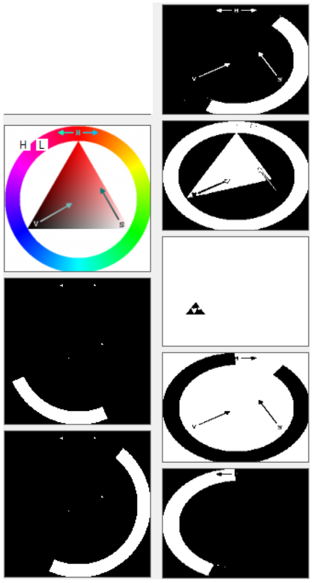


Figure 18 - Communication Interface Section 9 - Color Filter Examples

Section 9 panel contains pictures filtered by HSV filter and display the result for color filter parameter tuning convenience.

6.2 Inputs and Outputs

6.2.1 Inputs

1. Information generated by vision system.
   a. Estimated puck position
   b. Predicted path of travel
   c. Predicted intersection point
   d. Motor control firmware debugging output
2. User specified motor control parameters.
   a. Working space dimensioning (location of goal, boundary size)

6.2.2 Outputs

1. Packet of essential information to motor controller
   a. Current mallet position
   b. Predicted mallet destination

6.3 Parameters

Packet design	Optimize amount of data transferred. The Packet contains three bytes (Start Byte, Mallet Destination Location, and Mallet Current Location). End byte is not used because the values will never be larger than 255 for the frame resolution is only 240 in height.
Serial Baud Rate	A text box is used for default baud rate when com port is connected.
Additional Boundary Limits for Mallet Position	Three values (Mallet Lower Limit, Mallet Centre, Mallet Upper Limit) for mallet position saturation limit are used to make sure that the mallet never goes to a position where it becomes too far to defend the goal. These values are currently set manually and can be implemented to be automatic in the future if so desired.

6.4 Testing and Results

Testing features were implemented to provide intermediate testing of each functions.

1. Changeable Bounce Boundary Numeric Control

Some numeric control elements were used for displaying the user set boundary points and changing the boundary in case it needs to be changed.

2. Constant Target Position

A constant target position mode for data output is implemented to test the reliability of motor driving control.

3. Packet Transmit Mode

Packet transmit mode can be changed to be continuous or sent by button click. This helps debugging the motor driving problem and also the controller design.

The system is functional for the scope of this project.

7. Functional Requirement #4 - Motor control hardware

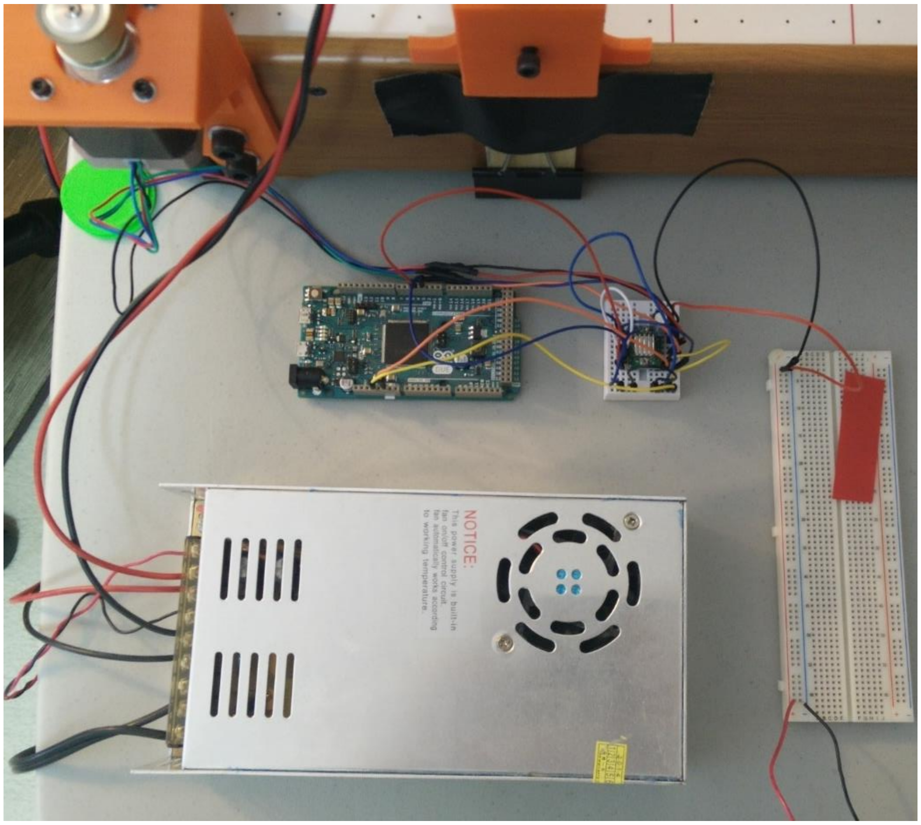


Figure 19 – Overview of motor driver hardware setup

7.1 Approach and Design

The objective of motor control hardware includes designing and sizing hardware components for powering the motors and controlling the motors. All of the motor hardware components were available already from an existing 3D printer. These components were adequate for our project and took precedence over purchasing new hardware.

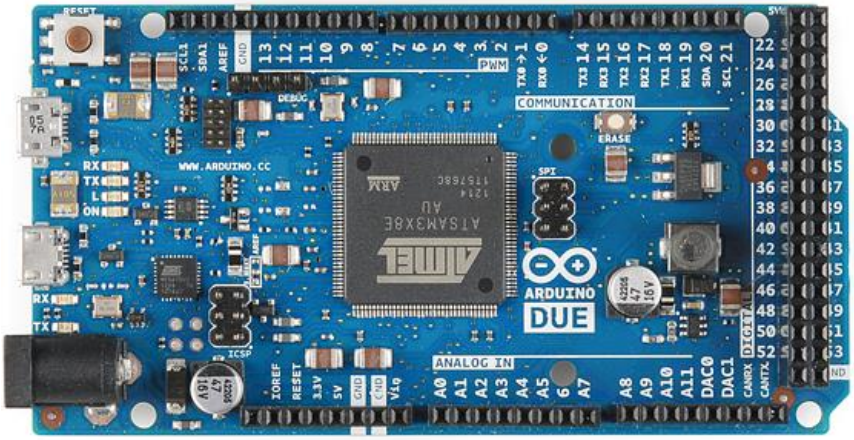


Figure 20 - Arduino for controlling stepper motor

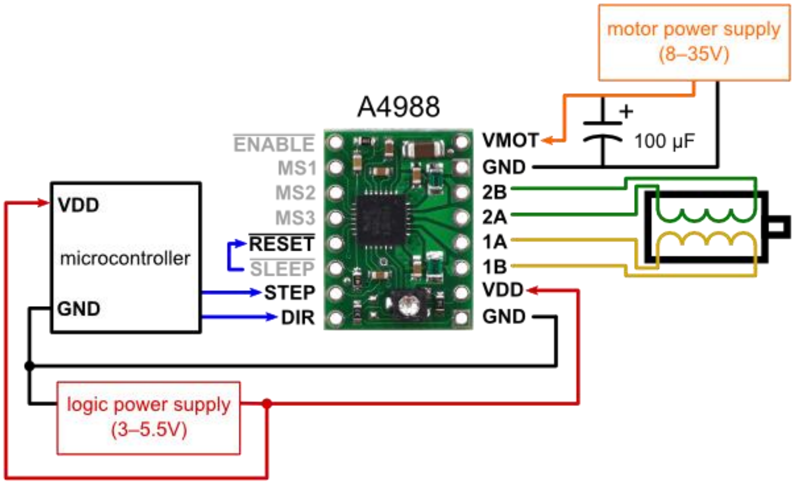


Figure 21 - A4988 Stepper motor driver

The Arduino Due was used as the controller and the A4988 Stepper Motor Driver for powering the NEMA 17 stepper motor. The A4988 motor driver was wired according to Figure 21. A bypass capacitor was added in parallel with the power supply to reduce noise.

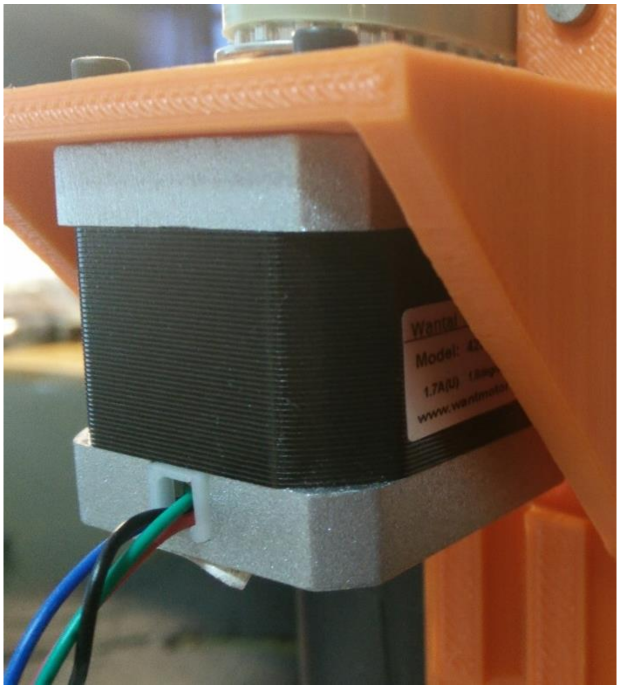


Figure 22 - Stepper motor used for moving the hockey mallet

A stepper motor was used because it was available for the project at the time. A DC motor would have sufficed for this project as well.

7.2 Inputs and Outputs

Inputs

1. 12 V DC (stepped down with a power supply)
2. Serial data stream from computer
3. Target position and current position from serial data

Outputs

1. Voltage to stepper motors
2. Frequency of PWM signal to motor

7.3 Parameters

Motor operating speed	The operating speed will depend on the motor as well as what speeds we will need to move the mallet before the puck hits.
Motor size/type	Depends on how much power we need
Motor torque	It will need to be high enough to overcome any friction and forces in the system such as the puck hitting the mallet.

7.4 Testing and Results

Once the all the physical hardware was mounted on the air hockey table, we were able to test the motor hardware and check that everything is powered correctly.

Testing Functionality by Manually Stepping

To check if everything is powered correctly, the motor was manually stepped by applying 3.3 V (high) to the STEP pin on the A4988 motor driver. Once this was working, the next step would be to output a signal to the STEP pin from the Arduino.

Testing Functionality with Varying Frequency Signal from Arduino

The Arduino code was tested initially by outputting a constant frequency and moving the stepper motor at constant velocity. We adjusted the speed by manually entering values from the serial monitor in the Arduino IDE. This was successful and the next step was reading packets sent from the vision processing algorithm.

8. Functional Requirement #5 - Stepper Motor Control Software

8.1 Approach and Design

The objective of stepper motor control and driving is to move the air hockey mallet to where the puck will be and deflect the puck back at the player. We will need at least 1 stepper motor to drive the mallet horizontally across the table. An encoder will also be needed to determine the current position of the mallet. The software needed will have to energize the stepper motor in the proper sequence and speed. This is important to move the mallet to the position where we determined the puck will be.

8.1.1 Controller Feedback

An encoder was determined to be out of our budget. Instead, we used the camera and computer vision processing algorithm to provide feedback on the position of the mallet. A proportional-integral (PI) controller was used for controlling the position of the mallet. The gains K_p and K_I would need to be tuned.

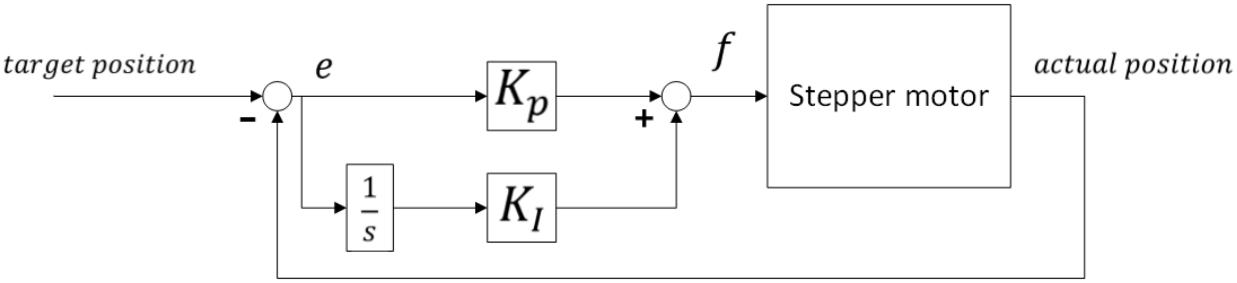
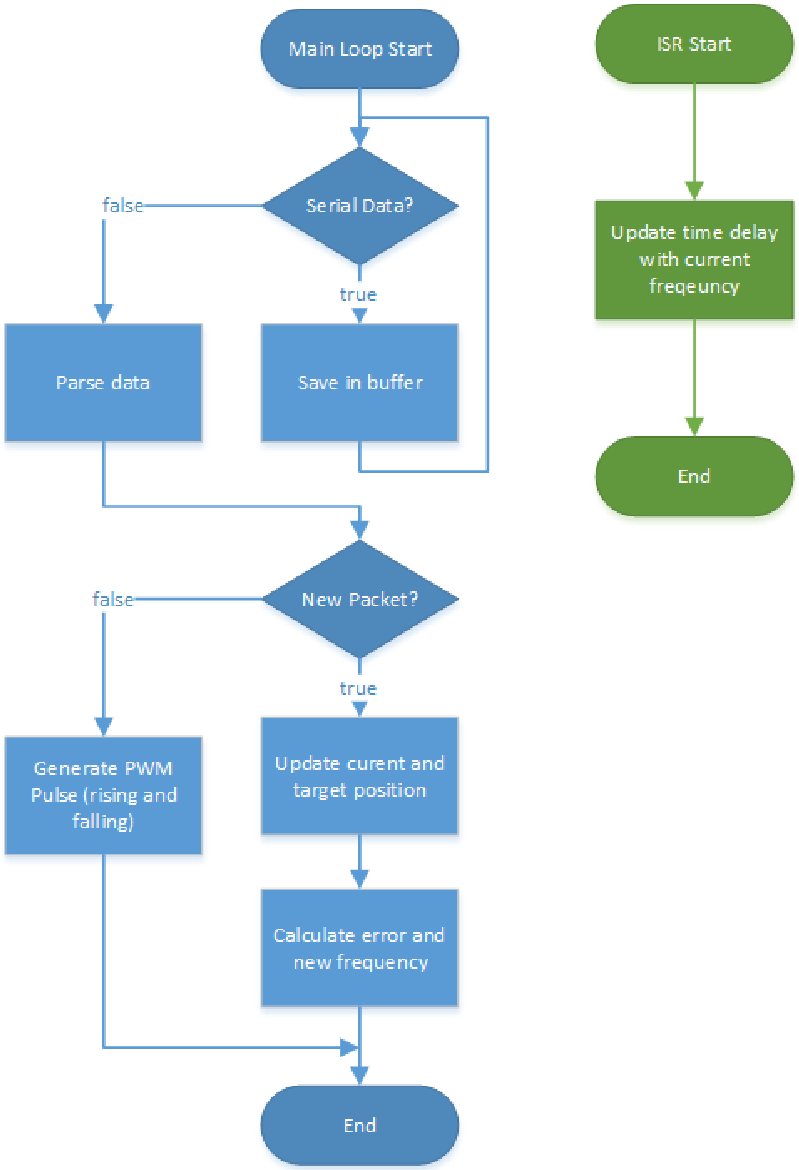


Figure 23 – Block diagram for controller

The error e is multiplied by proportional and integral gains K_p and K_I. The result is the frequency f that the stepper motor is stepped at to adjust the speed of the motor.

8.1.2 Code

The Arduino code uses a timer interrupt to update the PWM frequency to the stepper motor driver and the main loop to generate the PWM pulses and parse serial data from the desktop computer.



Main Loop Start
Serial Data?
false
true
Parse data
Save in buffer
New Packet?
false
true
Generate PWM Pulse (rising and falling)
Update curent and target position
Calculate error and new frequency
End
ISR Start
Update time delay with current freqeuncy
End

Figure 24 - Overview of code

The interrupt service routine was triggered every 1 ms. It was triggered frequently enough to update the PWM frequency to the stepper motor driver without slowing down the main loop.

8.2 Inputs and Outputs

Inputs

1. Position of mallet along horizontal axis
2. Predicted position of the puck along the horizontal axis

Outputs

3. PWM Frequency to the stepper motor driver
4. Power for the digital logic on the stepper motor driver

8.3 Parameters

Speed of control algorithm	Depending on how we implement the control algorithm to decide what signal to send to the stepper motor we can dictate the response time
Controller type (P, PI, PID)	The gains of each type of controller depending on which one we use will need to be tuned.
Packet Processing	Parse data and keep the most recent value. The Packet contains three bytes (Start Byte, Mallet Destination Location, and Mallet Current Location). Bytes will never be larger than 255 for the frame resolution is only 240px in height.

8.4 Testing and Results

The gains K_p and K_I of the controller were tuned manually. There were difficulties in determining the experimental transfer function of the mechanical plant. The manually tuned values were sufficient enough to move the mallet to the correct position.

Setting K_p and K_I

The K_p value was adjusted until the stepper motor could move without stalling when asked to move from one side of the table to the other side (maximum error). The K_I value was then adjusted to increase acceleration and the K_p was further tuned to avoid stalling. The system is asked to move and we observed the response. This allowed us to tune both gains to achieve a fast response and no overshoot.

9. System Evaluation

The functionality of the system matches the scope of the project from the proposal. However, in the proposal we did not define a specific measure of performance such as defending x number of shots before the user can score on the robot. The proposal only claimed to be able to defend the goal from shots. For a normal game, it is difficult to score on the robot unless the player makes fast shots too fast for the robot to respond.

10. Reflections

A set-back that happened was with the stepper motor control. Initially we were going to control the motor with feedback from an encoder, but we found encoders to be quite expensive. We decided it was not necessary and tried controlling the stepper motor in open loop and counting the number of steps. We found that without proper trajectory generation the stepper motor can lose step at constant speed starting and stopping. We believe this may be due to the torque caused by sudden stopping and then reversing the motor. The correct step position might be lost in that instant.

The project went also exactly as planned in the scope of our original proposal. If we were to do the project again with what we know now, we could probably implement a 2 axis machine to enable the robot to attack rather than just defend.

Some lessons learned were to remember to match hardware logic voltages or chips get damaged. There was also a significant learning curve for implementing the computer vision library. The lesson there is that there are plenty of libraries that may accomplish what you're trying to do, but they may not be well documented making the libraries difficult to use.

Most Useful Lessons from MECH 423

1. Serial communication and buffers

2. Timers and Interrupts

3. Motor Control

Things to learn going forward

1. Model the transfer function of the stepper motor or learn how to get an experimental transfer function of the stepper motor. It's not the same as what we did for characterizing a DC motor in our labs.

2. Self-tuning controllers

11. Appendix A – Motor Control Software

Arduino Code

```
const int pinHigh = 9;
const int pinLow = 8;
const int pinDir = 4;
const int pinPWM = 3;
const int pinLED = 13;
volatile boolean  LED_flag = false;
const int START_SPEED = 2000; // Hz
const int MAX_SPEED = 45000;   // HZ


/*==================== Physical Parameters for converting pixels to count
===================================*/
/*======================== (not necessary if only using pixel for control)
==================================*/
const float pi                = 3.14159;
const float pulleyDiameter  = 19.26;            // [mm] diameter
const float pulleyCirc      = pulleyDiameter * pi; // [mm] circumference

const float maxDistance     = 230;          // [mm] total physical distance
the
// mallet is moving

const int steppingMode      = 8;             // microstepping 1/16th
const int stepPerRev        = 200;            // NEMA 17 stepper motor -
steps/revolution
const float countPerLength  = stepPerRev * steppingMode / pulleyCirc; //
count(steps) per length [mm]
const int maxCount          = countPerLength * maxDistance;       // maximum
number of counts (mapped
// to maxDistance)

const int pixelResolution   = 160;                                 // 160 px
(using 40-240 out of 0-240px)
const float lengthPerPixel  = maxDistance / pixelResolution;       // [mm]
length per pixel
const int pixelToCount      = lengthPerPixel * countPerLength;     //
conversion from px to count

/*==================================== Global Variables for Positioning
=====================================*/
volatile int positionIndex      = 40 * pixelToCount;  // Used for tracking
number of steps the motor has taken
volatile int positionIndexCam   = 0;
volatile int positionTarget     = 40 * pixelToCount;  // Assume we start in
the middle of playing area
volatile int positionTargetlast = 0;

byte b;
byte b1 = 0;
int b_int = 0;
volatile int b2 = 0;
volatile int b3 = 0;
volatile int movingFlag     = 0;               // Flag to determine if
system is moving or not
volatile boolean fullPacketFlag     = false;

/*==================================== Control
===============================================================*/
volatile int A              = 50;      // Frequency increment
volatile long timeMicro     = 0;        // Used to keep track of time
float timePeriod            = 0;        // Hold the calculated delay for PWM

volatile int Kp = 200;                  // Proportional Gain
volatile double Ki = 0.0000015;                   // Integrator Gain
volatile int error = 0;                 // Error between target position and
current position
volatile int errorThreshold = 180;            // Threshold to begin adding on
frequency
volatile double errorSum = 0;

volatile long currentTime = 0;
volatile long lastTime = 0;
volatile long timeChange = 0;

volatile long output = 0;

/*======================================= Circular Buffer Variables
=========================================*/
volatile int packetBuffer[10];        // Declare size of buffer array
volatile int bufferMax = 10;          // Maximum size of buffer
volatile int bufferSize = 0;          //
volatile int bufferStart = 0;
volatile int bufferEnd = 0;

volatile int packetStart = 0;         // Index for start of latest packet

/*================================== Timer
===================================================================*/
// Interrupt variables
volatile uint32_t frequency_PWM = START_SPEED; // this value divide by 2 for
actual pwm freq
volatile boolean  PWM_flag = false;

/*================================== Timer ISR
===============================================================*/
void TC6_Handler() {  // GPS timer
  TC_GetStatus(TC2, 0); // Accept interrupt

  // Increase speed (PWM frequency)
  if (output > MAX_SPEED) {
    output = MAX_SPEED;
  }
  if ( frequency_PWM >= 0 && frequency_PWM < MAX_SPEED && movingFlag == 1) {
    /*if( output > errorThreshold){
      frequency_PWM = frequency_PWM + A;
    }
    else {
      */
      frequency_PWM = output;
      //digitalWrite(pinLED, LED_flag = !LED_flag);
    //}
  }
  else {
    // do not change frequency
  }
}

/*================================== Timer setup
==================================*/
void startTimer(Tc *tc, uint32_t channel, IRQn_Type irq, uint32_t frequency)
{
  pmc_set_writeprotect(false);
  pmc_enable_periph_clk((uint32_t)irq);
  TC_Configure(tc, channel, TC_CMR_WAVE | TC_CMR_WAVSEL_UP_RC |
TC_CMR_TCCLKS_TIMER_CLOCK4);
  uint32_t rc = VARIANT_MCK / 128 / frequency; //128 because we selected
TIMER_CLOCK4 above
  TC_SetRA(tc, channel, rc / 2); //50% high, 50% low
  TC_SetRC(tc, channel, rc);
  TC_Start(tc, channel);
  tc->TC_CHANNEL[channel].TC_IER = TC_IER_CPCS;
  tc->TC_CHANNEL[channel].TC_IDR = ~TC_IER_CPCS;
  NVIC_EnableIRQ(irq);
}


void setup() {
  // initialize serial communication at 9600 bits per second:
  Serial.begin(57600);

  // Set modes for digital pins
  // Pins for turning on DRV884 chip on MECH423 Motor Board
  pinMode(pinHigh, OUTPUT);    // HIGH
  pinMode(pinLow, OUTPUT);    // LOW
  pinMode(pinDir, OUTPUT);    // DIRECTION
  pinMode(pinPWM, OUTPUT);    // PWM
  pinMode(pinLED, OUTPUT);    // LED for blinking

  // Initialize pins
  digitalWrite(pinHigh, HIGH);
  digitalWrite(pinLow, LOW);
  digitalWrite(pinDir, HIGH);  // DIR = 1
  //digitalWrite(pinLED, HIGH);

  // Timer Setup
  startTimer(TC2, 0, TC6_IRQn, 10000); // Timer freq is twice PWM freq

}

void loop() {

  /*
    // Read value from serial and set as PWM frequency
    if(Serial.available()){
      frequency_PWM = Serial.parseInt();
      Serial.println(frequency_PWM);
      startTimer(TC2, 0, TC6_IRQn, frequency_PWM);
      //movingFlag = 1; // set flag to zero to see if motor stops
      //digitalWrite(pinDir, LOW);  // change from HIGH to LOW
    }
  */

      /*====================== Update PWM frequency
=========================================== */
  if ( timeMicro > micros() ) {  // Deal with overflow
    timeMicro = micros();
  }

  // If the time interval has passed then execute one high/low pulse
  if (frequency_PWM == 0) {
    frequency_PWM = 1;
    timePeriod = (1 / (float)frequency_PWM) * 1000000.0;
  }
  else {
    timePeriod = (1 / (float)frequency_PWM) * 1000000.0;
  }


  if ( (micros() - timeMicro) > timePeriod ) {

    timeMicro = micros();

    if ( positionIndex == positionTarget ) {
      movingFlag = 0;
      //errorSum = 0;
      frequency_PWM = START_SPEED;    // Reset to initial frequency
(velocity)
    }

    else if ( positionIndex < positionTarget ) {
      digitalWrite(pinDir, LOW);   // go

      movingFlag = 1;
    }

    else if ( positionIndex > positionTarget ) {
      digitalWrite(pinDir, HIGH);    // go back


      movingFlag = 1;

    }

       /*================ Generate PWM (turn pin HIGH/LOW)
================================*/
    if ( movingFlag == 0 ) {
      digitalWrite(pinPWM, LOW);
    }
    else {
      digitalWrite(pinPWM, HIGH);
      delayMicroseconds(timePeriod);
      digitalWrite(pinPWM, LOW);
    }
    //Serial.print(positionIndex);
    //Serial.print('\n');
  }

      /*==================== Read from serial
=============================================*/
  while (Serial.available()) {
    if (bufferSize < bufferMax) {
      b = Serial.read();
      packetBuffer[bufferEnd] = b;

      bufferEnd = (bufferEnd + 1) % bufferMax;
      bufferSize++;
    }
  }

  // Only parse buffer with at least least 3 bytes (start, postionTarget,
positionIndex)
  if ( bufferSize >= 3) {

    while ( bufferSize > 0 ) {
      if (packetBuffer[bufferStart] == 255 && ((bufferStart + 2) % bufferMax)
<= bufferEnd) {
        // save start index of last complete packet
        packetStart = bufferStart;
      }
      bufferStart = (bufferStart + 1) % bufferMax;
      bufferSize--;

    }

    if (packetBuffer[packetStart] == 255 && ((packetStart + 2) % bufferMax)
<= bufferEnd) {
      positionTarget = packetBuffer[(packetStart + 1) % bufferMax];
      positionIndexCam = packetBuffer[(packetStart + 2) % bufferMax];
      //Serial.print(positionTarget);
      fullPacketFlag = 1;
    }

  }

  //b = Serial.read();
  //positionTarget = b;
  //b1 = Serial.read();
  //Serial.println(b1);

/*========================= Update current position and target position
=================== */
  if (fullPacketFlag == true) {

    // Update target
    if ( positionTarget > 200) {
      positionTarget = 200;
    }
    else if (positionTarget < 40) {
      positionTarget = 40;
    }
    //positionTarget = pixelToCount * positionTarget;
    positionTarget = positionTarget;

    // Update count index
    if ( positionIndex > 200) {
      positionIndex = 200;
    }
    else if (positionIndex < 40) {
      positionIndex = 40;
    }
    //positionIndex = pixelToCount * positionIndexCam;
    positionIndex = positionIndexCam;

    // Reset flag
    fullPacketFlag = 0;

    // Controller
    currentTime = micros();
    timeChange = currentTime - lastTime;

    error = positionTarget - positionIndex;
    errorSum = errorSum + (error * timeChange);
    output = abs( (Kp*error) + (Ki*errorSum) );

    lastTime = currentTime;

    Serial.print(error);
  }

}
```

12. Appendix B – C# Software

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Diagnostics;
using System.Threading;
using System.Windows.Forms;
using System.IO;
using Emgu.CV;
using Emgu.CV.Util;
using Emgu.CV.CvEnum;
using Emgu.CV.Structure;


namespace CameraCapture
{
    public partial class lblConvertTest : Form
    {
        #region Globle Var
        //declaring global variables

        const int frameFps = 60;
        const double interval = 1 / 60.0;
        const int frameWidth = 320;
        const int frameHeight = 240;
        const int MALLET_LIMIT_LOWER = 88;
        const int MALLET_LIMIT_UPPER = 146;
        const int MALLET_LIMIT_CENTER = 120;
        const int ALPHA_PUCK_P = 27;
        const int ALPHA_PUCK_V = 27;
        const int ALPHA_MALLET_P = 27;
        const int BAUDRATE = 57600;
        const int RETURNCOUNTTH = 10;

        int malletDestinationRange = 25;
        int malletCurrentRange = 2;
        double puckJitterRange = 1.5;

        bool isFrameSkipped = false;
        // Saving data to file
        StreamWriter outputFile;
        string pathName = @"C:\Dropbox\Mech 5_2\423\Final Project\Vision\CameraCapture
EMGU 3_1\CameraCapture\bin\Debug";
        bool initializeFile = false;

        int upperR = 0,
            upperG = 0,
            upperB = 0,
            lowerR = 0,
            lowerG = 0,
            lowerB = 0;
        Mat mat = new Mat();
        private Capture capture;
        private bool captureInProgress;
        Image<Bgr, Byte> ImageFrame = new Image<Bgr, Byte>(frameWidth, frameHeight);
        Image<Hsv, Byte> hsvImage = new Image<Hsv, Byte>(0, 0);
        Image<Bgr, Byte> ImageHSVwheel = new Image<Bgr, Byte>("HSV-Wheel.png");
        Image<Bgr, Byte> ImageHSVwheel_filtered = new Image<Bgr, Byte>("HSV-Wheel.png");

        Image<Bgr, byte> img_bitmap;
        Image<Bgr, Byte> img = new Image<Bgr, Byte>(frameWidth, frameHeight);

        Mat uimage = new Mat();
        Mat rimage = new Mat();
        Mat pyrDown = new Mat();
        static Size frameSize = new Size(frameWidth, frameHeight);
        VideoWriter file_Quarry = new VideoWriter(@"quarry.avi", frameFps, frameSize, 
true);

        int diff_LH;
        long frameIndex = 1;

        double puckX = 0;
        double puckY = 0;
        double puckX_last = 0;
        double puckY_last = 0;
        double puckX_velocity = 0;
        double puckY_velocity = 0;
        double malletX = 0;
        double malletY = 0;

        double puck_alpha = 0.5;
        double puck_alpha_V = 0.5;
        int framesSkipped = 1;
        int puckRadius = 0;

        int tableBondaryPt1X = 0;
        int tableBondaryPt1Y = 0;
        int tableBondaryPt2X = 0;
        int tableBondaryPt2Y = 0;
        int tableBondaryPt1X_clicked = 0;
        int tableBondaryPt1Y_clicked = 0;
        int tableBondaryPt2X_clicked = 0;
        int tableBondaryPt2Y_clicked = 0;
        int tableBondaryPt1X_puckBounce = 0;
        int tableBondaryPt1Y_puckBounce = 0;
        int tableBondaryPt2X_puckBounce = 0;
        int tableBondaryPt2Y_puckBounce = 0;

        bool pt1_selected = false;
        bool pt2_selected = false;

        bool serialTransmit_flag = false;
        bool serialTransmitSaveData_flag = false;

        bool serialTransmit_flag_timer = false;
        int destinationLocation_int = 0;
        int currentLocation_int = 0;
        byte destinationLocation = 0;
        byte currentLocation = 0;
        byte malletPreviousTargetLocationY = 0;
        byte malletPreviousCurrentLocationY = 0;

        int returnCount = 0;

        bool ib3C;
        int ib3C_HV, ib3C_Lable_X, ib3C_Lable_Y;
        #endregion

        double ms = (DateTime.Now - DateTime.MinValue).TotalMilliseconds;
        //Process.GetCurrentProcess().ProcessorAffinity = new IntPtr(2); // Use only the
second core
        Stopwatch stopWatch = new Stopwatch();


        private void CameraCapture_Load(object sender, EventArgs e)
        {
            stopWatch.Start();

            ckbByte1.Checked = false;
            ckbByte2.Checked = false;
            ckbByte3.Checked = false;
            ckbByte4.Checked = false;
            ckbByte5.Checked = false;
            txtByte1.Enabled = false;
            txtByte2.Enabled = false;
            txtByte3.Enabled = false;
            txtByte4.Enabled = false;
            txtByte5.Enabled = false;

            txtBaudRate.Text = BAUDRATE.ToString();

            ckbShowResponse.Checked = true;
            ckbSendByte.Checked = true;
            ckbBackToCenter.Checked = false;
            ckbOverlayPuckDetection.Checked = true;
            ckbOverlayMalletDetection.Checked = true;
            ckbPacketContent.Checked = true;
            ckbUseConstantTargetPos.Checked = false;

            nmc_Puck_Alpha_P.Value = ALPHA_PUCK_P;
            nmc_Puck_Alpha_V.Value = ALPHA_PUCK_V;
            nmc_Mallet_Alpha_P.Value = ALPHA_MALLET_P;

            nmcMalletCenter.Value = MALLET_LIMIT_CENTER;
            nmcMalletLowerLimit.Value = MALLET_LIMIT_LOWER;
            nmcMalletUpperLimit.Value = MALLET_LIMIT_UPPER;

            nmc_returnCountTH.Value = RETURNCOUNTTH;

            ComPortUpdate();
        }

        public lblConvertTest()
        {
            InitializeComponent();
            img_HSVwheel.Image = ImageHSVwheel;
            img_HSVwheel_filtered.Image = ImageHSVwheel_filtered;
            img_HSVwheel_filtered2.Image = ImageHSVwheel_filtered;
        }

        private void ProcessFrame(object sender, EventArgs arg)
        {



            mat = capture.QueryFrame();
            ImageFrame = mat.ToImage<Bgr, Byte>();
            img = mat.ToImage<Bgr, Byte>();

            CvInvoke.CvtColor(img, uimage, ColorConversion.Bgr2Gray);

            //use image pyr to remove noise

            CvInvoke.PyrDown(uimage, pyrDown);
            CvInvoke.PyrUp(pyrDown, uimage);

            Hsv lowerLimit = new Hsv(upperR, upperG, upperB);
            Hsv upperLimit = new Hsv(lowerR, lowerG, lowerB);
            Image<Hsv, byte> hsv = mat.ToImage<Hsv, byte>();
            Image<Gray, byte> imageHSVDest = hsv.InRange(lowerLimit, upperLimit);

            img_bitmap = mat.ToImage<Bgr, Byte>();
            //Image <Bgr, byte> blur = img_bitmap.SmoothBlur(10, 10, true);
            //Image<Bgr, byte> mediansmooth = img_bitmap.SmoothMedian(15);
            //Image<Bgr, byte> bilat = img_bitmap.SmoothBilatral(7, 255, 34);
            Image<Bgr, byte> gauss = img_bitmap.SmoothGaussian(3, 3, 34.3, 45.3);

            ImageProcessing();

            #region Send Data thru Serial
            if (ckbTransmit.Checked )
            {

                if (ckbUseConstantTargetPos.Checked)
                {
                    #region Constant target position
                    // Convert target location
                    destinationLocation =
checkBoundary_intToByte(Convert.ToInt16(nmc_malletTargetPosTest.Value));

                    if (txtMalletCurrentY.Text != "")
                    {
                        // Convert current location
                        currentLocation =
checkBoundary_intToByte(Convert.ToInt16(txtMalletCurrentY.Text));
                    }

                    if (serialPort1.IsOpen)
                    {
                        if (ckbPacketContent.Checked)   // Send both target and current
location
                        {
                            serialTransmit(destinationLocation, currentLocation);
                        }
                        else    // Send only target location
                        {
                            serialTransmit(destinationLocation);
                        }

                        malletPreviousTargetLocationY = destinationLocation;
                        malletPreviousCurrentLocationY = currentLocation;

                    }
                    else
                    {
                        // Commented out for now
                        #region Back to center
                        /*
                        if (ckbBackToCenter.Checked)
                        {
                            destinationLocation = Convert.ToByte(nmcMalletCenter.Value);
// Set target location to be in the center

                            currentLocation = 0;
                            if (txtMalletCurrentY.Text != "")
                            {
                                currentLocation_int =
Convert.ToInt16(txtMalletCurrentY.Text);

                                if (currentLocation_int > 0 && currentLocation_int < 255)
// Limit number to byte range
                                {
                                    currentLocation =
Convert.ToByte(currentLocation_int);
                                }
                            }

                            if (serialPort1.IsOpen)
                            {
                                if (ckbPacketContent.Checked)   // Send both target and
current location
                                {
                                    serialTransmit(destinationLocation, currentLocation);
                                }
                                else   // Send only target location
                                {
                                    serialTransmit(destinationLocation);
                                }
                            }

                            malletPreviousTargetLocationY = destinationLocation;
                            malletPreviousCurrentLocationY = currentLocation;
                        }
                        */
                        #endregion
                    }
                    serialTransmitSaveData_flag = true;
                    serialTransmit_flag_timer = false;
                    #endregion
                }
                else
                {
                    #region Real time target position
                    if (txtMalletTargetY.Text != "")
                    {

                        // Convert target location
                        destinationLocation = 
checkBoundary_intToByte(Convert.ToInt16(txtMalletTargetY.Text));
                        if (destinationLocation != malletPreviousTargetLocationY)
                        {
                            returnCount = 0;
                        }

                    }
                    if (txtMalletCurrentY.Text != "")
                    {
                        // Convert current location
                        currentLocation = 
checkBoundary_intToByte(Convert.ToInt16(txtMalletCurrentY.Text));
                    }

                    if (ckbBackToCenter.Checked &&  returnCount >= 
Convert.ToInt16(nmc_returnCountTH.Value))
                    {
                        destinationLocation = 
checkBoundary_intToByte(Convert.ToInt16(nmcMalletCenter.Value));  // Set target location 
to be in the center
                    }

                    if (serialPort1.IsOpen)
                    {
                        if (ckbPacketContent.Checked)   // Send both target and current 
location
                        {
                            serialTransmit(destinationLocation, currentLocation);
                        }
                        else    // Send only target location
                        {
                            serialTransmit(destinationLocation);
                        }

                        malletPreviousTargetLocationY = destinationLocation;
                        malletPreviousCurrentLocationY = currentLocation;
                    }
                    //txtMalletTargetY.Text = "";

                    serialTransmitSaveData_flag = true;
                    serialTransmit_flag_timer = false;

                    #endregion
                }

            }
            if (serialTransmit_flag)
            {
                serialTransmit();
            }
            #endregion

            #region Elapsed time
            //stopWatch.Stop();
            TimeSpan ts = stopWatch.Elapsed;
            string elapsedTime = String.Format("{0:00}:{1:00}:{2:00}.{3:000}",
            ts.Hours, ts.Minutes, ts.Seconds, ts.Milliseconds);

            txtTimeElapsed.AppendText(elapsedTime + "\n");
            #endregion

            #region Saving Data to CSV
            try
            {
                if (ckbSaveData.Checked == true)
                {
                    // Initialize headings in file
                    if (initializeFile == false)
                    {
                        // Setup labels
                        outputFile.Write("Frame Index"); outputFile.Write(",");
                        outputFile.Write("Elapsed Time"); outputFile.Write(",");
                        outputFile.Write("FPS"); outputFile.Write(",");
                        outputFile.Write("Width"); outputFile.Write(",");
                        outputFile.Write("Height"); outputFile.Write(",");

                        outputFile.Write("Puck Position X"); outputFile.Write(",");
                        outputFile.Write("Puck Position Y"); outputFile.Write(",");
                        outputFile.Write("Puck Position Alpha"); outputFile.Write(",");

                        outputFile.Write("Puck Velocity X"); outputFile.Write(",");
                        outputFile.Write("Puck Velocity Y"); outputFile.Write(",");
                        outputFile.Write("Puck Velocity Alpha"); outputFile.Write(",");

                        outputFile.Write("Mallet Position X"); outputFile.Write(",");
                        outputFile.Write("Mallet Position Y"); outputFile.Write(",");
                        outputFile.Write("Mallet Position Alpha"); outputFile.Write(",");

                        outputFile.Write("TableBoundary1X"); outputFile.Write(",");
                        outputFile.Write("TableBoundary1Y"); outputFile.Write(",");
                        outputFile.Write("TableBoundary2X"); outputFile.Write(",");
                        outputFile.Write("TableBoundary2Y"); outputFile.Write(",");
                        outputFile.Write("Boundary Collision X"); outputFile.Write(",");
                        outputFile.Write("Boundary Collision Y"); outputFile.Write(",");
                        outputFile.Write("Mallet Target X"); outputFile.Write(",");
                        outputFile.Write("Mallet Target Y"); outputFile.Write(",");
                        outputFile.Write("Mallet Current X"); outputFile.Write(",");
                        outputFile.Write("Mallet Current Y"); outputFile.Write(",");
                        outputFile.Write("Puck Radius"); outputFile.Write(",");
                        outputFile.Write("Serial - Mallet Target Location");
outputFile.Write(",");
                        outputFile.Write("Serial - Mallet Current Location");
                        outputFile.Write("\n");

                        initializeFile = true;
                    }
                    outputFile.Write(frameIndex.ToString()); outputFile.Write(",");
                    outputFile.Write((DateTime.Now -
DateTime.MinValue).TotalMilliseconds.ToString()); outputFile.Write(",");
                    //outputFile.Write(ts.Milliseconds.ToString());
outputFile.Write(",");
```

```
                    outputFile.Write(lblfps.Text.ToString()); outputFile.Write(",");
                    outputFile.Write(lblWidth.Text.ToString()); outputFile.Write(",");
                    outputFile.Write(lblHeight.Text.ToString()); outputFile.Write(",");

                    outputFile.Write(txt_Puck_P_X.Text); outputFile.Write(",");
                    outputFile.Write(txt_Puck_P_Y.Text); outputFile.Write(",");
                    outputFile.Write(nmc_Puck_Alpha_P.Value.ToString());
outputFile.Write(",");

                    outputFile.Write(txt_Puck_V_X.Text); outputFile.Write(",");
                    outputFile.Write(txtPUCK_V_Y.Text); outputFile.Write(",");
                    outputFile.Write(nmc_Puck_Alpha_V.Value.ToString());
outputFile.Write(",");

                    outputFile.Write(txt_Mallet_P_X.Text); outputFile.Write(",");
                    outputFile.Write(txt_Mallet_P_Y.Text); outputFile.Write(",");
                    outputFile.Write(nmc_Mallet_Alpha_P.Value.ToString());
outputFile.Write(",");

                    outputFile.Write(nmcTableBoundary1X.Value.ToString());
outputFile.Write(",");
                    outputFile.Write(nmcTableBoundary1Y.Value.ToString());
outputFile.Write(",");
                    outputFile.Write(nmcTableBoundary2X.Value.ToString());
outputFile.Write(",");
                    outputFile.Write(nmcTableBoundary2Y.Value.ToString());
outputFile.Write(",");
                    outputFile.Write(txtBoundaryPtX.Text); outputFile.Write(",");
                    outputFile.Write(txtBoundaryPtY.Text); outputFile.Write(",");
                    outputFile.Write(txtMalletTargetX.Text); outputFile.Write(",");
                    outputFile.Write(txtMalletTargetY.Text); outputFile.Write(",");
                    outputFile.Write(txtMalletCurrentX.Text); outputFile.Write(",");
                    outputFile.Write(txtMalletCurrentY.Text); outputFile.Write(",");
                    outputFile.Write(nmcPuckRadius.Value.ToString());

                    if (serialTransmitSaveData_flag)
                    {
                        outputFile.Write(",");
                        outputFile.Write(destinationLocation.ToString());
outputFile.Write(",");
                        outputFile.Write(currentLocation.ToString());
                    }
                    outputFile.Write("\n");
                }

            }
            catch
            { }
            serialTransmitSaveData_flag = false;
            #endregion

            frameIndex++;
        }

        private void ImageProcessing()
        {
            #region Filter
            // Filter puck circle
            Image<Bgr, Byte> ImageFrameDetection2 = new Image<Bgr,
Byte>(ImageFrame.Size);
            Image<Gray, Byte> ImageFrameDetection = cvAndHsvImage(
            ImageFrame,
            Convert.ToInt32(nmc_HL.Value), Convert.ToInt32(nmc_HH.Value),
            Convert.ToInt32(nmc_SL.Value), Convert.ToInt32(nmc_SH.Value),
            Convert.ToInt32(nmc_VL.Value), Convert.ToInt32(nmc_VH.Value),
            ckb_EH.Checked, ckb_ES.Checked, ckb_EV.Checked, ckb_IV.Checked);
            // Filter HSV
            Image<Gray, Byte> HSVwheel_Detection = cvAndHsvImage(
            ImageHSVwheel_filtered,
            Convert.ToInt32(nmc_HL.Value), Convert.ToInt32(nmc_HH.Value),
            Convert.ToInt32(nmc_SL.Value), Convert.ToInt32(nmc_SH.Value),
            Convert.ToInt32(nmc_VL.Value), Convert.ToInt32(nmc_VH.Value),
            ckb_EH.Checked, ckb_ES.Checked, ckb_EV.Checked, ckb_IV.Checked);


            // Filter mallet circle
            Image<Bgr, Byte> ImageFrameDetection2_mallet = new Image<Bgr,
Byte>(ImageFrame.Size);
            Image<Gray, Byte> ImageFrameDetection_mallet = cvAndHsvImage(
            ImageFrame,
            Convert.ToInt32(nmc_HL_mallet.Value), Convert.ToInt32(nmc_HH_mallet.Value),
            Convert.ToInt32(nmc_SL.Value), Convert.ToInt32(nmc_SH.Value),
            Convert.ToInt32(nmc_VL.Value), Convert.ToInt32(nmc_VH.Value),
            ckb_EH.Checked, ckb_ES.Checked, ckb_EV.Checked, ckb_IV.Checked);
            // Filter HSV
            Image<Gray, Byte> HSVwheel_Detection2 = cvAndHsvImage(
            ImageHSVwheel_filtered,
            Convert.ToInt32(nmc_HL_mallet.Value), Convert.ToInt32(nmc_HH_mallet.Value),
            Convert.ToInt32(nmc_SL.Value), Convert.ToInt32(nmc_SH.Value),
            Convert.ToInt32(nmc_VL.Value), Convert.ToInt32(nmc_VH.Value),
            ckb_EH.Checked, ckb_ES.Checked, ckb_EV.Checked, ckb_IV.Checked);

            #endregion

            #region find Hough circles

            double cannyThreshold = Convert.ToDouble(nmc_cannyThreshold.Value);
            double circleAccumulatorThreshold =
Convert.ToDouble(nmc_circleAccumulatorThreshold.Value);
            Image<Bgr, Byte> circleImage = ImageFrameDetection2.Copy();

            // Puck
            Image<Gray, byte> gauss =
ImageFrameDetection.SmoothGaussian(Convert.ToInt16(txt_kernelWidth.Text),
Convert.ToInt16(txt_kernelHeight.Text), Convert.ToDouble(txt_sigma1.Text),
Convert.ToDouble(txt_sigma2.Text));
            CvInvoke.CvtColor(ImageFrameDetection, ImageFrameDetection2,
ColorConversion.Gray2Bgr);
            CvInvoke.CvtColor(gauss, ImageFrameDetection2, ColorConversion.Gray2Bgr);
            CircleF[] circles = CvInvoke.HoughCircles(gauss, HoughType.Gradient, 2.0,
20.0, cannyThreshold, circleAccumulatorThreshold,
Convert.ToInt32(nmcCircleMinRadius.Value), Convert.ToInt32(nmcCircleMaxRadius.Value));

            double circArea = 0;
            CircleF circIndex = new CircleF();
            for (int circ_i = 0; circ_i < circles.Length; circ_i++)
            {
                if (circles[circ_i].Area > circArea)
                {
                    circArea = circles[circ_i].Area;
                    circIndex = circles[circ_i];
                }
            }

            // Mallet
            Image<Gray, byte> gauss_mallet =
ImageFrameDetection_mallet.SmoothGaussian(Convert.ToInt16(txt_kernelWidth.Text),
Convert.ToInt16(txt_kernelHeight.Text), Convert.ToDouble(txt_sigma1.Text),
Convert.ToDouble(txt_sigma2.Text));
            CvInvoke.CvtColor(ImageFrameDetection_mallet, ImageFrameDetection2_mallet,
ColorConversion.Gray2Bgr);
            CvInvoke.CvtColor(gauss_mallet, ImageFrameDetection2_mallet,
ColorConversion.Gray2Bgr);
            CircleF[] circles_mallet = CvInvoke.HoughCircles(gauss_mallet,
HoughType.Gradient, 2.0, 20.0, cannyThreshold, circleAccumulatorThreshold,
Convert.ToInt32(nmcCircleMinRadius.Value), Convert.ToInt32(nmcCircleMaxRadius.Value));

            double circArea_mallet = 0;
            CircleF circIndex_mallet = new CircleF();

            for (int circ_i = 0; circ_i < circles_mallet.Length; circ_i++)
            {
                if (circles_mallet[circ_i].Area > circArea_mallet)
                {
                    circArea_mallet = circles_mallet[circ_i].Area;
                    circIndex_mallet = circles_mallet[circ_i];
                }
            }

            #endregion

            #region Circle location smoothening filter
            isFrameSkipped = true;
            if (circIndex.Center.X != 0) {
                isFrameSkipped = false;
                puck_alpha = Convert.ToDouble(nmc_Puck_Alpha_P.Value)/100.0;

                puckX = circIndex.Center.X * puck_alpha + puckX * (1 - puck_alpha);
                puckY = circIndex.Center.Y * puck_alpha + puckY * (1 - puck_alpha);
                framesSkipped = 1;

                txt_Puck_P_X.Text = Convert.ToInt16(puckX).ToString();
                txt_Puck_P_Y.Text = Convert.ToInt16(puckY).ToString();
            }
            else
            {
                framesSkipped++;
            }

            if(circIndex_mallet.Center.X != 0)
            {
                puck_alpha = Convert.ToDouble(nmc_Mallet_Alpha_P.Value) / 100.0;
                malletX = circIndex_mallet.Center.X * puck_alpha + malletX * (1 -
puck_alpha);
                malletY = circIndex_mallet.Center.Y * puck_alpha + malletY * (1 -
puck_alpha);


                txt_Mallet_P_X.Text = Convert.ToInt16(malletX).ToString();
                txt_Mallet_P_Y.Text = Convert.ToInt16(malletY).ToString();
                txtMalletCurrentX.Text = Convert.ToInt16(malletX).ToString();
                txtMalletCurrentY.Text = Convert.ToInt16(malletY).ToString();
            }

            #endregion

            #region Calculate velocity vector
            PointF center = new PointF(Convert.ToSingle(puckX), Convert.ToSingle(puckY));
            CircleF circIndex_filter = new CircleF(center, circIndex.Radius);

            if( Math.Abs(puckX - puckX_last) > puckJitterRange) //1.8
            {
                puck_alpha_V = Convert.ToDouble(nmc_Puck_Alpha_V.Value) / 100.0;
                //puckX_velocity = ((puckX - puckX_last) / (framesSkipped * interval)) *
puck_alpha_V + puckX_velocity * (1 - puck_alpha_V);
                //puckY_velocity = ((puckY - puckY_last) / (framesSkipped * interval)) *
puck_alpha_V + puckY_velocity * (1 - puck_alpha_V);
                puckX_velocity = ((puckX - puckX_last) *10) * puck_alpha_V +
puckX_velocity * (1 - puck_alpha_V);
                puckY_velocity = ((puckY - puckY_last) *10) * puck_alpha_V +
puckY_velocity * (1 - puck_alpha_V);
            }
            else
            {
                puckX_velocity = 0;
                puckY_velocity = 0;
            }

            txt_Puck_V_X.Text = puckX_velocity.ToString();
            txtPUCK_V_Y.Text = puckY_velocity.ToString();

            Point velocityPt1 = new Point(Convert.ToInt32(circIndex_filter.Center.X),
Convert.ToInt32(circIndex_filter.Center.Y));
            Point velocityPt2 = new Point(Convert.ToInt32(circIndex_filter.Center.X +
puckX_velocity), Convert.ToInt32(circIndex_filter.Center.Y + puckY_velocity));

            // Save puck location for velocity calculation
            puckX_last = puckX;
            puckY_last = puckY;
            #endregion

            #region Draw boundary rectangles, calculate path, draw vector arrowed lines,
and draw circles
            nmcTableBoundary1X.Text = tableBondaryPt1X.ToString();
            nmcTableBoundary1Y.Text = tableBondaryPt1Y.ToString();
            nmcTableBoundary2X.Text = tableBondaryPt2X.ToString();
            nmcTableBoundary2Y.Text = tableBondaryPt2Y.ToString();

            if (pt1_selected && pt2_selected)
            {
                // Boundary corner points
                Point boundaryPt1 = new Point(Convert.ToInt32(tableBondaryPt1X),
Convert.ToInt32(tableBondaryPt1Y));
                Point boundaryPt2 = new Point(Convert.ToInt32(tableBondaryPt1X),
Convert.ToInt32(tableBondaryPt2Y));
                Point boundaryPt3 = new Point(Convert.ToInt32(tableBondaryPt2X),
Convert.ToInt32(tableBondaryPt2Y));
                Point boundaryPt4 = new Point(Convert.ToInt32(tableBondaryPt2X),
Convert.ToInt32(tableBondaryPt1Y));

                // Draw boundary lines on filtered image
                if (ckbFilterDetection.Checked)
                {
                    CvInvoke.Line(circleImage, boundaryPt1, boundaryPt2, new MCvScalar(0,
128, 255, 0));
                    CvInvoke.Line(circleImage, boundaryPt2, boundaryPt3, new MCvScalar(0,
128, 255, 0));
                    CvInvoke.Line(circleImage, boundaryPt3, boundaryPt4, new MCvScalar(0,
128, 255, 0));
                    CvInvoke.Line(circleImage, boundaryPt4, boundaryPt1, new MCvScalar(0,
128, 255, 0));
                }

                // Draw boundary lines on quarry image
                if (ckbQuarryDetection.Checked)
                {
                    CvInvoke.Line(img, boundaryPt1, boundaryPt2, new MCvScalar(0, 128,
255, 0));
                    CvInvoke.Line(img, boundaryPt2, boundaryPt3, new MCvScalar(0, 128,
255, 0));
                    CvInvoke.Line(img, boundaryPt3, boundaryPt4, new MCvScalar(0, 128,
255, 0));
                    CvInvoke.Line(img, boundaryPt4, boundaryPt1, new MCvScalar(0, 128,
255, 0));
                }

                // Puck bounce boundary points
                Point boundaryPt1_puckBounce = new
Point(Convert.ToInt32(tableBondaryPt1X_puckBounce),
Convert.ToInt32(tableBondaryPt1Y_puckBounce));
                Point boundaryPt2_puckBounce = new
Point(Convert.ToInt32(tableBondaryPt1X_puckBounce),
Convert.ToInt32(tableBondaryPt2Y_puckBounce));
                Point boundaryPt3_puckBounce = new
Point(Convert.ToInt32(tableBondaryPt2X_puckBounce),
Convert.ToInt32(tableBondaryPt2Y_puckBounce));
                Point boundaryPt4_puckBounce = new
Point(Convert.ToInt32(tableBondaryPt2X_puckBounce),
Convert.ToInt32(tableBondaryPt1Y_puckBounce));

                // Draw puck bounce boundary lines on filtered image
                if (ckbFilterDetection.Checked)
                {
                    CvInvoke.Line(circleImage, boundaryPt1_puckBounce,
boundaryPt2_puckBounce, new MCvScalar(0, 128, 255, 0));
                    CvInvoke.Line(circleImage, boundaryPt2_puckBounce,
boundaryPt3_puckBounce, new MCvScalar(0, 128, 255, 0));
```

```
                    CvInvoke.Line(circleImage, boundaryPt3_puckBounce,
boundaryPt4_puckBounce, new MCvScalar(0, 128, 255, 0));
                    CvInvoke.Line(circleImage, boundaryPt4_puckBounce,
boundaryPt1_puckBounce, new MCvScalar(0, 128, 255, 0));
                }

                // Draw puck bounce boundary lines on quarry image
                if (ckbQuarryDetection.Checked)
                {
                    CvInvoke.Line(img, boundaryPt1_puckBounce, boundaryPt2_puckBounce,
new MCvScalar(0, 128, 255, 0));
                    CvInvoke.Line(img, boundaryPt2_puckBounce, boundaryPt3_puckBounce,
new MCvScalar(0, 128, 255, 0));
                    CvInvoke.Line(img, boundaryPt3_puckBounce, boundaryPt4_puckBounce,
new MCvScalar(0, 128, 255, 0));
                    CvInvoke.Line(img, boundaryPt4_puckBounce, boundaryPt1_puckBounce,
new MCvScalar(0, 128, 255, 0));
                }


                // Intersection point on boundary
                Point boundaryPt = intersectionBoundaryPoint(velocityPt1, velocityPt2);

                // Calculate collision path
                if (boundaryPt.X != -1 && boundaryPt.Y != -1)
                {

                    if (ckbFilterDetection.Checked)
                        CvInvoke.Line(circleImage, velocityPt1, boundaryPt, new
MCvScalar(255, 0, 255, 0));
                    if (ckbQuarryDetection.Checked)
                        CvInvoke.Line(img, velocityPt1, boundaryPt, new MCvScalar(255, 0,
255, 0));
                }

                if(boundaryPt.X < tableBondaryPt2X_puckBounce)
                {
                    Point malletCrossPt = intersectionMallet(velocityPt1, boundaryPt);
                    if (malletCrossPt.Y != -1)
                    {

                        if (ckbFilterDetection.Checked)
                            CvInvoke.Line(circleImage, boundaryPt, malletCrossPt, new
MCvScalar(255, 0, 255, 0));
                        if (ckbQuarryDetection.Checked)
                            CvInvoke.Line(img, boundaryPt, malletCrossPt, new
MCvScalar(255, 0, 255, 0));

                        txtMalletTargetX.Text = malletCrossPt.X.ToString();
                        txtMalletTargetY.Text = malletCrossPt.Y.ToString();

                    }
                }
                else
                {
                    txtBoundaryPtX.Text = boundaryPt.X.ToString();
                    txtBoundaryPtY.Text = boundaryPt.Y.ToString();
                    txtMalletTargetX.Text = txtBoundaryPtX.Text;
                    txtMalletTargetY.Text = txtBoundaryPtY.Text;
                }
            }

            // Draw Circles
            if (ckbFilterDetection.Checked)
            {
                CvInvoke.ArrowedLine(circleImage, velocityPt1, velocityPt2, new 
MCvScalar(255, 255, 0, 0));
                circleImage.Draw(circIndex_mallet, new Bgr(Color.Green), 2);
                circleImage.Draw(circIndex, new Bgr(Color.Red), 2);
                circleImage.Draw(circIndex_filter, new Bgr(Color.Blue), 2);
            }
            if (ckbQuarryDetection.Checked)
            {
                CvInvoke.ArrowedLine(img, velocityPt1, velocityPt2, new MCvScalar(255, 
255, 0, 0));
                img.Draw(circIndex_mallet, new Bgr(Color.Green), 2);
                img.Draw(circIndex, new Bgr(Color.Red), 2);
                img.Draw(circIndex_filter, new Bgr(Color.Blue), 2);
            }
            if (ckbOverlayPuckDetection.Checked)
            {
                ImageFrameDetection2.Draw(circIndex, new Bgr(Color.Red), 2);
            }
            if (ckbOverlayMalletDetection.Checked)
            {
                ImageFrameDetection2_mallet.Draw(circIndex_mallet, new Bgr(Color.Green), 
2);
            }


            if (ckbQuarryDetection.Checked)
            {
                Point malletLower = new 
Point(tableBondaryPt2X_puckBounce,Convert.ToInt32(nmcMalletLowerLimit.Value));
                Point malletCenter = new Point(tableBondaryPt2X_puckBounce, 
Convert.ToInt32(nmcMalletCenter.Value));
                Point malletUpper = new Point(tableBondaryPt2X_puckBounce, 
Convert.ToInt32(nmcMalletUpperLimit.Value));
                Point malletLower2 = new Point(tableBondaryPt2X_puckBounce - 10, 
Convert.ToInt32(nmcMalletLowerLimit.Value));
                Point malletCenter2 = new Point(tableBondaryPt2X_puckBounce - 10, 
Convert.ToInt32(nmcMalletCenter.Value));
                Point malletUpper2 = new Point(tableBondaryPt2X_puckBounce - 10, 
Convert.ToInt32(nmcMalletUpperLimit.Value));

                MCvScalar myYellow = new MCvScalar(0, 255, 255);
                CvInvoke.Line(img, malletLower, malletLower2, myYellow);
                CvInvoke.Line(img, malletCenter, malletCenter2, myYellow);
                CvInvoke.Line(img, malletUpper, malletUpper2, myYellow);
                CvInvoke.Line(img, malletLower, malletUpper, myYellow);
            }
            #endregion

            #region Draw referece frame coord
            Point coordRef_Origin = new Point(10, 10);
            Point coordRef_Origin_X = new Point(30, 10);
            Point coordRef_Origin_Y = new Point(10, 30);

            // Draw reference frame coord
            CvInvoke.ArrowedLine(img, coordRef_Origin, coordRef_Origin_X, new
MCvScalar(255, 255, 255));
            CvInvoke.ArrowedLine(img, coordRef_Origin, coordRef_Origin_Y, new
MCvScalar(255, 255, 255));
            CvInvoke.PutText(img, "X", new Point(35, 15), FontFace.HersheyComplexSmall,
0.5, new MCvScalar(255, 255, 255));
            CvInvoke.PutText(img, "Y", new Point(10, 40), FontFace.HersheyComplexSmall,
0.5, new MCvScalar(255, 255, 255));
            CvInvoke.PutText(img, frameIndex.ToString(), new Point(45, 15),
FontFace.HersheyComplexSmall, 0.5, new MCvScalar(255, 255, 255));
            #endregion

            #region Resize and display images
            // Resize image for imageBox
            Image<Bgr, Byte> circleImage_re = new Image< Bgr, Byte>
(camFiltered_Puck.Width, camFiltered_Puck.Height);
            CvInvoke.Resize(circleImage, circleImage_re, new Size(camFiltered_Puck.Width,
camFiltered_Puck.Height));

            // Display images it in pictureBox
            CamImageBox.Image = img;
            camFiltered_Circles.Image = circleImage_re;
            camFiltered_Puck.Image = ImageFrameDetection2;
            camFiltered_Mallet.Image = ImageFrameDetection2_mallet;
            img_HSVwheel_filtered.Image = HSVwheel_Detection;
            img_HSVwheel_filtered2.Image = HSVwheel_Detection2;

            // Save frame in video file
            if (ckbRecordVideo.Checked){
                file_Quarry.Write(img.Mat);
            }


            #endregion
        }

        private Point intersectionBoundaryPoint(Point pt1, Point pt2)
        {
            Point ptResult = new Point();

            if(pt1.X > pt2.X) // Vector pointing to player
            {
                ptResult.X = -1;
                ptResult.Y = -1;
                return ptResult;
            }
            if (pt1.X == pt2.X && pt1.Y == pt2.Y) // Zero vector
            {
                ptResult.X = -1;
                ptResult.Y = -1;
                return ptResult;
            }
            if (pt1.X == pt2.X) // Vertical vector
            {
                ptResult.X = -1;
                ptResult.Y = -1;
                return ptResult;
            }

            //calculate line equation
            double pt1X = pt1.X;
            double pt1Y = pt1.Y;
            double pt2X = pt2.X;
            double pt2Y = pt2.Y;
            double pt3X = 0;
            double pt3Y = 0;
            double ptCornerX = 0;
            double ptCornerY = 0;
            double angle_pt1_ptCorner = 0;
            double angle_pt1_Pt2 = 0;

            Point boundaryPt3_puckBounce = new
Point(Convert.ToInt32(tableBondaryPt2X_puckBounce),
Convert.ToInt32(tableBondaryPt2Y_puckBounce));
            Point boundaryPt4_puckBounce = new
Point(Convert.ToInt32(tableBondaryPt2X_puckBounce),
Convert.ToInt32(tableBondaryPt1Y_puckBounce));

            /* Region 1    Region 2
            --|--------. <- Corner 4
              |      /  |
              |    /    | Region 3
              |  /      |
              p1 -------| Region 4
              |  \      |
              |    \    | Region 5
              |      \  |
            --|---------. <- Corner 3
                 Region 7   Region 6
            */

            // Up, middle or down
            if (pt2Y < pt1Y) // Up, Region 1-3
            {
                ptCornerX = boundaryPt4_puckBounce.X;
                ptCornerY = boundaryPt4_puckBounce.Y;
                angle_pt1_ptCorner = Math.Atan(Math.Abs(ptCornerY - pt1Y)/
Math.Abs(ptCornerX - pt1X));
                angle_pt1_Pt2      = Math.Atan(Math.Abs(pt2Y - pt1Y) / Math.Abs(pt2X -
pt1X));

                if(angle_pt1_Pt2 > angle_pt1_ptCorner)  // Region 1
                {
                    pt3Y = tableBondaryPt1Y_puckBounce;
                    // Calculate pt3X
                    pt3X = pt1X + (pt2X - pt1X) * ((pt3Y - pt1Y) / (pt2Y - pt1Y));
                }
                else if (angle_pt1_Pt2 < angle_pt1_ptCorner) // Region 3
                {
                    pt3X = tableBondaryPt2X_puckBounce;
                    // Calculate pt3Y
                    pt3Y = pt1Y + (pt2Y - pt1Y) * ((pt3X - pt1X) / (pt2X - pt1X));
                }
                else // Region 2
                {
                    pt3X = ptCornerX;
                    pt3Y = ptCornerY;
                }
            }
            else if(pt2Y > pt1Y) // Down, Region 5-7
            {
                ptCornerX = boundaryPt3_puckBounce.X;
                ptCornerY = boundaryPt3_puckBounce.Y;
                angle_pt1_ptCorner = Math.Atan(Math.Abs(ptCornerY - pt1Y) /
Math.Abs(ptCornerX - pt1X));
                angle_pt1_Pt2      = Math.Atan(Math.Abs(pt2Y - pt1Y) / Math.Abs(pt2X -
pt1X));

                if (angle_pt1_Pt2 > angle_pt1_ptCorner)  // Region 7
                {
                    pt3Y = tableBondaryPt2Y_puckBounce;
                    // Calculate pt3X
                    pt3X = pt1X + (pt2X - pt1X) * ((pt3Y - pt1Y) / (pt2Y - pt1Y));
                }
                else if (angle_pt1_Pt2 < angle_pt1_ptCorner) // Region 5
                {
                    pt3X = tableBondaryPt2X_puckBounce;
                    // Calculate pt3Y
                    pt3Y = pt1Y + (pt2Y - pt1Y) * ((pt3X - pt1X) / (pt2X - pt1X));
                }
                else // Region 6
                {
                    pt3X = ptCornerX;
                    pt3Y = ptCornerY;
                }
            }
            else  // Middle, Region 4
            {
                pt3X = tableBondaryPt2X_puckBounce;
                pt3Y = pt1Y;
            }

            ptResult.X = Convert.ToInt32(pt3X);
            ptResult.Y = Convert.ToInt32(pt3Y);


            txbtest.AppendText(pt1X + "," + pt1Y + "," + pt2X + "," + pt2Y + "," + pt3X +
"," + pt3Y + "\n");
            return ptResult;
        }

        private Point intersectionMallet(Point pt1, Point pt2)
        {
            Point ptResult = new Point();

            if (pt1.X > pt2.X) // Vector pointing to player
            {
                ptResult.X = -1;
                ptResult.Y = -1;
                return ptResult;
            }
            //calculate line equation
            double pt1X = pt1.X;
            double pt1Y = pt1.Y;
            double pt2X = pt2.X;
            double pt2Y = pt2.Y;
            double pt3X = 0;
            double pt3Y = 0;


            pt3X = tableBondaryPt2X_puckBounce;

            pt1X = pt1X + (pt2X - pt1X) * 2;

            if(pt2X != pt1X)
            {
                pt3Y = pt1Y + (pt2Y - pt1Y) * ((pt3X - pt1X) / (pt2X - pt1X));
            }
            else
            {
                pt3Y = -1;
                pt3X = -1;
                ptResult.X = Convert.ToInt32(pt3X);
                ptResult.Y = Convert.ToInt32(pt3Y);

                return ptResult;
            }

            ptResult.X = Convert.ToInt32(pt3X);
            ptResult.Y = Convert.ToInt32(pt3Y);

            return ptResult;
        }

        private Image<Gray, Byte> cvAndHsvImage(Image<Bgr, Byte> imgFame, int L1, int H1,
int L2, int H2, int L3, int H3, bool H, bool S, bool V, bool I)
        {
            Image<Hsv, Byte> hsvImage = imgFame.Convert<Hsv, Byte>();
            Image<Gray, Byte> ResultImage = new Image<Gray, Byte>(hsvImage.Width,
hsvImage.Height);
            Image<Gray, Byte> ResultImageH = new Image<Gray, Byte>(hsvImage.Width,
hsvImage.Height);
            Image<Gray, Byte> ResultImageS = new Image<Gray, Byte>(hsvImage.Width,
hsvImage.Height);
            Image<Gray, Byte> ResultImageV = new Image<Gray, Byte>(hsvImage.Width,
hsvImage.Height);

            Image<Gray, Byte> img1 = inRangeImage(hsvImage, L1, H1, 0);
            Image<Gray, Byte> img2 = inRangeImage(hsvImage, L2, H2, 1);
            Image<Gray, Byte> img3 = inRangeImage(hsvImage, L3, H3, 2);
            Image<Gray, Byte> img4 = inRangeImage(hsvImage, 0, L1, 0);
            Image<Gray, Byte> img5 = inRangeImage(hsvImage, H1, 180, 0);

            Image<Gray, Byte> img1_re = new Image<Gray, Byte>(imb_1.Width, imb_1.Height);
            Image<Gray, Byte> img2_re = new Image<Gray, Byte>(imb_2.Width, imb_2.Height);
            Image<Gray, Byte> img3_re = new Image<Gray, Byte>(imb_3.Width, imb_3.Height);
            Image<Gray, Byte> img4_re = new Image<Gray, Byte>(imb_4.Width, imb_4.Height);
            Image<Gray, Byte> img5_re = new Image<Gray, Byte>(imb_5.Width, imb_5.Height);
```

```
CvInvoke.Resize(img1, img1_re, new Size(imb_1.Width, imb_1.Height));
CvInvoke.Resize(img2, img2_re, new Size(imb_2.Width, imb_2.Height));
CvInvoke.Resize(img3, img3_re, new Size(imb_3.Width, imb_3.Height));
CvInvoke.Resize(img4, img4_re, new Size(imb_4.Width, imb_4.Height));
CvInvoke.Resize(img5, img5_re, new Size(imb_5.Width, imb_5.Height));

imb_1.Image = img1_re;
imb_2.Image = img2_re;
imb_3.Image = img3_re;
imb_4.Image = img4_re;
imb_5.Image = img5_re;

#region checkBox Color Mode

if (H)
{
    if (I)
    {
        CvInvoke.BitwiseOr(img4, img5, img4);
        ResultImageH = img4;
    }
    else { ResultImageH = img1; }
}

if (S) ResultImageS = img2;
if (V) ResultImageV = img3;

if (H && !S && !V) ResultImage = ResultImageH;
if (!H && S && !V) ResultImage = ResultImageS;
if (!H && !S && V) ResultImage = ResultImageV;

if (H && S && !V)
{
    CvInvoke.BitwiseAnd(ResultImageH, ResultImageS, ResultImageH);
    ResultImage = ResultImageH;
}

if (H && !S && V)
{
    CvInvoke.BitwiseAnd(ResultImageH, ResultImageV, ResultImageH);
    ResultImage = ResultImageH;
}

if (!H && S && V)
{
    CvInvoke.BitwiseAnd(ResultImageS, ResultImageV, ResultImageS);
    ResultImage = ResultImageS;
}

if (H && S && V)
{
    CvInvoke.BitwiseAnd(ResultImageH, ResultImageS, ResultImageH);
    CvInvoke.BitwiseAnd(ResultImageH, ResultImageV, ResultImageH);
    ResultImage = ResultImageH;
}
#endregion
Point pts = new Point(-1,-1);
```

```
            Mat tempMat = new Mat();

            CvInvoke.Erode(ResultImage, ResultImage, new Mat(), pts, 1,
BorderType.Constant,CvInvoke.MorphologyDefaultBorderValue);
            //CvInvoke.Dilate(ResultImage, ResultImage, new Mat(), pts, 1,
BorderType.Constant, CvInvoke.MorphologyDefaultBorderValue);

            return ResultImage;
        }

        private Image<Gray, Byte> inRangeImage(Image<Hsv, Byte> hsvImage, int Lo, int Hi,
int con)
        {
            Image<Gray, Byte> ResultImage = new Image<Gray, Byte>(hsvImage.Width,
hsvImage.Height);
            Image<Gray, Byte> IlowCh = new Image<Gray, Byte>(hsvImage.Width,
hsvImage.Height, new Gray(Lo));
            Image<Gray, Byte> IHiCh = new Image<Gray, Byte>(hsvImage.Width,
hsvImage.Height, new Gray(Hi));
            CvInvoke.InRange(hsvImage[con], IlowCh, IHiCh, ResultImage);
            return ResultImage;
        }

        private byte checkBoundary_intToByte(int _int)
        {
            if (_int < 0)
            {
                _int = 0;
            }
            else if (_int > 255)
            {
                _int = 255;
            }

            if (_int < Convert.ToInt16(nmcMalletLowerLimit.Value))
            {
                _int = Convert.ToInt16(nmcMalletLowerLimit.Value);
            }
            else if(_int > Convert.ToInt16(nmcMalletUpperLimit.Value))
            {
                _int = Convert.ToInt16(nmcMalletUpperLimit.Value);
            }

            return Convert.ToByte(_int);
        }

        private bool puckOutOfBound()
        {
            if(pt1_selected && pt2_selected)
            {
                if(puckX < tableBondaryPt1X_puckBounce)
                {
                    return true;
                }
            }
            return false;
        }
        private void btnStart_Click(object sender, EventArgs e)
        {
            #region if capture is not created, create it now
            if (capture == null)
            {
                try
                {
                    capture = new Capture(Emgu.CV.CvEnum.CaptureType.Any);
                }
                catch (NullReferenceException excpt)
                {
                    MessageBox.Show(excpt.Message);
                }
            }
            #endregion

            if (capture != null)
            {
                if (captureInProgress)
                {  //if camera is getting frames then stop the capture and set button
Text
                    // "Start" for resuming capture
                    btnStart.Text = "Resume"; //
                    Application.Idle -= ProcessFrame;
                }
                else
                {
                    btnStart.Text = "Stop";

                    capture.SetCaptureProperty(CapProp.Fps, frameFps);
                    capture.SetCaptureProperty(CapProp.FrameWidth, frameWidth);
                    capture.SetCaptureProperty(CapProp.FrameHeight, frameHeight);
                    Application.Idle += ProcessFrame;

                }

                captureInProgress = !captureInProgress;
            }
        }

        private void genericTextBoxEventHandler(object sender, EventArgs e)
        {
            TextBox currentTextBox = sender as TextBox;
            short parseResult;
            if (Int16.TryParse((currentTextBox.Text), out parseResult))
            {
                if (parseResult > 255)
                    parseResult = 255;
                if (parseResult <= 0)
                    parseResult = 0;
                currentTextBox.Text = parseResult.ToString();
            }
            else
                currentTextBox.Text = "0";
        }

        private void CamImageBox_MouseClick(object sender, MouseEventArgs e)
        {
            if (!pt1_selected)
            {
                double _eX = e.X;
                double _eY = e.Y;
                tableBondaryPt1X_clicked = Convert.ToInt16(_eX / (CamImageBox.Width  / 
frameWidth ));
                tableBondaryPt1Y_clicked = Convert.ToInt16(_eY / (CamImageBox.Height / 
frameHeight));
                pt1_selected = true;
            }
            else
            {
                if (!pt2_selected)
                {
                    double _eX = e.X;
                    double _eY = e.Y;
                    tableBondaryPt2X_clicked = Convert.ToInt16(_eX / (CamImageBox.Width 
/ frameWidth ));
                    tableBondaryPt2Y_clicked = Convert.ToInt16(_eY / (CamImageBox.Height 
/ frameHeight));
                    pt2_selected = true;

                    if (tableBondaryPt2X_clicked > tableBondaryPt1X_clicked)
                    {
                        tableBondaryPt1X = tableBondaryPt1X_clicked;
                        tableBondaryPt2X = tableBondaryPt2X_clicked;
                    }
                    else
                    {
                        tableBondaryPt1X = tableBondaryPt2X_clicked;
                        tableBondaryPt2X = tableBondaryPt1X_clicked;
                    }

                    if (tableBondaryPt2Y_clicked > tableBondaryPt1Y_clicked)
                    {
                        tableBondaryPt1Y = tableBondaryPt1Y_clicked;
                        tableBondaryPt2Y = tableBondaryPt2Y_clicked;
                    }
                    else
                    {
                        tableBondaryPt1Y = tableBondaryPt2Y_clicked;
                        tableBondaryPt2Y = tableBondaryPt1Y_clicked;
                    }
                    puckRadius = Convert.ToInt16(nmcPuckRadius.Value);
                    tableBondaryPt1X_puckBounce = tableBondaryPt1X + puckRadius;
                    tableBondaryPt1Y_puckBounce = tableBondaryPt1Y + puckRadius;
                    tableBondaryPt2X_puckBounce = tableBondaryPt2X - puckRadius;
                    tableBondaryPt2Y_puckBounce = tableBondaryPt2Y - puckRadius;
                }
                else
                {
                    pt1_selected = false;
                    pt2_selected = false;
                }
            }
        }

        private void mncPuckRadius_ValueChanged(object sender, EventArgs e)
```

```
{
    puckRadius = Convert.ToInt16(nmcPuckRadius.Value);
    tableBondaryPt1X_puckBounce = tableBondaryPt1X + puckRadius;
    tableBondaryPt1Y_puckBounce = tableBondaryPt1Y + puckRadius;
    tableBondaryPt2X_puckBounce = tableBondaryPt2X - puckRadius;
    tableBondaryPt2Y_puckBounce = tableBondaryPt2Y - puckRadius;
}

private void btnReset_Click_1(object sender, EventArgs e)
{
    ckb_EH.Checked = false;
    ckb_ES.Checked = false;
    ckb_EV.Checked = false;

    // H Puck Blue
    nmc_HL.Value = 0;
    nmc_HH.Value = 0;

    // H Mallet green
    nmc_HL_mallet.Value = 0;
    nmc_HH_mallet.Value = 0;

    // S
    nmc_SL.Value = 0;
    nmc_SH.Value = 255;
    //V
    nmc_VL.Value = 0;
    nmc_VH.Value = 255;
}

private void btnpreset_Click(object sender, EventArgs e)
{
    ckb_EH.Checked = true;
    ckb_ES.Checked = true;
    ckb_EV.Checked = true;

    // H Puck Blue
    nmc_HL.Value = 78;
    nmc_HH.Value = 123;

    // H Mallet green
    nmc_HL_mallet.Value = 19;
    nmc_HH_mallet.Value = 102;

    // S
    nmc_SL.Value = 75;
    nmc_SH.Value = 255;
    //V
    nmc_VL.Value = 49;
    nmc_VH.Value = 255;
}

private void ComPortUpdate()
{
    cmbComPort.Items.Clear();
    string[] comPortArray = System.IO.Ports.SerialPort.GetPortNames().ToArray();
    Array.Reverse(comPortArray);
    cmbComPort.Items.AddRange(comPortArray);
    if (cmbComPort.Items.Count != 0)
        cmbComPort.SelectedIndex = 0;
    else
        cmbComPort.Text = "No Ports Found!";
}

private void CameraCapture_FormClosing(object sender, FormClosingEventArgs e)
{
    if (serialPort1.IsOpen){
        serialPort1.Close();
    }
    // Make sure output file is closed
    if (outputFile != null)
        outputFile.Close();
}

private void btnConnect_Click(object sender, EventArgs e)
{
    if (btnConnect.Text == "Connect")
    {
        if (cmbComPort.Items.Count > 0)
        {
            try
            {
                serialPort1.BaudRate = Convert.ToInt32(txtBaudRate.Text);
                serialPort1.PortName = cmbComPort.SelectedItem.ToString();
                serialPort1.Open();
                btnConnect.Text = "Disconnect";
                //timer1.Enabled = true;
                //lblIncomingDataRate.Visible = true;
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message);
            }
        }
    }
    else
    {
        try
        {
            serialTransmit(0, 0);
            serialPort1.Close();
            btnConnect.Text = "Connect";
            //timer1.Enabled = false;
            //lblIncomingDataRate.Visible = false;
        }
        catch (Exception ex)
        {
            MessageBox.Show(ex.Message);
        }
    }
}

private void ckbByte1_CheckedChanged(object sender, EventArgs e)
{
    if (ckbByte1.Checked == true)
        txtByte1.Enabled = true;
    else
    {
        txtByte1.Clear();
        txtByte1.Enabled = false;
    }
}

private void ckbByte2_CheckedChanged(object sender, EventArgs e)
{
    if (ckbByte2.Checked == true)
        txtByte2.Enabled = true;
    else
    {
        txtByte2.Clear();
        txtByte2.Enabled = false;
    }
}

private void ckbByte3_CheckedChanged(object sender, EventArgs e)
{
    if (ckbByte3.Checked == true)
        txtByte3.Enabled = true;
    else
    {
        txtByte3.Clear();
        txtByte3.Enabled = false;
    }
}

private void ckbByte4_CheckedChanged(object sender, EventArgs e)
{
    if (ckbByte4.Checked == true)
        txtByte4.Enabled = true;
    else
    {
        txtByte4.Clear();
        txtByte4.Enabled = false;
    }
}

private void btnSerialSend_Click(object sender, EventArgs e)
{
    //serialTransmit_flag = true;
    serialTransmit();
}

private void serialTransmit()
{
    byte[] TxBytes = new Byte[5];

    try
    {
        if (serialPort1.IsOpen)
        {
            if (ckbByte1.Checked && (txtByte1.Text != ""))
            {
                TxBytes[0] = Convert.ToByte(txtByte1.Text);
                serialPort1.Write(TxBytes, 0, 1);
            }
            if (ckbByte2.Checked && (txtByte2.Text != ""))
            {
                TxBytes[1] = Convert.ToByte(txtByte2.Text);
                serialPort1.Write(TxBytes, 1, 1);
            }
            if (ckbByte3.Checked && (txtByte3.Text != ""))
            {
                TxBytes[2] = Convert.ToByte(txtByte3.Text);
                serialPort1.Write(TxBytes, 2, 1);
            }
            if (ckbByte4.Checked && (txtByte4.Text != ""))
            {
                TxBytes[3] = Convert.ToByte(txtByte4.Text);
                serialPort1.Write(TxBytes, 3, 1);
            }
            if (ckbByte4.Checked && (txtByte4.Text != ""))
            {
                TxBytes[4] = Convert.ToByte(txtByte5.Text);
                serialPort1.Write(TxBytes, 4, 1);
            }

        }
    }
    catch (Exception Ex)
    {
        MessageBox.Show(Ex.Message);
        ckbTransmit.Checked = false;
    }
}

private void serialTransmit(Byte byte1)
{
    byte[] TxBytes = new Byte[1];
    try
    {
        TxBytes[0] = byte1;
        serialPort1.Write(TxBytes, 0, 1);
     }
    catch (Exception Ex)
    {
        MessageBox.Show(Ex.Message);
        ckbTransmit.Checked = false;
    }
}

private void serialTransmit(Byte byte1, Byte byte2)
{
    byte[] TxBytes = new Byte[3];
    try
    {
        TxBytes[0] = 255;
        serialPort1.Write(TxBytes, 0, 1);

        TxBytes[1] = byte1;
        serialPort1.Write(TxBytes, 1, 1);

        TxBytes[2] = byte2;
```

```
                serialPort1.Write(TxBytes, 2, 1);
            }
            catch (Exception Ex)
            {
                MessageBox.Show(Ex.Message);
                ckbTransmit.Checked = false;
            }
        }

        private void serialTransmit(Byte byte1, Byte byte2, Byte byte3, Byte byte4, Byte
byte5)
        {
            byte[] TxBytes = new Byte[5];
            int MyInt = byte1;
            byte[] b = BitConverter.GetBytes(MyInt);
            serialPort1.Write(b, 0, 4);
            try
            {
                if (serialPort1.IsOpen)
                {

                    if (ckbByte1.Checked && (txtByte1.Text != ""))
                    {
                        TxBytes[0] = byte1;
                        serialPort1.Write(b, 0, 4);
                    }
                    if (ckbByte2.Checked && (txtByte2.Text != ""))
                    {
                        TxBytes[1] = byte2;
                        serialPort1.Write(TxBytes, 1, 1);
                    }
                    if (ckbByte3.Checked && (txtByte3.Text != ""))
                    {
                        TxBytes[2] = byte3;
                        serialPort1.Write(TxBytes, 2, 1);
                    }
                    if (ckbByte4.Checked && (txtByte4.Text != ""))
                    {
                        TxBytes[3] = byte4;
                        serialPort1.Write(TxBytes, 3, 1);
                    }
                    if (ckbByte4.Checked && (txtByte4.Text != ""))
                    {
                        TxBytes[4] = byte5;
                        serialPort1.Write(TxBytes, 4, 1);
                    }

                }
            }
            catch (Exception Ex)
            {
                MessageBox.Show(Ex.Message);
                ckbTransmit.Checked = false;
            }
        }

        private void btnSerialPreset_Click(object sender, EventArgs e)
        {
            ckbByte1.Checked = true;
            txtByte1.Text = "255";
            ckbByte2.Checked = true;
            txtByte2.Text = "0";
            ckbByte3.Checked = true;
            txtByte3.Text = "0";
            /*
            ckbByte4.Checked = true;
            txtByte4.Text = "120";
            ckbByte5.Checked = true;
            txtByte5.Text = "0";
            */
        }

        private void btnSerialReset_Click(object sender, EventArgs e)
        {
            ckbByte1.Checked = false;
            txtByte1.Text = "";
            ckbByte2.Checked = false;
            txtByte2.Text = "";
            ckbByte3.Checked = false;
            txtByte3.Text = "";
            ckbByte4.Checked = false;
            txtByte4.Text = "";
            ckbByte5.Checked = false;
            txtByte5.Text = "";
        }

        private void ckbByte5_CheckedChanged(object sender, EventArgs e)
        {
            if (ckbByte5.Checked == true)
                txtByte5.Enabled = true;
            else
            {
                txtByte5.Clear();
                txtByte5.Enabled = false;
            }
        }

        private void serialPort1_DataReceived(object sender,
System.IO.Ports.SerialDataReceivedEventArgs e)
        {
            while (serialPort1.IsOpen && serialPort1.BytesToRead != 0)
            {
                int currentByte = serialPort1.ReadByte();
                currentByte -= 48;
                if (ckbShowResponse.Checked)
                    this.BeginInvoke(new EventHandler(delegate
                    {
                        txtRawSerial.AppendText(currentByte.ToString() + ", ");
                    }));

            }
        }

        private void ckbSendInt_CheckedChanged(object sender, EventArgs e)
        {
            if(ckbSendInt.Checked == true)
```

```
                ckbSendByte.Checked = false;
            else
                ckbSendByte.Checked = true;
        }

        private void ckbSendByte_CheckedChanged(object sender, EventArgs e)
        {
            if (ckbSendByte.Checked == true)
                ckbSendInt.Checked = false;
            else
                ckbSendInt.Checked = true;
        }

        private void btnClearText_Click(object sender, EventArgs e)
        {
            txtRawSerial.Text = "";
        }

        private void btnCalcualteIntersection_Click(object sender, EventArgs e)
        {
            Image<Bgr, Byte> img_test = ImageFrame;

            Image<Bgr, Byte> img_test_re = new Image<Bgr, Byte>(camTest.Width,
camTest.Height);

            PointF center1 = new PointF(Convert.ToSingle(nmcPt1X_test.Value),
Convert.ToSingle(nmcPt1Y_test.Value));
            PointF center2 = new PointF(Convert.ToSingle(nmcPt2X_test.Value),
Convert.ToSingle(nmcPt2Y_test.Value));
            CircleF circ1 = new CircleF(center1, 2);
            CircleF circ2 = new CircleF(center2, 2);



            // Intersection point on boundary
            Point boundaryPt = intersectionBoundaryPoint(
                new Point(Convert.ToInt32(nmcPt1X_test.Value),
Convert.ToInt32(nmcPt1Y_test.Value)),
                new Point(Convert.ToInt32(nmcPt2X_test.Value),
Convert.ToInt32(nmcPt2Y_test.Value))
                );

            CircleF circ3 = new CircleF(boundaryPt, 2);

            // Calculate collision path
            if (boundaryPt.X != -1 && boundaryPt.Y != -1)
            {
                txtBoundaryX_test.Text = boundaryPt.X.ToString();
                txtBoundaryY_test.Text = boundaryPt.Y.ToString();
                CvInvoke.Line(img_test, new Point(Convert.ToInt32(nmcPt1X_test.Value),
Convert.ToInt32(nmcPt1Y_test.Value)), boundaryPt, new MCvScalar(255, 0, 255, 0));
            }

            img_test.Draw(circ1, new Bgr(Color.Blue), 2);
            img_test.Draw(circ2, new Bgr(Color.Green), 2);
            img_test.Draw(circ3, new Bgr(Color.Red), 2);

            // Boundary corner points
            Point boundaryPt1 = new Point(Convert.ToInt32(tableBondaryPt1X),
Convert.ToInt32(tableBondaryPt1Y));
            Point boundaryPt2 = new Point(Convert.ToInt32(tableBondaryPt1X),
Convert.ToInt32(tableBondaryPt2Y));
            Point boundaryPt3 = new Point(Convert.ToInt32(tableBondaryPt2X),
Convert.ToInt32(tableBondaryPt2Y));
            Point boundaryPt4 = new Point(Convert.ToInt32(tableBondaryPt2X),
Convert.ToInt32(tableBondaryPt1Y));

            // Draw boundary lines on quarry image
            CvInvoke.Line(img_test, boundaryPt1, boundaryPt2, new MCvScalar(0, 128, 255,
0));
            CvInvoke.Line(img_test, boundaryPt2, boundaryPt3, new MCvScalar(0, 128, 255,
0));
            CvInvoke.Line(img_test, boundaryPt3, boundaryPt4, new MCvScalar(0, 128, 255,
0));
            CvInvoke.Line(img_test, boundaryPt4, boundaryPt1, new MCvScalar(0, 128, 255,
0));

            // Puck bounce boundary points
            Point boundaryPt1_puckBounce = new
Point(Convert.ToInt32(tableBondaryPt1X_puckBounce),
Convert.ToInt32(tableBondaryPt1Y_puckBounce));
            Point boundaryPt2_puckBounce = new
Point(Convert.ToInt32(tableBondaryPt1X_puckBounce),
Convert.ToInt32(tableBondaryPt2Y_puckBounce));
            Point boundaryPt3_puckBounce = new
Point(Convert.ToInt32(tableBondaryPt2X_puckBounce),
Convert.ToInt32(tableBondaryPt2Y_puckBounce));
            Point boundaryPt4_puckBounce = new
Point(Convert.ToInt32(tableBondaryPt2X_puckBounce),
Convert.ToInt32(tableBondaryPt1Y_puckBounce));

            // Draw puck bounce boundary lines on quarry image
            CvInvoke.Line(img_test, boundaryPt1_puckBounce, boundaryPt2_puckBounce, new
MCvScalar(0, 128, 255, 0));
            CvInvoke.Line(img_test, boundaryPt2_puckBounce, boundaryPt3_puckBounce, new
MCvScalar(0, 128, 255, 0));
            CvInvoke.Line(img_test, boundaryPt3_puckBounce, boundaryPt4_puckBounce, new
MCvScalar(0, 128, 255, 0));
            CvInvoke.Line(img_test, boundaryPt4_puckBounce, boundaryPt1_puckBounce, new
MCvScalar(0, 128, 255, 0));


            CvInvoke.Resize(img_test, img_test_re, new Size(camTest.Width,
camTest.Height));
            camTest.Image = img_test_re;

        }

        private void timer2_Tick(object sender, EventArgs e)
        {
            returnCount++;
        }

        private void ckbPacketContent_CheckedChanged(object sender, EventArgs e)
        {
            if(ckbPacketContent.Checked == true)
```

```
    {
        ckbPacketContent.Text = "Puck and mallet";
    }
    else
    {
        ckbPacketContent.Text = "Just Puck";
    }
}

private void btnUpdateComPort_Click(object sender, EventArgs e)
{
    ComPortUpdate();
}

private void btnReset_CannyEdge_Click(object sender, EventArgs e)
{
    nmc_cannyThreshold.Value = 180;
    nmc_circleAccumulatorThreshold.Value = 30;
    nmcCircleMinRadius.Value = 5;
    nmcCircleMaxRadius.Value = 15;
}

private void btnReset_GaussFilter_Click(object sender, EventArgs e)
{
    txt_kernelWidth.Text = "3";
    txt_kernelHeight.Text = "3";
    txt_sigma1.Text = "34.3";
    txt_sigma2.Text = "45.3";
}

private void chkSaveData_CheckedChanged(object sender, EventArgs e)
{
    try
    {
        // If the field is checked, write to output file
        if (ckbSaveData.Checked == true)
        {
            outputFile = new StreamWriter(txtFileName.Text);
        }
        else {
            outputFile.Close();
            initializeFile = false;
        }
    }
    catch (System.ArgumentException err)
    {
        // Catch exception and display error
        ckbSaveData.Checked = false;
        ckbRecordBoth.Checked = false;
        ckbRecordVideo.Checked = false;
        ckbRecordBoth.Text = "Record both";
        MessageBox.Show("Filename cannot be empty.");
    }
    catch (System.NullReferenceException)
    {
        // Error handling
    }
    catch (System.UnauthorizedAccessException)
    {
        MessageBox.Show("File path invalid.");
    }
}

private void btnSelectFileName_Click(object sender, EventArgs e)
{
    SaveFileDialog myDialogBox = new SaveFileDialog();
    myDialogBox.AutoUpgradeEnabled = false;
    myDialogBox.InitialDirectory = pathName;
    myDialogBox.Filter = "Text files (*.csv)|*.csv|All files (*.*)|*.*";
    myDialogBox.DefaultExt = "csv";
    myDialogBox.ShowDialog();
    txtFileName.Text = myDialogBox.FileName.ToString();
}

private void ckbRecordBoth_CheckedChanged(object sender, EventArgs e)
{
    if (ckbRecordBoth.Checked)
    {
        ckbRecordVideo.Checked = true;
        ckbSaveData.Checked = true;
        ckbRecordBoth.Text = "Press to Stop";
    }
    else
    {
        ckbRecordVideo.Checked = false;
        ckbSaveData.Checked = false;
        ckbRecordBoth.Text = "Record both";
    }
}

private void btnSetRanges_Click(object sender, EventArgs e)
{
    malletDestinationRange = Convert.ToInt16(nmc_malletDestinationRange.Value);
    malletCurrentRange = Convert.ToInt16(nmc_malletCurrentRange.Value);
    puckJitterRange = Convert.ToDouble(txt_puckJitterRange.Text);
}

private void button1_Click(object sender, EventArgs e)
{
    int _int = 340;
    lblTest.Text = checkBoundary_intToByte(_int).ToString();
}

private void nmcTableBoundary1X_ValueChanged(object sender, EventArgs e)
{
    tableBondaryPt1X = Convert.ToInt16(nmcTableBoundary1X.Value);
    puckRadius = Convert.ToInt16(nmcPuckRadius.Value);
    tableBondaryPt1X_puckBounce = tableBondaryPt1X + puckRadius;
}

private void nmcTableBoundary1Y_ValueChanged(object sender, EventArgs e)
{
    tableBondaryPt1Y = Convert.ToInt16(nmcTableBoundary1Y.Value);
    puckRadius = Convert.ToInt16(nmcPuckRadius.Value);
    tableBondaryPt1Y_puckBounce = tableBondaryPt1Y + puckRadius;
}
```

```
        private void nmcTableBoundary2X_ValueChanged(object sender, EventArgs e)
        {
            tableBondaryPt2X = Convert.ToInt16(nmcTableBoundary2X.Value);
            puckRadius = Convert.ToInt16(nmcPuckRadius.Value);
            tableBondaryPt2X_puckBounce = tableBondaryPt2X - puckRadius;
        }

        private void nmcTableBoundary2Y_ValueChanged(object sender, EventArgs e)
        {
            tableBondaryPt2Y = Convert.ToInt16(nmcTableBoundary2Y.Value);
            puckRadius = Convert.ToInt16(nmcPuckRadius.Value);
            tableBondaryPt2Y_puckBounce = tableBondaryPt2Y - puckRadius;
        }

        private void img_HSVwheel_MouseClick(object sender, MouseEventArgs e)
        {
            Image<Hsv, Byte> imgHsv = ImageHSVwheel.Convert<Hsv, Byte>();
            Hsv hsv = imgHsv[e.Y, e.X];
            if (!ib3C)
            {
                ib3C_HV = Convert.ToInt32(hsv.Hue);
                ib3C_Lable_X = img_HSVwheel.Location.X + e.X;
                ib3C_Lable_Y = img_HSVwheel.Location.Y + e.Y;
                label_L.Location = new Point(ib3C_Lable_X, ib3C_Lable_Y);
                label_H.Hide();
            }
            if (ib3C)
            {
                label_H.Show();
                if (ib3C_HV < hsv.Hue)
                {
                    nmc_HL.Value = ib3C_HV;
                    nmc_HH.Value = Convert.ToInt32(hsv.Hue);
                    label_H.Location = new Point(img_HSVwheel.Location.X + e.X,
img_HSVwheel.Location.Y + e.Y);
                }
                else
                {
                    nmc_HH.Value = ib3C_HV;
                    nmc_HL.Value = Convert.ToInt32(hsv.Hue);
                    ib3C_HV = 0;
                    label_H.Location = new Point(ib3C_Lable_X, ib3C_Lable_Y);
                    label_L.Location = new Point(img_HSVwheel.Location.X + e.X,
img_HSVwheel.Location.Y + e.Y);
                }
            }
            ib3C = !ib3C;
        }

        private void ReleaseData()
        {
            if (capture != null)
                capture.Dispose();
        }

        private void timer1_Tick(object sender, EventArgs e)
        {
            if (capture != null)
            {
                try
                {
                    lblWidth.Text = 
capture.GetCaptureProperty(CapProp.FrameWidth).ToString();
                    lblHeight.Text = 
capture.GetCaptureProperty(CapProp.FrameHeight).ToString();
                    lblfps.Text = capture.GetCaptureProperty(CapProp.Fps).ToString();
                }
                catch (NullReferenceException excpt)
                {
                    MessageBox.Show(excpt.Message);
                }
            }

        }

    }
}
```