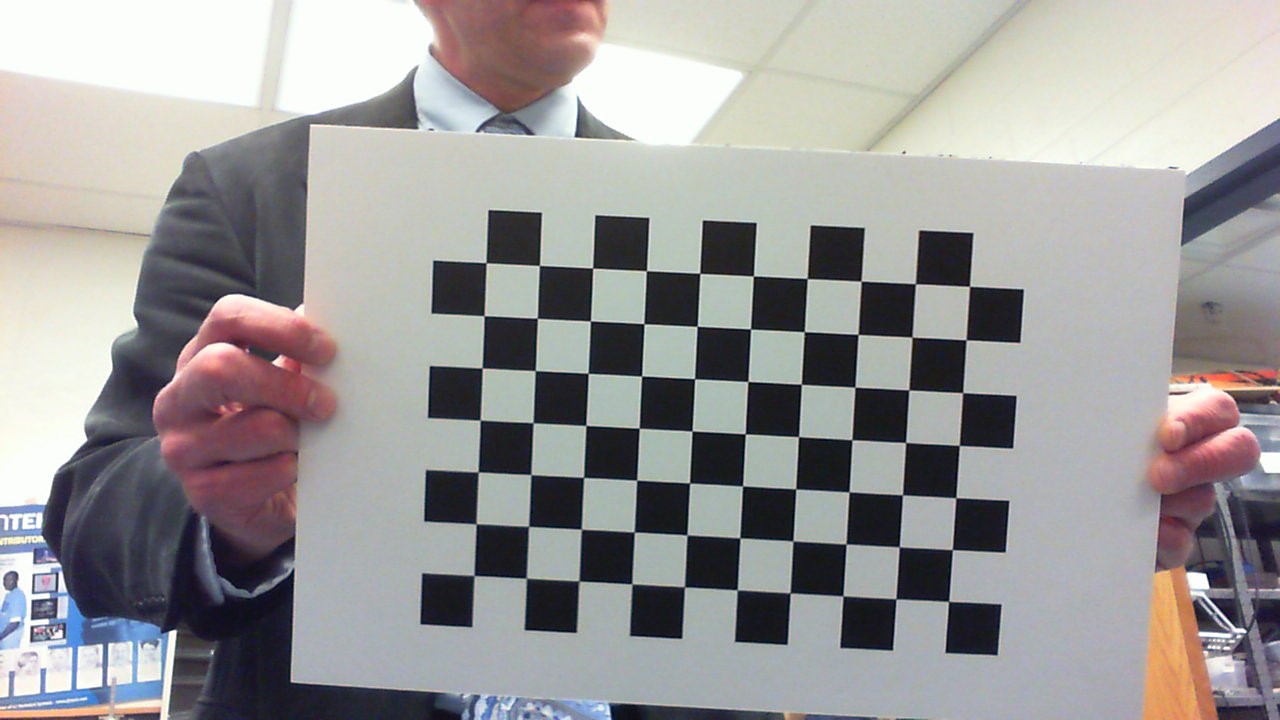
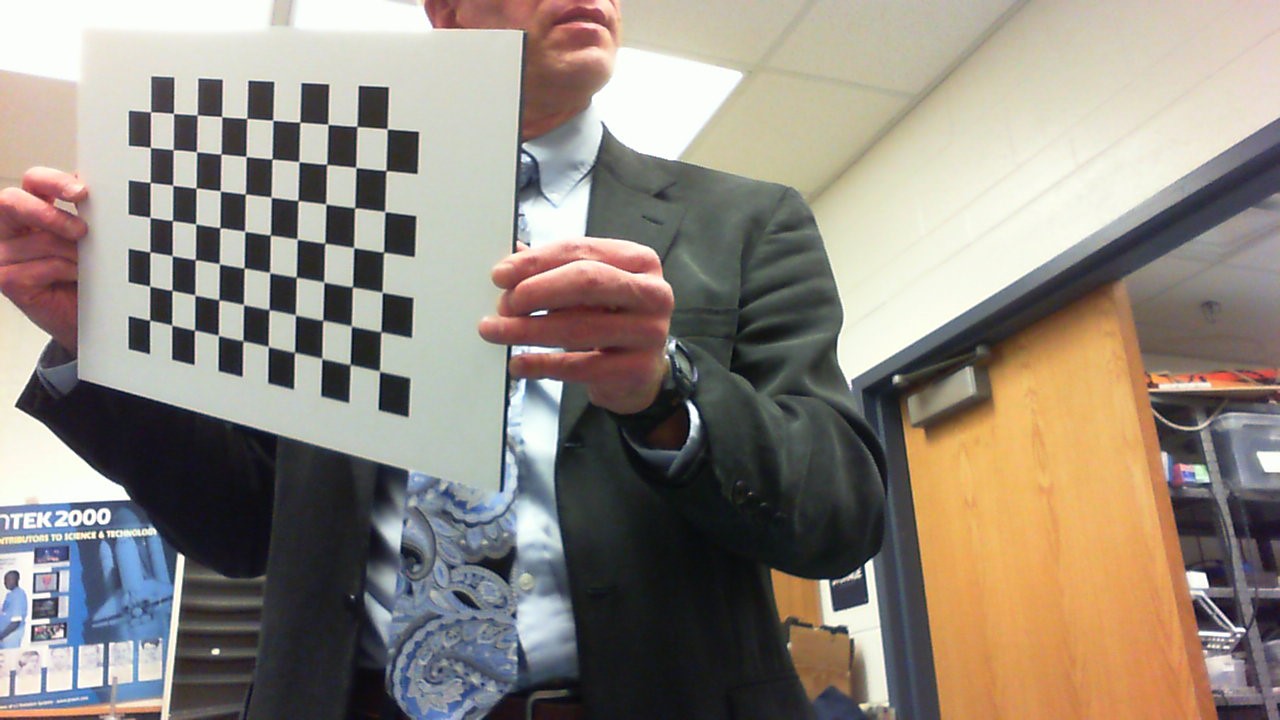
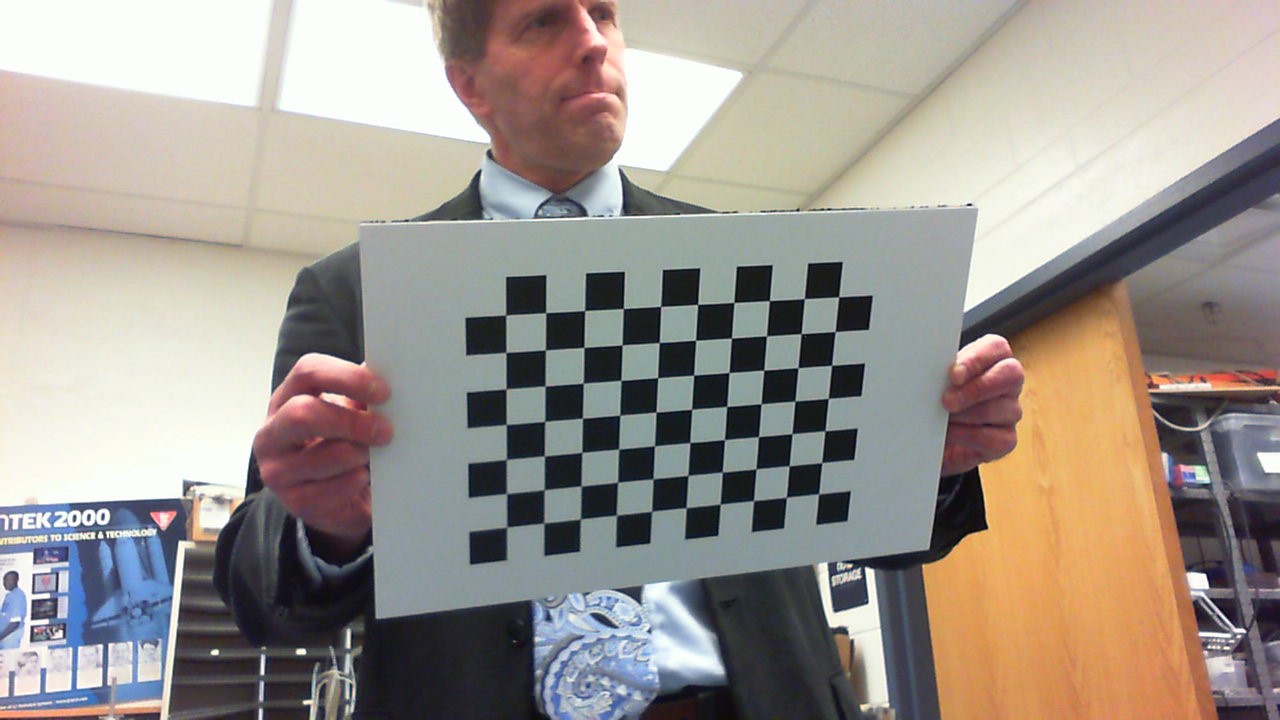
Camera Calibration

* Print a chessboard pattern and glue it on to a rigid cardboard.
* Take several images of the pattern (at least 25-30) changing its position and orientation

Example:





This code works for a 7x10 squares pattern. (We neglect the outer ring because it needs to see the contrast between black and white). If you have a different pattern, substitute the 7s and the 10s with your values.

Check the correct calibration by looking at the output image. It should look something like this:



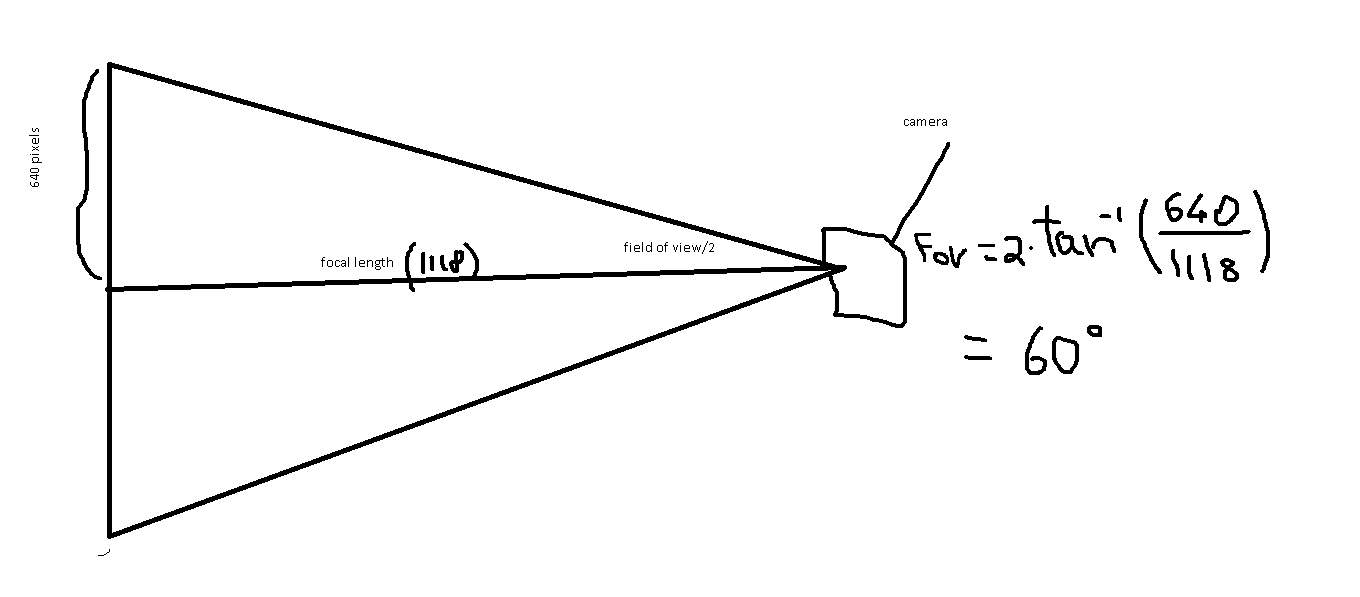
At the end of the process, something like this will come out:



The values that we need are the ones in bold:

* 1.124x103 and 1.113x103 are two focal lengths. We will consider the mean value (1118)
* 6.42x102 and 3.43x102 are the optical centers of the camera.

How to find field of view



How to find pixels per degree

We know we have a resolution of 640 (horizontally) pixels and we know that the horizontal field of view is 60°. Therefore we have 640pixel/60degrees = 10.67pixels/degree

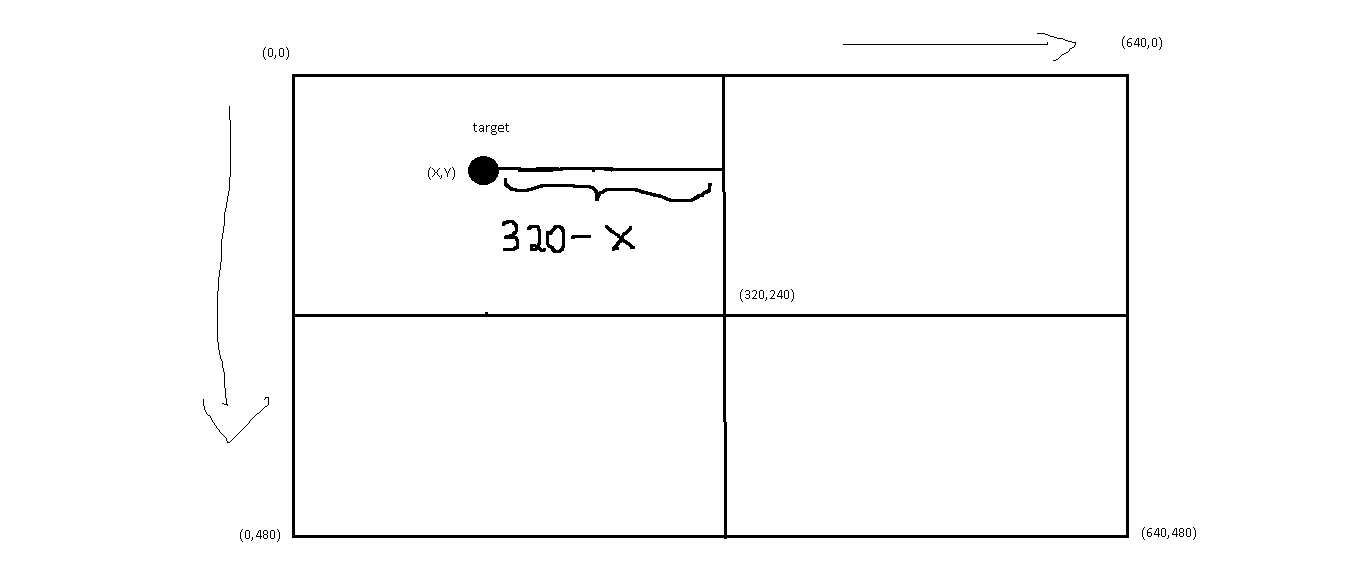
We know we have a resolution of 480 (vertically) pixels and we know that the vertical field of view is 34°. Therefore we have a 480/34=14.12 pixels/degree

Boiler detection:

The boiler detection software has to find two main things: the delta angle between the center of the camera and the center of the tape, and the distance between the camera and the target.

Angle:

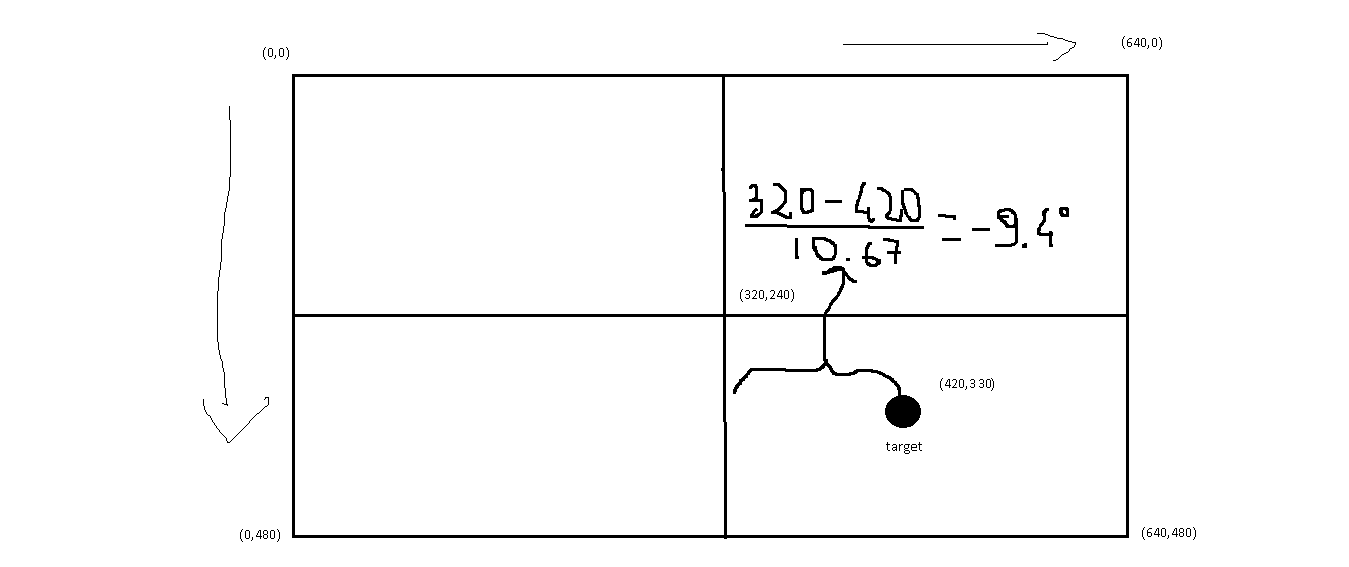
In order to find the delta angle we need to know two things: the field of view and the resolution of the camera. First off, we find the delta pixels between the center of the camera and the target:



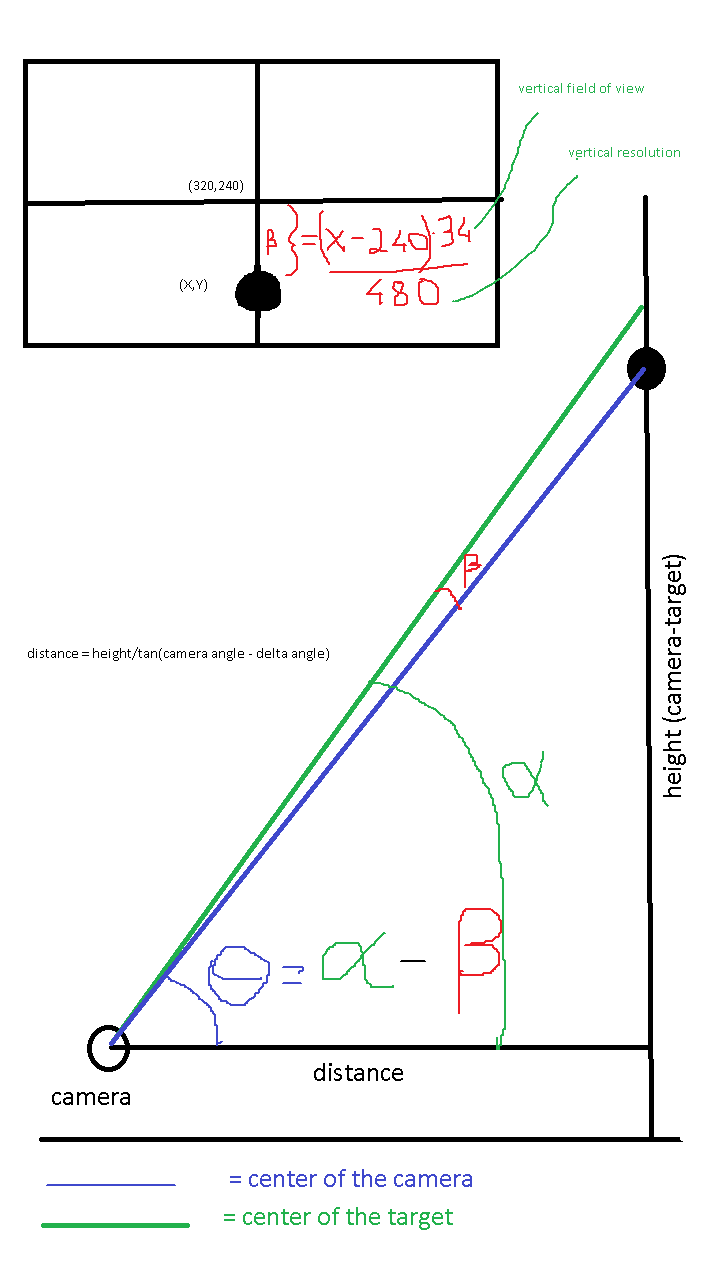
We can do that by doing 320 (the X-coordinate of the center of the camera) – the X-coordinate of the target.

After we find how many pixels we have, we can find how many degrees by dividing that number by 10.67 (the pixels/degree)

Example:



Distance:



Gear Recognition

For the gear recognition, we used the same principle of the boiler angle. We need to find the angle between the center of the camera and the center of the gear stick. In my case, I considered the midpoint in between the two vertical tape stripes. After finding how many pixels there are, we divide it by the angle/degree (horizontal), and we find the rotation of the robot.

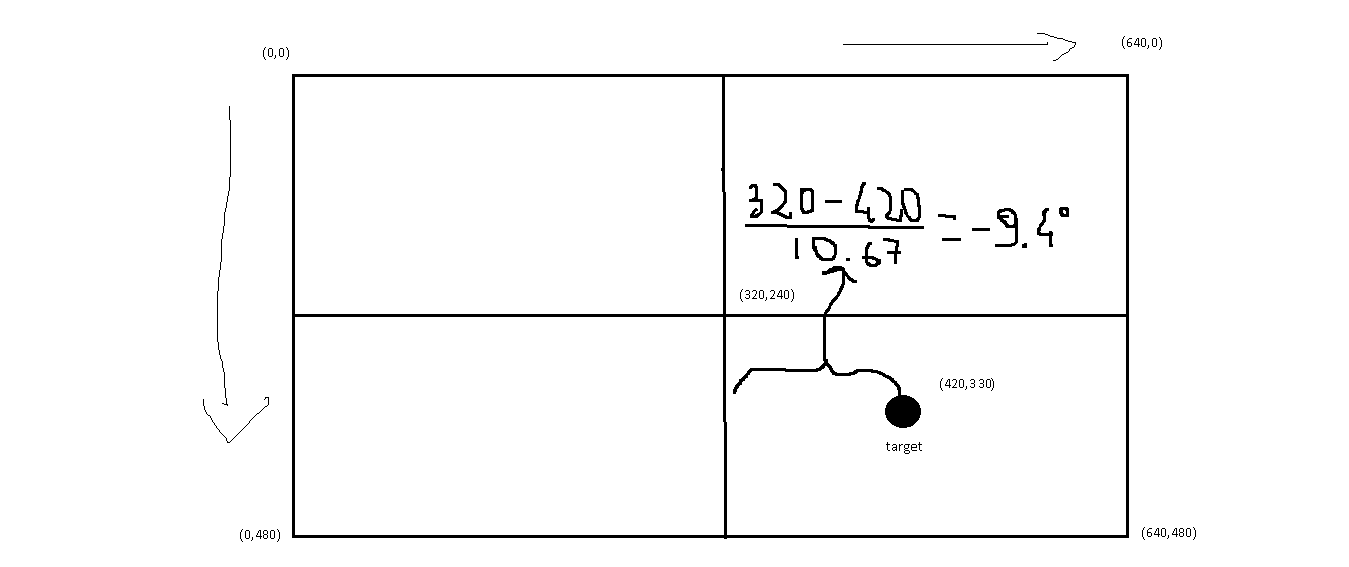


Image Processing

In my final version of the code, the only image processing that I do is the conversion from RBG (red, blue, green) to HSV (hue, saturation, value). HSV is more precise and less affected by external light.

After the conversion, we use a threshold to create a binary map of everything within that range. This will find many little blobs of the specified color.

Now we can process the image even more with morphological transformations such as an “opening” in order to eliminate some of the noise. I decided to avoid this and use another approach.

Considering that in a field, the biggest shapes and contours we can find are the two pieces of tape. This means we can just sort out all the contours and keep only the biggest two. (Two pieces of tape)

If wanted to, we could do some math and analyze all the contours in order to focus our attention only on the right ones. We could, for instance, compare height and width of these contours and decide to keep only those that have height > 2\*width. (The strips of gear tape are positioned vertically and therefore the height is bigger than the width). With this simple algorithm, we could neglect most of the little blobs or noise we have found.

Boiler Recognition Code

1. **import** cv2
2. **import** numpy as np
3. **from** networktables **import** NetworkTables
4. #import argparse
5. **import** math
6. ########################################################
8. #IP address of the RoboRio
9. NetworkTables.setIPAddress("10.6.14.2")
11. #NetworkTables initialization
12. NetworkTables.setClientMode()
13. NetworkTables.initialize()
14. #retrieval of the correct table. The Roborio needs to use the same table
15. table = NetworkTables.getTable("shooterCamera")
17. #ap = argparse.ArgumentParser()
18. #ap.add\_argument("-he", "--height",type=float, required=True,
19. #   help="GlobalHeight")
20. #ap.add\_argument("-a", "--angle", type=float, required=True,
21. #    help="Angle of the camera")
22. #args = vars(ap.parse\_args())

25. #HEIGHT\_ROBOT\_TOPTAPE = args["height"]
26. #CAMERA\_ANGLE = args["angle"]
28. #height from the camera and the top edge of the boiler's tape
29. HEIGHT\_ROBOT\_TOPTAPE =20.0
30. #angle the camera is mounted at
31. CAMERA\_ANGLE = 30.0


35. **def** min(val1, val2):
36. **return** val1 **if** val1<val2 **else** val2
38. **def** deltaAngle(yVal):
39. **return** float((yVal -240)\*34/480)
41. **def** retrieveYTopContour(contours):
42. #we get the two contours and we decide which one is the top one.
43. x1,y1,w1,h1 = cv2.boundingRect(contours[0])
44. x2,y2,w2,h2 = cv2.boundingRect(contours[1])
45. **return** float(min(y1,y2))
47. **def** distanceFromCamera(height, deltAngle):
48. distance = height/math.tan(math.radians(CAMERA\_ANGLE) - math.radians(deltAngle))
49. **return** distance
51. **def** distanceFromCenter(midPoint):
52. #we calculate the distance between the center of the camera and the
53. #midpoint of the contour
54. #in this case we use 320 because we are using a 640\*480 resolution
55. #we could also use 320 - midPoint if we wanted to have positive angles on the right
56. d = midPoint - 320
57. **return** d
59. **def** midMidPoint(contour1,contour2):
60. #standard opencv midpoint detection
61. M1 = cv2.moments(contour1)
62. M2 = cv2.moments(contour2)
63. #to avoid a division by 0 we check if those moments are null or not
64. **if** M1['m00'] != 0 **and** M2['m00'] != 0:
65. cx1 = int(M1['m10']/M1['m00'])
66. cx2 = int(M2['m10']/M2['m00'])
67. cy1 = int(M1['m01']/M1['m00'])
68. cy2 = int(M2['m01']/M2['m00'])
69. **else**:
70. cx1 = 0
71. cx2 = 0
72. cy1 = 0
73. cy2 = 0
75. midX = (cx1 +cx2)/2
76. midY = (cy1 +cy2)/2
77. mid = (midX, midY)
78. **return** mid

81. **def** printXY(contour, img):
82. #debugging function that prints on the image values such as position of
83. #edges and verious heights.
84. x,y,w,h = cv2.boundingRect(contour)
86. writeText(str(y),(x-10, y),img)
87. writeText(str(h),(x-10, y+ h/2),img)
88. writeText(str(w), (x + w/2, y), img)



93. **def** pixelToDegree(dPix):
94. #after camera calibration, we found out that the focal length is 1118 pixels
95. #the field of view was calculated to be 60 degrees
96. #we have 640/60 pixels per degree
98. **return** dPix / 10.67
100. **def** drawBoundingBox(contour, img):
101. #it draws the bounding box around the contours we find
102. x,y,w,h = cv2.boundingRect(contour)
104. p1 = (x,y)
105. p2 = ((x + w),(y+h))
107. cv2.rectangle(img, p1,p2, (255,0,0))

110. **def** writeText(text, orig, img, color = (255,255,255)):
111. #generic function that writes something on screen
112. cv2.putText(img, text, orig, cv2.FONT\_HERSHEY\_SIMPLEX, 0.5,color)

115. ##################################################################
116. #hsv range of detection found with GRIP
117. lower\_red = np.array([0, 205, 90])
118. upper\_red = np.array([180, 255, 255])
120. #it gets the video stream from the main camera (the number 0)
121. cap = cv2.VideoCapture(0)
122. #set the resolution
123. cap.set(3, 640)##widh
124. cap.set(4, 480)##height
126. #######################################################################
127. **while** True:
128. #read a single frame
129. \_, input = cap.read()
130. # RGB TO HSV
131. out = cv2.cvtColor(input, cv2.COLOR\_BGR2HSV)
132. # threshold HSV
133. mask = cv2.inRange(out, lower\_red, upper\_red)
134. contours, hierarchy = cv2.findContours(
135. mask, cv2.RETR\_TREE, cv2. CHAIN\_APPROX\_SIMPLE)
136. #we sort our contours based on their area (how big they are) and we just
137. #keep the first two (we have to tapes on the boiler and they are gonna be
138. #the biggest shapes on screen
139. contours = sorted(contours, key=cv2.contourArea, reverse=True)[:2]
141. # focal length = 1118
143. #we check if we have both the contours (1 is useless)
144. **if** len(contours) == 2:
146. #drawBoundingBox(contours[0], input)
147. #drawBoundingBox(contours[1], input)
149. #cv2.drawContours(input, [contours[0]], -1, (0, 255, 0), 2)
150. #cv2.drawContours(input, [contours[1]], -1, (0, 255, 0), 2)
152. #we find mid point of the two contours
153. mid = midMidPoint(contours[0], contours[1])
154. #we find the angle between the camera and the center of the target
155. angle = pixelToDegree(distanceFromCenter(mid[0]))
156. #we find the distance between camera and boiler
157. distance = distanceFromCamera(HEIGHT\_ROBOT\_TOPTAPE, deltaAngle(retrieveYTopContour(contours)))
159. #writeText("Angle: %f" % angle,(50,200), input)
160. #writeText("Distance: %f" % distance,(50,300), input)
162. #printXY(contours[0], input)
163. #printXY(contours[1], input)
165. #we write on the table those values so that roborio can access them
166. table.putNumber("angle", angle)
167. table.putBoolean("targetFound", True)
168. table.putNumber("distance", distance)



173. **else**:
174. #if not both the contours are found, we ignore that
175. #writeText("Nothing found",(50,300), input, (0,0,255))
176. table.putBoolean("targetFound", False)
177. table.putNumber("angle",999)
178. table.putNumber("distance",999)

181. #cv2.imshow('mask', input)
183. k = cv2.waitKey(5) & 0xFF
184. **if** k == 27:
185. **break**
187. cv2.destroyAllWindows()