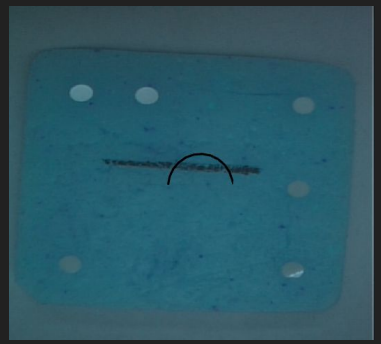
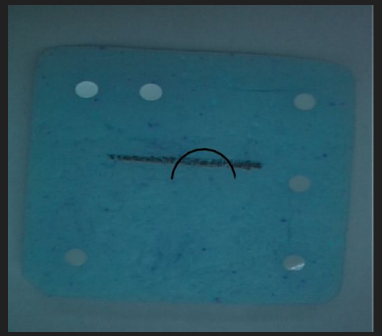
Particle Filter Auto-Initialization

Goals:

- Identify 3D location and orientation of the needle prior to running particle filter
  - The pose should serve as an accurate starting point for the particle filter
  - Should reduce the time to convergence, as well as allow for the particle filter to be re-initialized quickly if the needle is lost
- Identify pose quickly and consistently given real images from the endoscope cameras



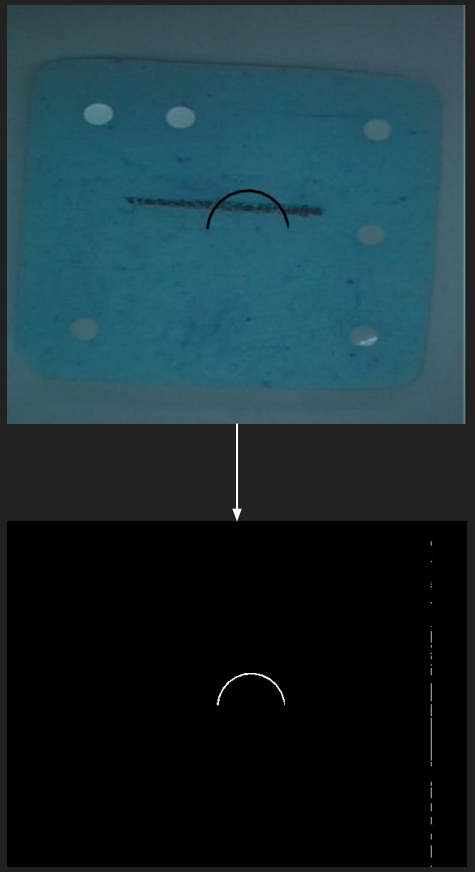


Method Overview

1. Needle Segmentation
2. Template Matching
3. 3D Location Computation

1. Needle Segmentation

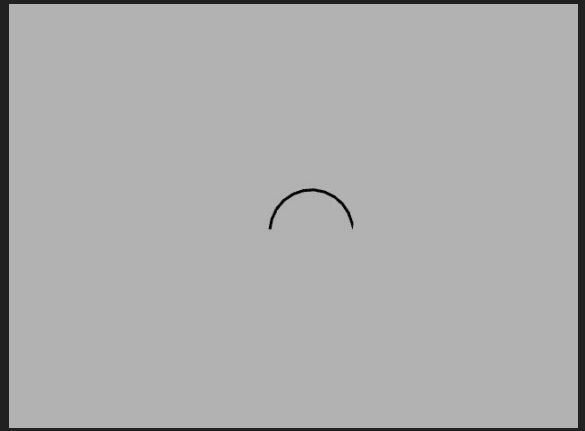
- Before locating the needle in the image, we segment the needle from the background
  - Easier to identify later during template matching
- Filters out non-gray pixels in HSV color space
  - This works well for the current, artificially created dataset
- Problems/Assumptions:
  - Assumes the background does not share color attributes with needle
  - Not robust for different lighting conditions (assumes all needles are always the same color throughout
- Other considered solutions:
  - Edge detection (generally identifies too many other needle-like structures
  - Thin feature extraction (“vesselness” in the project); similar issues to edge detection



2. Template Matching

- Goal:
  - Find the location of the needle in image space for the left and right images
  - Find rotation of the needle around z-axis
- 2a - Template Creation:
  - Take an existing camera image with a known needle location and crop to contain only the needle
  - Segment as in step 1.
  - Rotate to initial 0° position
  - Note: assumes the needle used to generate the template is very similar to live needle

Existing Camera Image

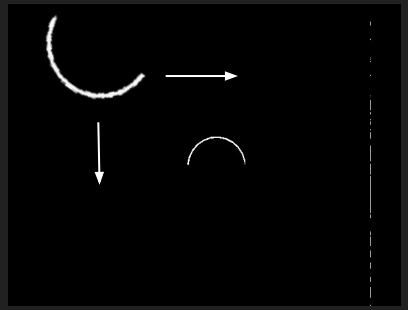


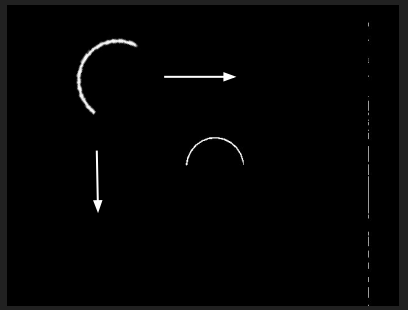
Cropped/Rotated Template

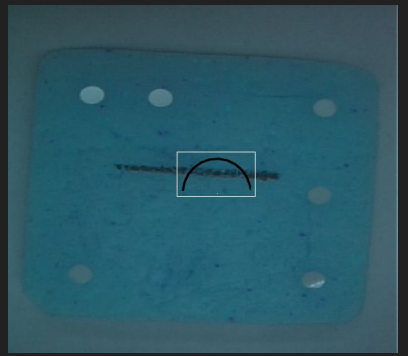
2. Template Matching

First, we identify a range and increment for both scale and rotation of the template (0.8x-2x at 0.1 scale increment, 0°-360° at 3° rotation increment currently)

- 2b - Matching
  - This occurs once for the left image and once for the right image
  - For every scale and for every rotation, the template is transformed and matched on the camera image using opencv's matchTemplate() function
  - The best match is returned, giving the (x,y,width,height) bounding rectangle containing the needle, as well as z rotation of the template for that best match
- Problems/Assumptions:
  - Good for perfectly segmented images, but may match on other needle-like structures
  - Depending on scale and rotation parameters, can be very slow (but the current parameters are fairly quick)
  - Only identifies a single potential needle location (even though the particle filter can accept several)







3. 3D Location Computation

- Goal: compute 3D location of needle origin from the image coordinates found in matching
- Needle Origin in Image
  - matchTemplate returns bounding rectangle, but we need the pixel space coordinates of the needle center
  - Take known origin from original, unmodified template
  - Apply identical rotation and scaling transformation to the original origin to get the origin in the matched template
- Triangulation:
  - Call opencv cv::sfm::triangulatePoints() with given intrinsic camera matrices and needle origin coordinates in left and right images
  - Returns the 3D location of the needle origin
- Problems/Assumptions
  - Because of the method for template creation, this assumes the needle (and in turn the origin) is a good model for the current needle being searched for

Sample Output

```
----------------------------------------------------
Match Summary: Left
----------------------------------------------------
Pixel Location: (284, 202)
Size: width: 108, height: 62
Yaw: degrees = -180, radians = -3.14159
Scale: 1
----------------------------------------------------
Match Summary: Right
----------------------------------------------------
Pixel Location: (265, 202)
Size: width: 108, height: 62
Yaw: degrees = -180, radians = -3.14159
Scale: 1
----------------------------------------------------
Experimental Results
----------------------------------------------------
Pos: (x,y,z)    = (0.00486842, 0.00486842, 0.174329)
Rot: (x,y,z,w) = (0, 0, -1, -4.37114e-08)
Rot: (r,p,y)    = (0, 0, -180)
----------------------------------------------------
```

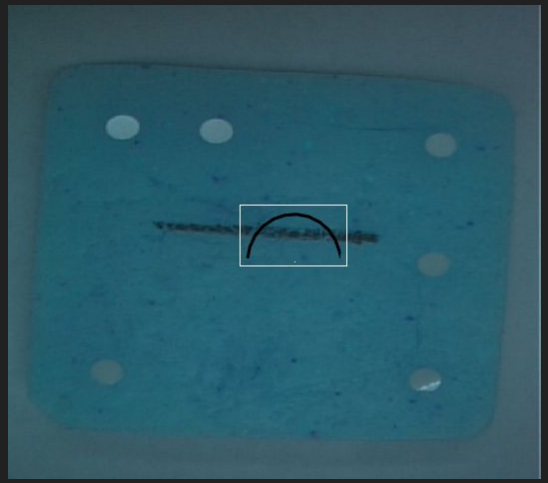
Left Bounding Rectangle

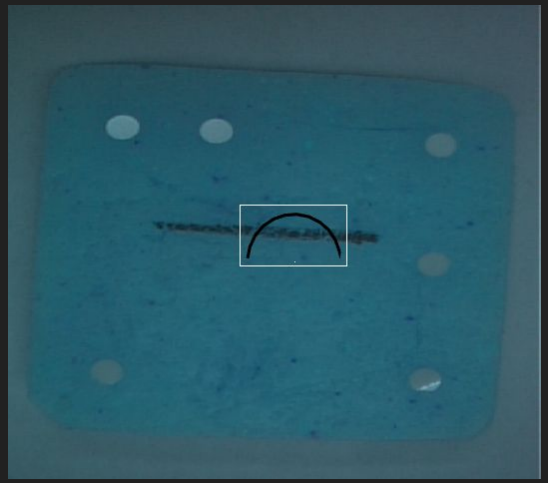
Right Bounding Rectangle

Triangulated Position

Result

The program returns the (x,y,z) location, as well as an euler angle or quaternion representation of the z-axis rotation.





```
Time: 3.09726 s
-------------------------------------------------------
Match Summary: Left
-------------------------------------------------------
Pixel Location: (284, 202)
Size: width: 108, height: 62
Yaw: degrees = -180, radians = -3.14159
Scale: 1
-------------------------------------------------------
Match Summary: Right
-------------------------------------------------------
Pixel Location: (265, 202)
Size: width: 108, height: 62
Yaw: degrees = -180, radians = -3.14159
Scale: 1

Experimental Results
-------------------------------------------------------
Pos: (x,y,z)   = (0.00486842, 0.00486842, 0.174329)
Rot: (x,y,z,w) = (0, 0, -1, -4.37114e-08)
Rot: (r,p,y)   = (0, 0, -180)

Scoring Results
-------------------------------------------------------
True Pos: (x,y,z)   = (0.005, 0.005, 0.177)
True Rot: (x,y,z,w) = (0, 0, -0.999574, 0.0291972)
Pos error (meters)  = 0.00267743
Rot error (degrees) = 3.34623
```