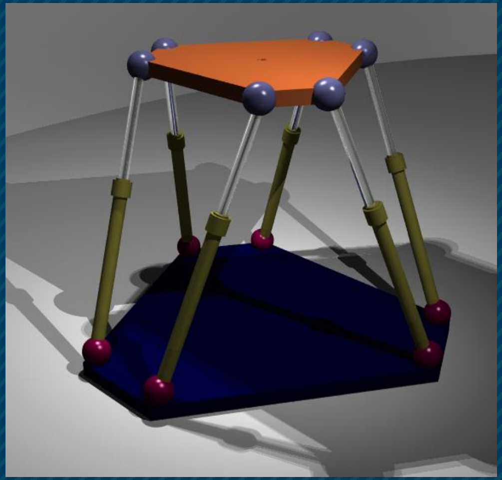
DRAW WITH "DELTA"

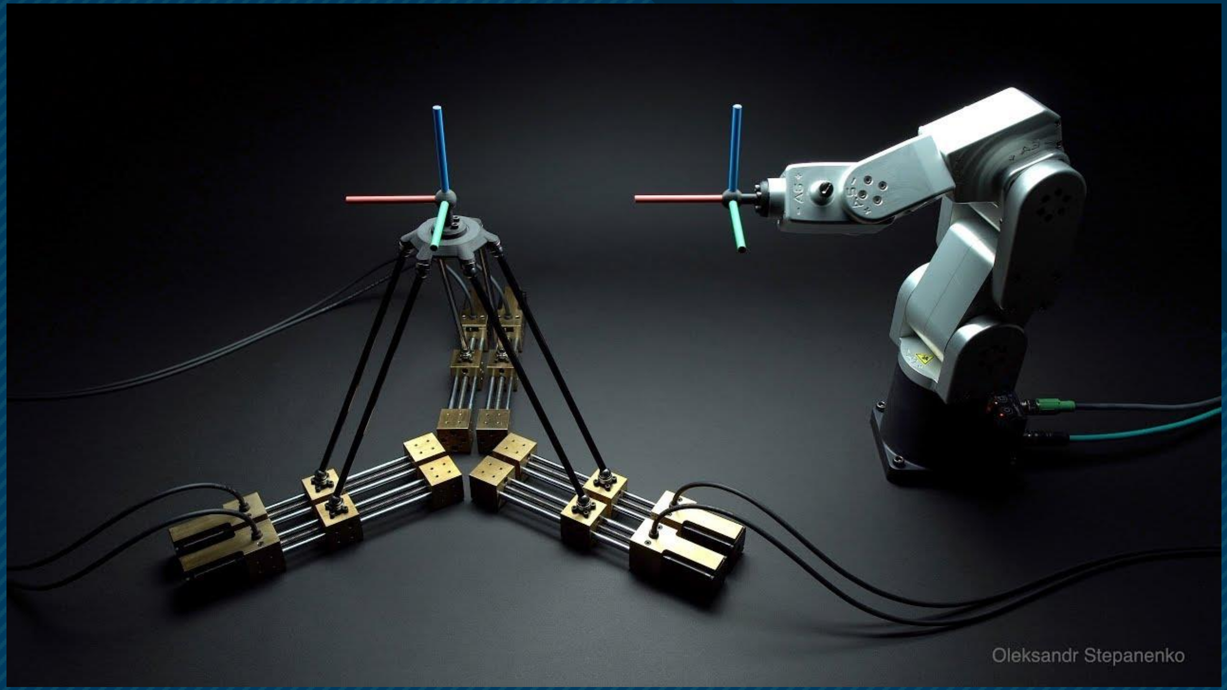
Group Members:

A. Faghihi – A. Irani – H. Effat Panah

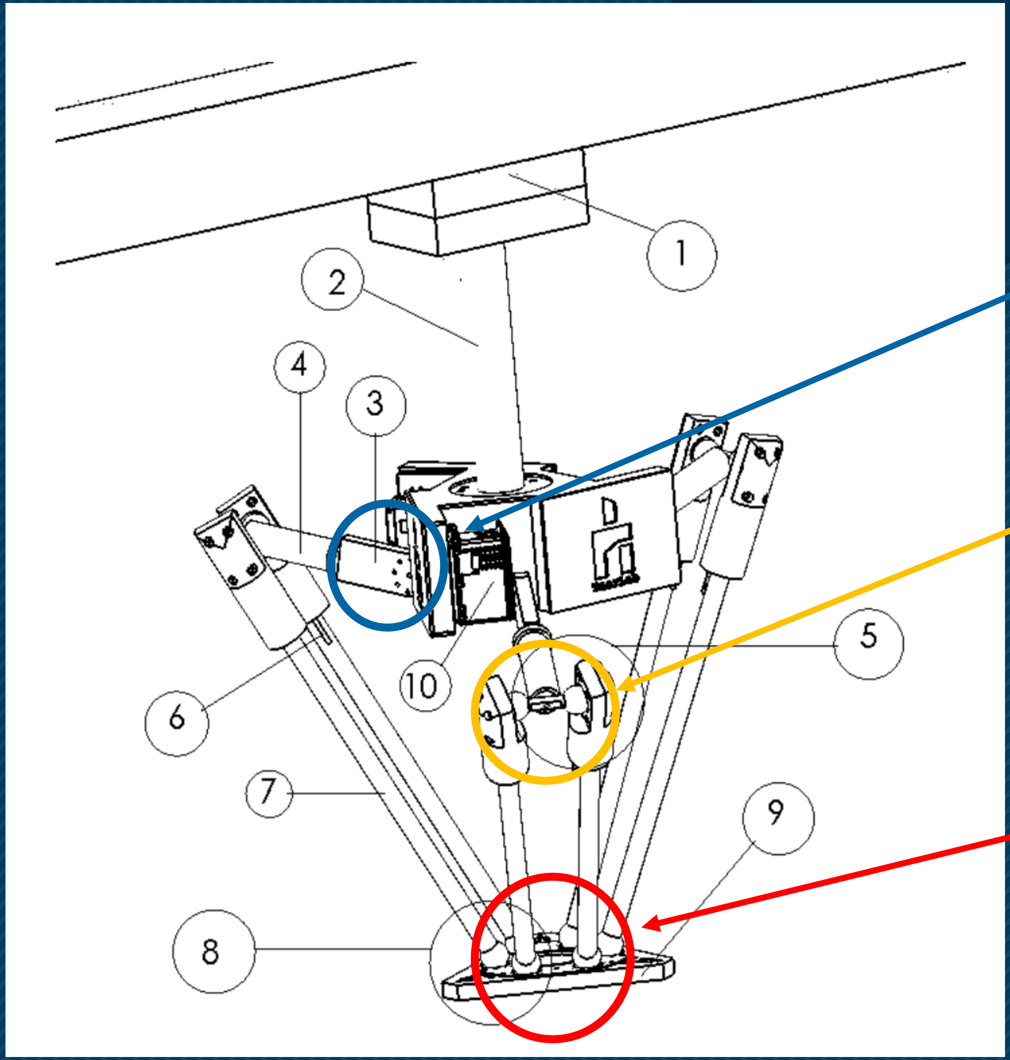
1. Introduction

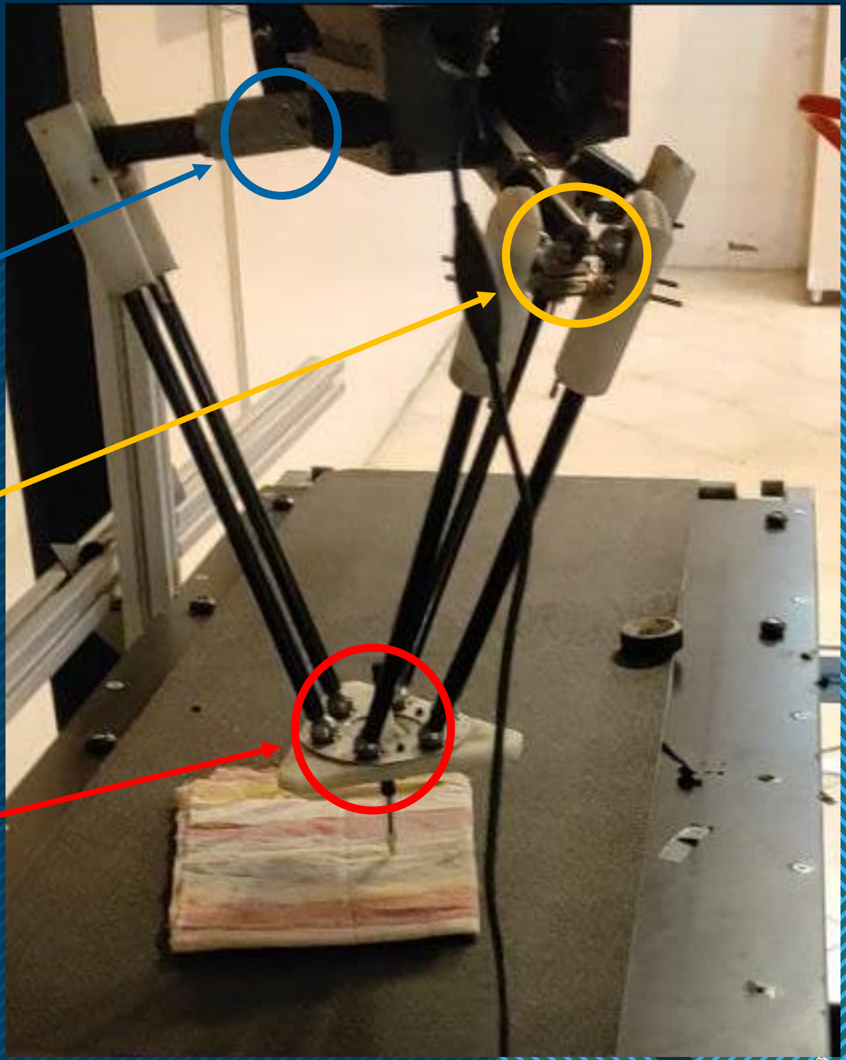
1.1 Parallel vs. Serial Robots





1.2 Delta Robot is a parallel robot





1.3 Motors

- Delta has 3 DYNAMIXEL MX-64 motors
- Operating modes:
  - Current control
  - Position control
  - Velocity control
  - Etc.

1.4 Controlling "Delta"

- Solving "Inverse Kinematic Problem" → finding θ_i ; $i = 1, 2, 3$ given x, y, z

$$E_i \cos\theta_i + F_i \sin\theta_i + G_i = 0 \quad i=1,2,3$$

$a = \boxed{d_B} - \boxed{e_P}$, $b = \boxed{f_P}\sin30 - \boxed{d_B}\cos30$, $c = \boxed{d_P} - \boxed{d_B}\sin30$

$E_1 = 2U(y+a)$

$E_2 = -U(2\cos30\,(x + b) + y + c)$

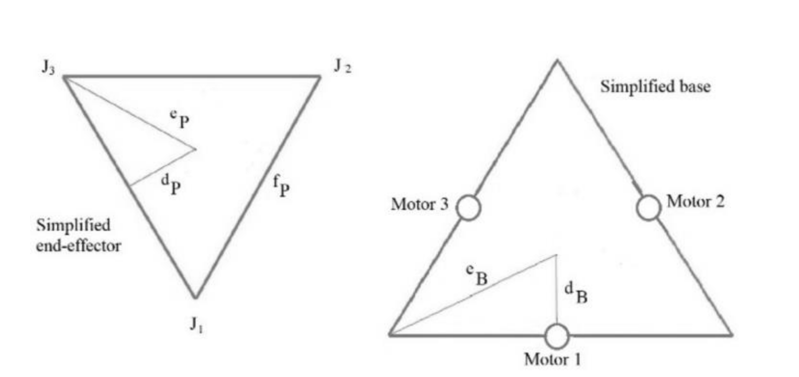
$E_3 = U(2\cos30\,(x - b) - y - c)$

$F_1=F_2=F_3= 2zU$

$G_1 = x^2 + y^2 + z^2 + a^2 + U^2 + 2ya - L^2$

$G_2 = x^2+y^2+z^2+b^2+c^2+U^2+2(xb-yc)-L^2$

$G_3 = x^2+y^2+z^2+b^2+c^2+U^2-2(xb+yc)-L^2$



$d_p = 1.65$

$e_p = 3.3$

$f_p = 5.5$

$d_b = 6.5$

$e_b = 13$

2. Method of Implementation

2.1 Image Processing

Greyscale

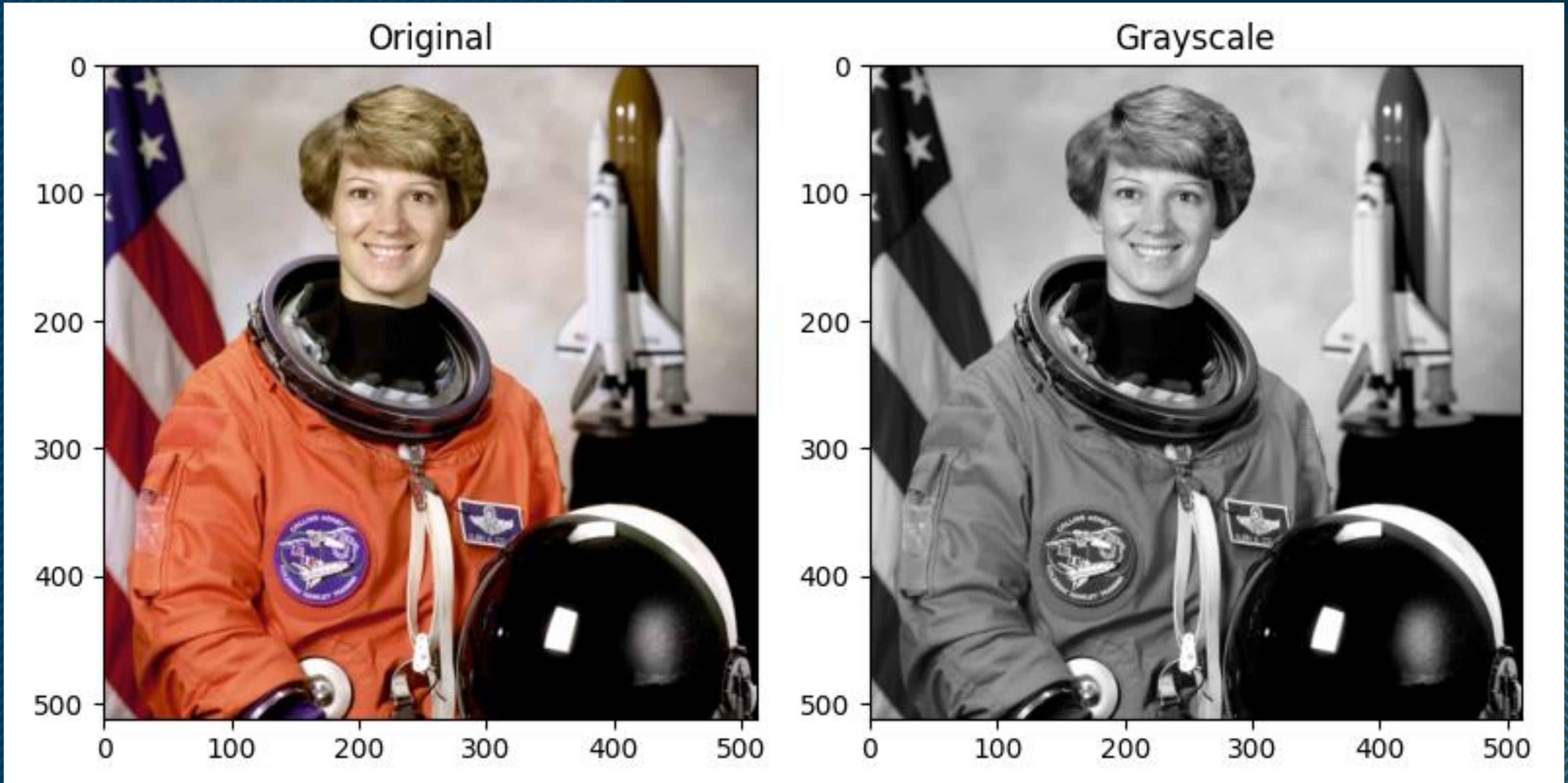
Gaussian Blur

Noise Reduction

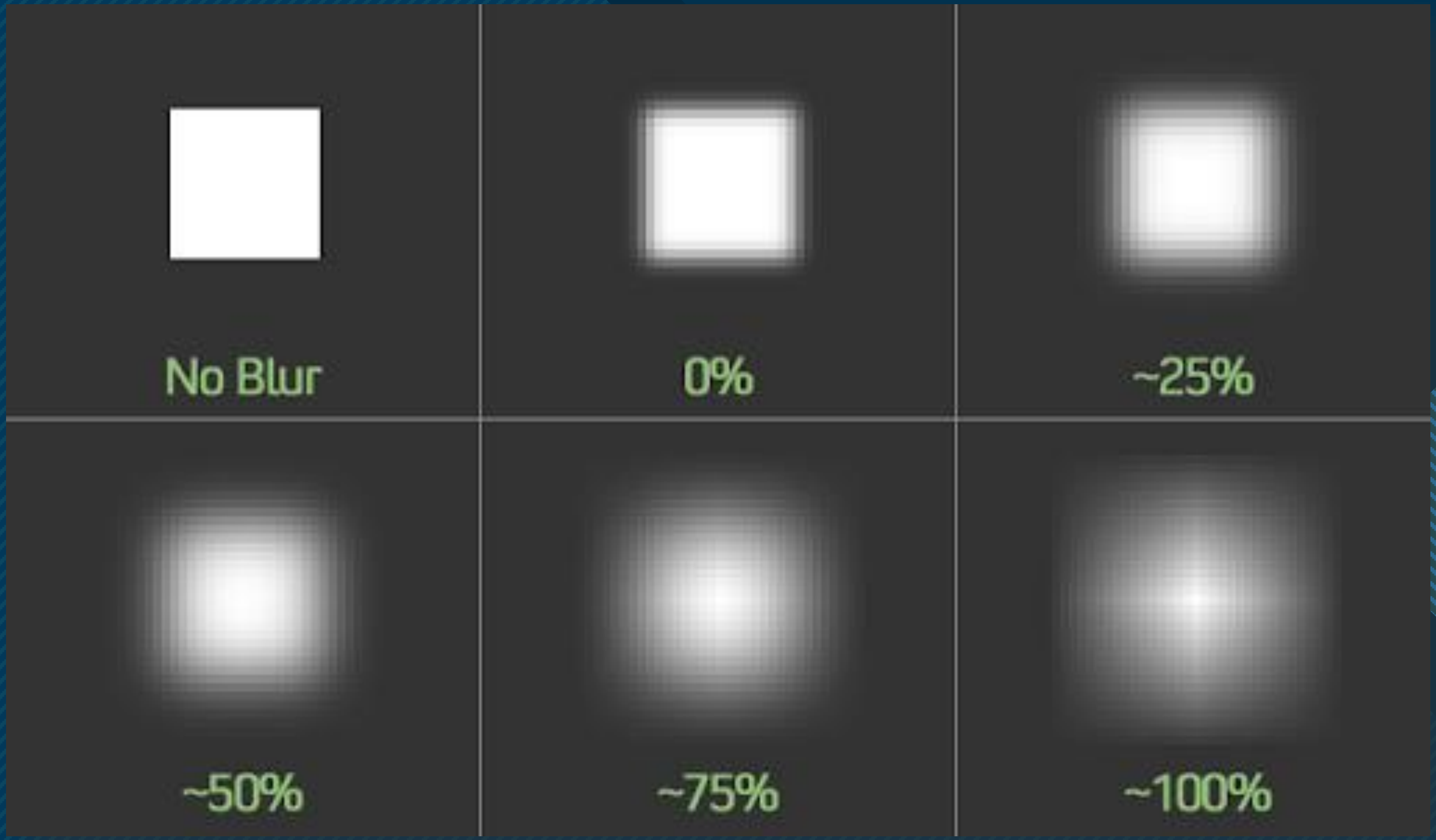
Canny Edge Detection

Find Contour points

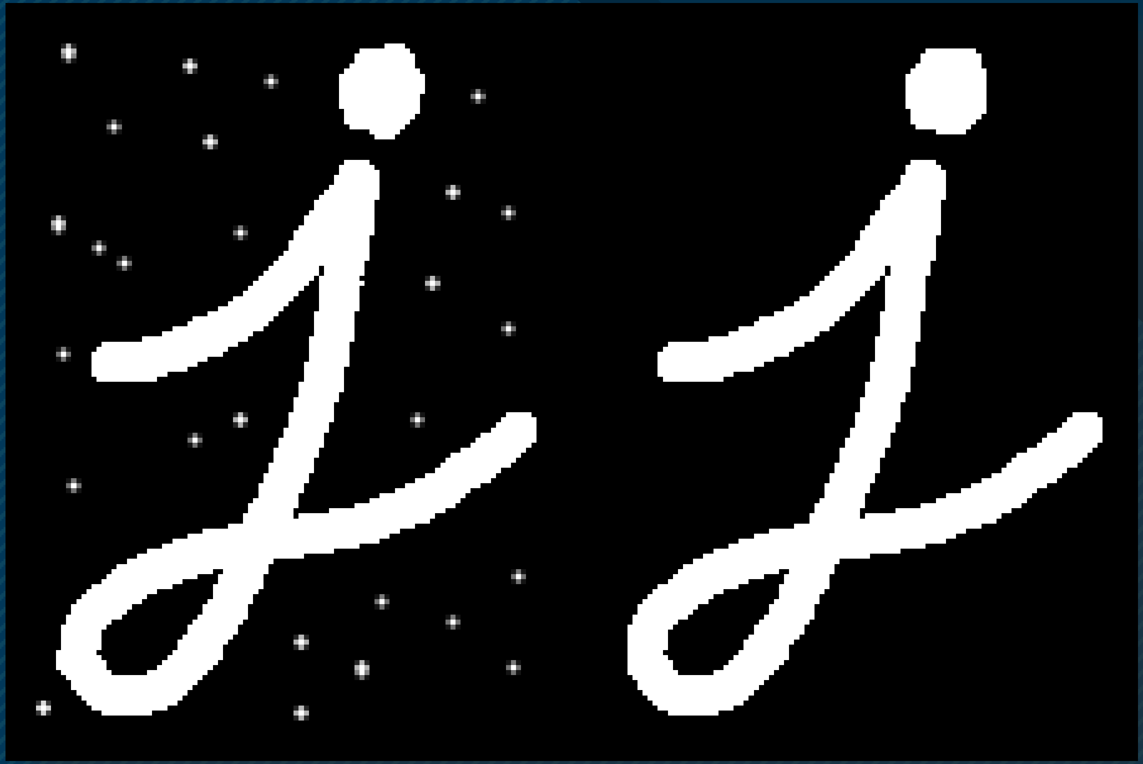
2.1.1 Grayscale



2.1.2 Gaussian Blur



2.1.2 Noise Reduction (using morphological transformations)



2.1.3 Canny Edge Detection

Output for different thresholds

2.2 Draw with “Delta”!

Sampling From Contour points & Normalize

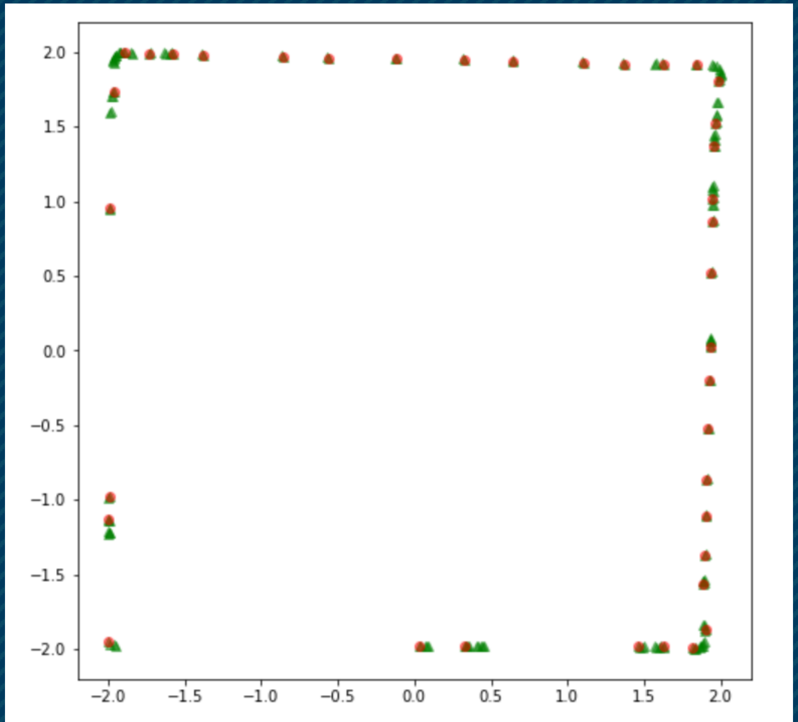
Go to homing position

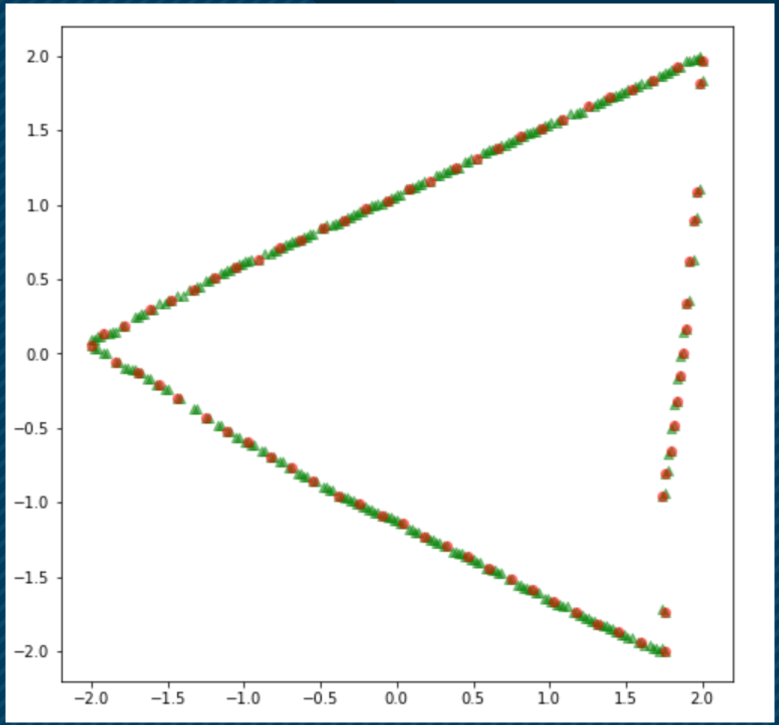
Go to the first sampled point’s position

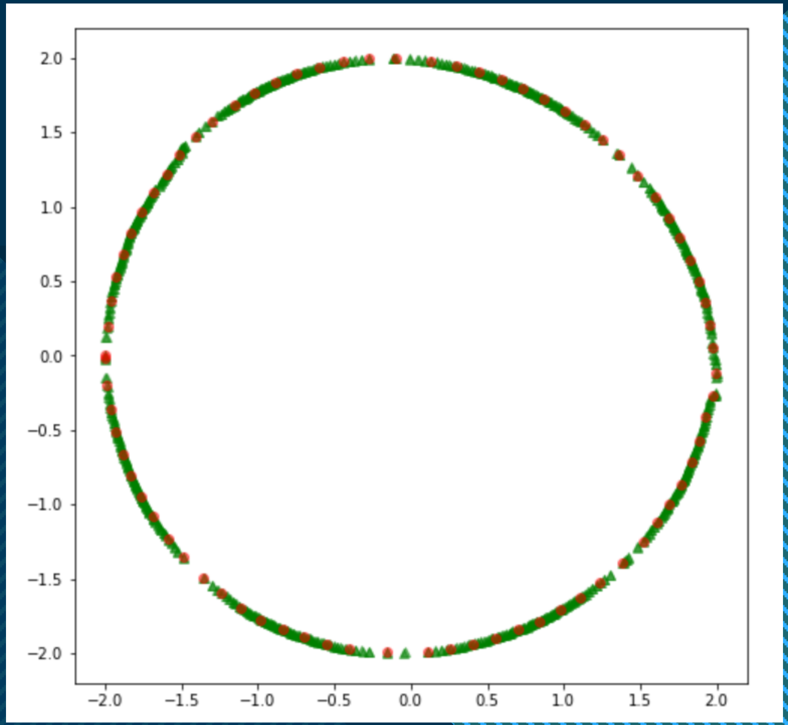
At each step: go from one point to the next

Control velocity to minimize position error

2.2.1 Normalization & sampling







Green: contour points - Red: sampled points

- Important parameter → sampling radius, normalization scale

2.2.2 Control "Delta" movement

```
step 1: go to home position
step 2: start from the first sampled point
step 3: go to first point's position, and set the next point in list as "goal"
step 4: go to the "goal" point in "velocity mode"
step 5: control velocity based on difference in position
step 6: at each iteration check the error
step 7: if error is less than threshold go to "step 8" else go to "step 5"
step 8: set the next point in list as your "goal" and go to "step 4"
```

- Points coordinates are in (x, y), so we need to find θ using IKP

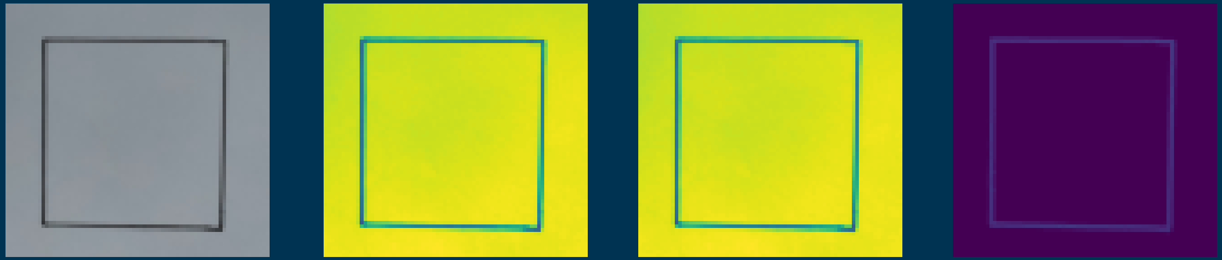
2.2.3 Important functions

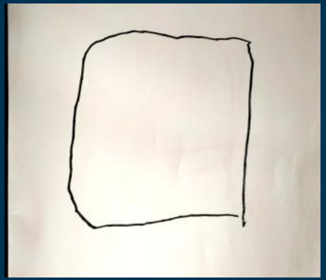
```python
def go_to_pos(x, y, z, alpha=10, mode="velocity"):
    goal_pos = IKP(x, y, z).astype(int)
    print("goal  ",goal_pos)
    if mode == "velocity":
        curr_pos = np.array(read_position())
        print("curr   ",curr_pos)
        error = goal_pos - curr_pos
        g_vel = np.ones(1)
        while True:
            g_vel = normalizer(error, alpha).astype(int)
            print("vel   ",g_vel)
            goal_velocity(g_vel[0], g_vel[1], g_vel[2])
            curr_pos = np.array(read_position())
            print("curr   ",curr_pos)
            error = goal_pos - curr_pos
            if (np.sqrt(np.square(error).sum()) < 30) or (np.sum(g_vel == 0) == 3):
                break
        print("----------------loop end")
    else:
        goal_position(goal_pos[0], goal_pos[1], goal_pos[2])
```

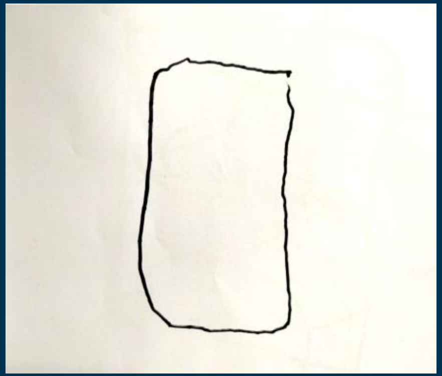
```python
def draw_shape(image_path):
    hp = [0, 0, -24.8]
    drawing_z = -25.84
    points = point_finder(image_path)
    sampled_points = sample_points2(points, 0.12)
    homing(IKP(hp[0], hp[1], hp[2]).astype(int))
    time.sleep(2.5)
    go_to_pos(sampled_points[0][0], sampled_points[0][1], hp[2], mode="position")
    time.sleep(2.5)
    go_to_pos(sampled_points[0][0], sampled_points[0][1], drawing_z + 0.34, mode="position")
    time.sleep(2.5)
    change_operating_mode(velocity)
    for point in sampled_points[1:]:
        print("point: ", point[0], point[1])
        go_to_pos(point[0], point[1], drawing_z)
        goal_velocity(0, 0, 0)
    go_to_pos(sampled_points[0][0], sampled_points[0][1], drawing_z)
    goal_velocity(0, 0, 0)
```

3. Results

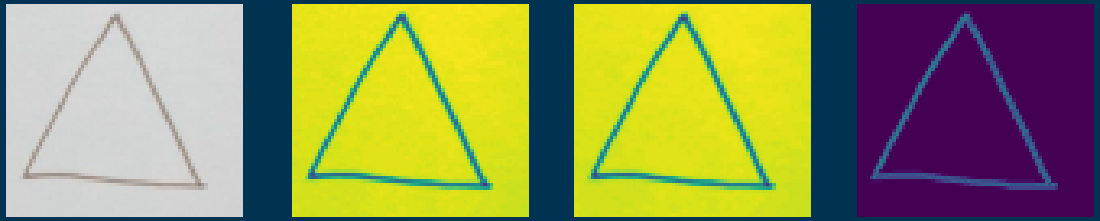
3.1 Square & Rectangle

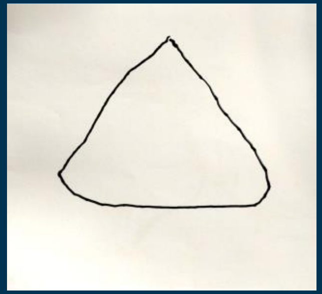


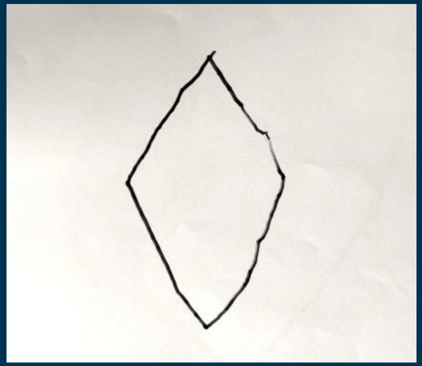




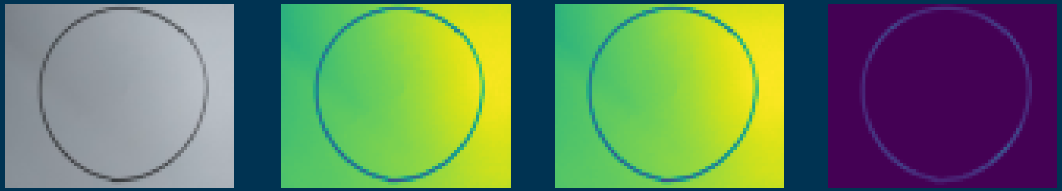
3.2 Triangle & Diamond

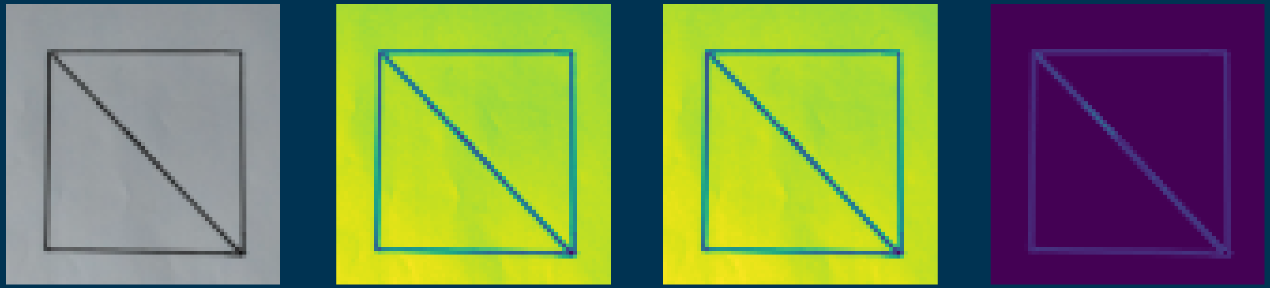


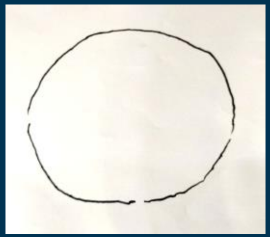


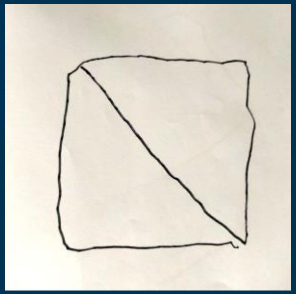


3.3 Circle & Square with diameter

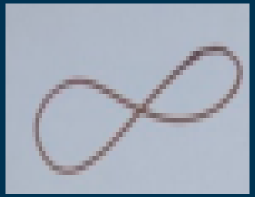
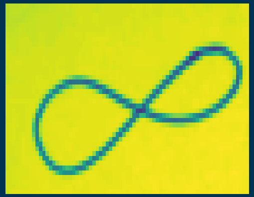
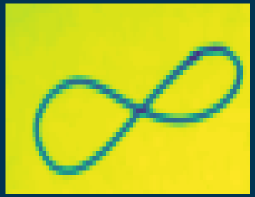
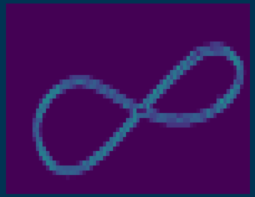


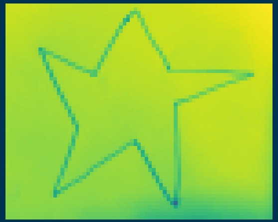
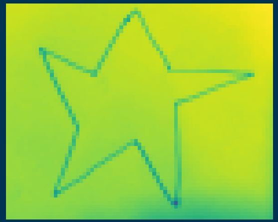
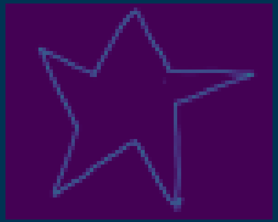


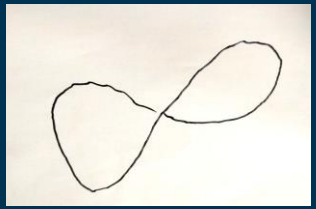




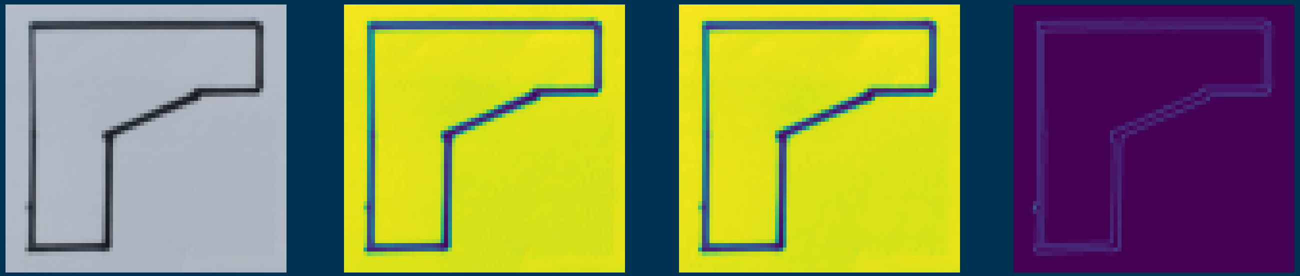
3.4 Infinity & Star

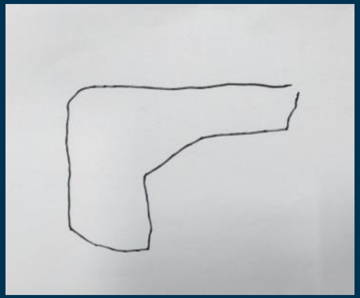








3.5 Random Shape





4. References

1. https://ieeexplore.ieee.org/abstract/document/8734975
2. https://en.wikipedia.org/wiki/Canny_edge_detector
3. https://en.wikipedia.org/wiki/Denavit%E2%80%93Hartenberg_parameters
4. https://en.wikipedia.org/wiki/Serial_manipulator
5. https://en.wikipedia.org/wiki/Parallel_manipulator
6. https://en.wikipedia.org/wiki/Delta_robot
7. https://www.tutorialspoint.com/opencv/opencv_gaussian_blur.htm
8. https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_morphological_ops/py_morphological_ops.html
9. https://docs.opencv.org/3.4/d4/d73/tutorial_py_contours_begin.html

Thank You!