LAMPIRAN-LAMPIRAN

LAMPIRAN L-1

Program-program:

1. Program Simulasi:

```
import env, sensors , feature
import random
import pygame
import math
import numpy as np

def random_color():
    levels = range(32,256,32)
    return tuple(random.choice(levels) for _ in range(3))

FeatureMap = feature.featuresDetection()
environment = env.buildenvironment((800,1400))
environment.originalMap = environment.map.copy()
laser = sensors.LaserSensor(200,environment.originalMap,uncertenty=(0.01,0.0))
#tdnya 0.5 001
environment.map.fill((255,255,255))
environment.infomap = environment.map.copy()
originalMap = environment.map.copy()

position = np.array([90, 552])          # initial robot pos

running = True
FEATURE_DETECTION = True
BREAKE_POINT_INO = 0

while running:
    environment.infomap = originalMap.copy()
    FEATURE_DETECTION = True
    BREAKE_POINT_INO = 0
    END_POINT = [0,0]
    sensorON = True

    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False
    if pygame.mouse.get_focused():
        sensorON = True

    if sensorON:
```

```
        if position[0]<1210: #posisi target
            # robot_step = np.array([0,0])        #perpindahan y
            # position = np.array([330, 552])
            robot_step = np.array([10,0])        #perpindahan x
        else:
            robot_step = np.array([0,0])

        position = position + robot_step
        laser.position = position
        sensor_data = laser.sense_obstacle()
        FeatureMap.laser_points_set(sensor_data)
        print("epoch")

        #Variabel-variabel
        person = []
        counter_person = 0
        n_counter = []
        FeatureMap.CIRCLES_DETECTED = []

        #draw all of the detected points
        # map_pts = np.array(FeatureMap.LASERPOINTS)
        # print("map= \n", FeatureMap.LASERPOINTS)
        # map_pts = map_pts[:,0]
        # COLOR = random_color()
        # for point in map_pts:
        #         environment.infomap.set_at((int(point[0]),int(point[1])),
(0,255,0))
        #         pygame.draw.circle(environment.infomap,COLOR,(int(point[0]),int(p
oint[1])),2,0)

        circless = []
        results = []
        while BREAKE_POINT_INO < abs(FeatureMap.NP - FeatureMap.PMIN):
            # print("BP LAST: ",BREAKE_POINT_INO,abs(FeatureMap.NP -
FeatureMap.PMIN))
            line_seedSeg = FeatureMap.seed_segment_detection(laser.position,
BREAKE_POINT_INO)
            circle_seedSeg =
FeatureMap.circle_seed_segment_detection(laser.position, BREAKE_POINT_INO)
            line_state = False
            circle_state = False
            circle_state2 = False

            if line_seedSeg == False and  circle_seedSeg == False:
                # BREAKE_POINT_INO =FeatureMap.break_point_backup
```

```
                break
            elif (line_seedSeg != False) and (circle_seedSeg != False):
                    line_state = True
            elif line_seedSeg == False and circle_seedSeg != False:
                circle_state = True
            elif circle_seedSeg == False and line_seedSeg != False:
                line_state = True


            if line_state == True:
                line_seedSegment = line_seedSeg[0]
                PREDICTED_POINTS_TODRAW = line_seedSeg[1]
                INDICES = line_seedSeg[2]
                results = FeatureMap.seed_segment_growing(INDICES,BREAKE_POINT_INO)
                results_line = results
                if results != False:
                    OUTERMOST = results[2]
                    #Balik ke circle detection kalu garis pendek
                    if (FeatureMap.distpoint2point(OUTERMOST[0], OUTERMOST[1])<20)
:
                        circle_results =
FeatureMap.circle_seed_segment_growing(INDICES,BREAKE_POINT_INO)
                        x_circ, y_circ, r_circ =  FeatureMap.CIRCLE_PARAMS
                        if (FeatureMap.res_circle < FeatureMap.res_line) and
(r_circ<10):
                            results = circle_results
                            circle_state2 = True
                            circle_state = True   #different bool var to escape
circle segment growing func
                            line_state = False

                elif results == False and circle_seedSeg != False:
                    circle_state = True
                    line_state = False


            if circle_state == True:
                if circle_state2 == True:
                    line_seedSegment = circle_seedSeg[0]
                    PREDICTED_POINTS_TODRAW = circle_seedSeg[1]
                    INDICES = circle_seedSeg[2]
                else:
                    if circle_seedSeg != False:
                        line_seedSegment = circle_seedSeg[0]
                        PREDICTED_POINTS_TODRAW = circle_seedSeg[1]
```

```
                        INDICES = circle_seedSeg[2]
                        results =
FeatureMap.circle_seed_segment_growing(INDICES,BREAKE_POINT_INO)
                        x_circ, y_circ, r_circ =  FeatureMap.CIRCLE_PARAMS
                        if (FeatureMap.res_circle > 10) or (r_circ>20):
                            if(line_seedSeg != False):
                                circle_state = False
                                line_seedSegment = line_seedSeg[0]
                                PREDICTED_POINTS_TODRAW = line_seedSeg[1]
                                INDICES = line_seedSeg[2]
                                results = results_line


                    if circle_seedSeg==False:
                        continue


            if results == False:
                    BREAKE_POINT_INO = INDICES[1]
                    circle_state = False
                    circle_state2 = False
                    line_state = False
                    continue
            line_eq = results[1]
            params = results[4]
            line_seg = results[0]
            OUTERMOST = results[2]

            BREAKE_POINT_INO = results[3]

            if line_state == True:
                END_POINT[0] = FeatureMap.projection_point2line(OUTERMOST[0],
params)
                END_POINT[1] = FeatureMap.projection_point2line(OUTERMOST[1],
params)
            if circle_state == True:
                if(params[2]<10):
                    FeatureMap.CIRCLES_DETECTED.append(params)
                END_POINT[0] = FeatureMap.projection_point2circle(OUTERMOST[0],
params)
                END_POINT[1] = FeatureMap.projection_point2circle(OUTERMOST[1],
params)

         #    OBJECT CLASSIFICATION
            circles_detected = FeatureMap.CIRCLES_DETECTED
```

```
            if circles_detected :
                for n in range(0,len(circles_detected)):
                    if n == (len(circles_detected)-1):
                        break
                    if n in n_counter:
                        continue
                    distance_circles =
FeatureMap.dist_twocircles(circles_detected[n], circles_detected[n+1])
                    if distance_circles <= 40:
                        counter_person += 1
                        person.append([circles_detected[n],circles_detected[n+1],
counter_person])
                        n_counter.append(n)
                        n_counter.append(n+1)

            COLOR = random_color()
            CIRCLE_COLOR = random_color()
            RECT_COLOR = (205,92,92)


            for point in line_seg:
                environment.infomap.set_at((int(point[0][0]),int(point[0][1])),
(0,255,0))
                pygame.draw.circle(environment.infomap,COLOR,(int(point[0][0]),int(
point[0][1])),2,0)


            # draw robot position:
            # externalMap= pygame.image.load('map/map_autotest.png')
            # environment.infomap.blit(externalMap,(0,0))
            pygame.draw.circle(environment.infomap, (0,0,255) ,laser.position,8) #
draw robot pos

            if line_state == True:
                pygame.draw.line(environment.infomap,  (0,0,255) ,END_POINT[0],
END_POINT[1],2)

            if circle_state == True or circle_state2==True:
                x_c, y_c, r_c = params
                pygame.draw.circle(environment.infomap, CIRCLE_COLOR
,(x_c,y_c),r_c,2)

            #draw square
            if person:
                for n in range(0, len(person)):
```

```
                    circle1, circle2, people = person[n]
                    x_c1, y_c1, r_c1 = circle1
                    x_c2, y_c2, r_c2 = circle2

                    if x_c1 < x_c2 :
                        left_rect = x_c1-r_c1-5
                    else:
                        left_rect = x_c2-r_c2-5

                    if y_c1 < y_c2 :
                        top_rect = y_c1-r_c1-5
                    else:
                        top_rect = y_c2-r_c2-5

                    width_rect = r_c1 + abs(x_c2 - x_c1) + r_c2 + 10
                    height_rect = r_c1 + abs(y_c2 - y_c1) + r_c2 + 10
                    pygame.draw.rect(environment.infomap, RECT_COLOR,(left_rect,
top_rect, width_rect, height_rect), 2)
                    # not using text because it takes too long for program to run

            environment.dataStorage(sensor_data)
            environment.show_sensorData(environment.infomap)

    environment.map.blit(environment.infomap,(0,0))
    pygame.display.update()
```

2. Program *Environment:*

```
import math
import pygame

class buildenvironment:
    def __init__(self,MapDimentions):
        pygame.init()
        self.pointCloud = []

        self.externalMap= pygame.image.load('map/map_autotest.png')
        self.maph, self.mapw = MapDimentions
        self.MapWindowsName = "KYPCOBA"
        pygame.display.set_caption(self.MapWindowsName)
        self.map = pygame.display.set_mode((self.mapw,self.maph))
        self.map.blit(self.externalMap,(0,0))

        self.black = (0,0,0)
```

```
        self.grey = (70, 70, 70)
        self.blue = (0, 0, 255)
        self.Green = (0, 255, 0)
        self.Red = (255, 0, 0)
        self.white = (255, 255, 255)


    def AD2pos(self, distance, angle, robot_position):
        x = distance*math.cos(angle) + robot_position[0]
        y = -distance*  math.sin(angle) + robot_position[1]
        return int(x),int(y)

    def dataStorage(self,data):
        for element in data:
            point = self.AD2pos(element[0],element[1],element[2])
            if point not in self.pointCloud:
                self.pointCloud.append(point)

    def show_sensorData(self, map):
        for point in self.pointCloud:
            map.set_at((int(point[0]), int(point[1])),(255, 0, 0))
```

3. Program Sensor:

```
import math
import numpy as np
import pygame


def uncertainty_add(distance, angle, sigma):
    mean = np.array([distance, angle])
    covariance = np.diag(sigma ** 2)
    distance, angle = np.random.multivariate_normal(mean, covariance)
    distance = max(distance, 0)
    angle = max(angle, 0)
    return [distance, angle]


class LaserSensor:
    def __init__(self, Range, map, uncertenty):
        self.Range = Range
        self.speed = 4
        self.sigma = np.array([uncertenty[0], uncertenty[1]])
        self.position = (0, 0)
        self.map = map
        self.W, self.H = pygame.display.get_surface().get_size()
        self.sensedobstacles = []
```

```
    def distance(self, obstaclePosition):
        px = (  obstaclePosition[0]- self.position[0] ) ** 2
        py = (   obstaclePosition[1] -self.position[1]) ** 2
        return math.sqrt((px + py))

    def sense_obstacle(self):
        data = []
        x1, y1 = self.position[0], self.position[1]
        dag =np.linspace(0, 2* math.pi,400,False )
        for angle in dag:
            x2, y2 = (x1 + self.Range * math.cos(angle), y1 - self.Range *
math.sin(angle))
            for i in range(0, 100):
                u = i / 100
                x = int(x2 * u + x1 * (1 - u))
                y = int(y2 * u + y1 * (1 - u))
                if 0 < x < self.W and 0 < y <self.H:
                    color = self.map.get_at((x,y))

                    if (color[0],color[1],color[2]) == (0,0,0):
                        distance = self.distance((x,y))
                        output = uncertainty_add(distance,angle, self.sigma)
                        output.append(self.position)
                        data.append(output)
                        break
        if len(data) > 0:
            return data
        else:
            return False
```

4. Program Pendeteksi Fitur:

Mmm

```
from ctypes import pointer
import math
from fractions import Fraction
from re import X
from turtle import distance

import numpy as np
from scipy.odr import *

import matplotlib.pyplot as plt


class featuresDetection:
    # Class variables
    res_line = 0


    def __init__(self):
        self.EPSILON = 10
        self.DELTA = 8
        self.DELTA_CIRCLE = 10
        self.EPSILON_CIRCLE = 20
        self.SNUM = 6
        self.SNUM_CIRCLE = 5
        self.PMIN = 15
        self.PMIN_CIRCLE = 6
        self.GMAX = 10 #30
        self.SEED_SEGMENTS = []
        self.LINE_SEGMENTS = []
        self.CIRCLE_SEGMENTS = []
        self.CIRCLES_DETECTED = []
        self.LASERPOINTS = []
        self.LINE_PARAMS = None
        self.CIRCLE_PARAMS = None
        self.NP = len(self.LASERPOINTS) - 1
        self.LMIN = 1
        self.LR = 0
        self.PR = 0
        self.res_line2 = 0
        self.res_circle2 = 0
        self.LEN_LINE_SEGMENTS = 0
        self.LEN_CIRCLE_SEGMENTS = 0
        self.dumbvar = 0
```

```
    def distpoint2point(self, point1, point2):
        Px = (point1[0] - point2[0]) ** 2
        Py = (point1[1] - point2[1]) ** 2
        return math.sqrt(Px + Py)

    def distpoint21ine(self, params, point):
        A, B, C = params

        distance = abs(A * point[0] + B * point[1] + C) / math.sqrt(A ** 2 + B **
2)
        return distance

    def line2points(self, m, b):
        x = 5
        y = m * x + b

        x2 = 2000
        y2 = m * x2 + b
        return [(x, y), (x2, y2)]

    def lineForm_G2SI(self, A, B, C):
        m = -A / B
        B = - C / B
        return m, B

    # slope-intercept to general form
    def lineForm_si2G(self, m, B):
        A, B, C = - m, 1, -B
        if A < 0:
            A, B, C = -A, -B, -C
        den_a = Fraction(A).limit_denominator(1000).as_integer_ratio()[1]
        den_c = Fraction(C).limit_denominator(1000).as_integer_ratio()[1]

        gcd = np.gcd(den_a, den_c)
        lcm = den_a * den_c / gcd

        A = A * lcm
        B = B * lcm
        C = C * lcm
        return [A, B, C]

    def line_intersect_general(self, params1, params2):

        a1, b1, c1 = params1
        a2, b2, c2 = params2
```

```
        x = (c1 * b2 - b1 * c2) / (b1 * a2 - a1 * b2)
        y = (a1 * c2 - a2 * c1)/(b1 * a2 - a1 * b2)
        return x, y

    def points_2Line(self, point1, point2):
        m, b = 0, 0
        if point2[0] == point1[0]:
            pass
        else:
            m = (point2[1] - point1[1]) / (point2[0] - point1[0])
            b = point2[1] - m * point2[0]
        return m, b

    def projection_point2line(self, point, params):
        m, b = params
        x, y = point
        m2 = -1 / m
        c2 = y - m2 * x
        intersection_x = - (b - c2) / (m - m2)
        intersection_y = m2 * intersection_x + c2
        return intersection_x, intersection_y

    def AD2pos(self, distance, angle, robot_position):
        x = distance * math.cos(angle) + robot_position[0]
        y = -distance * math.sin(angle) + robot_position[1]
        return int(x), int(y)

    def laser_points_set(self, data):
        self.LASERPOINTS = []
        if not data:
            pass
        else:
            for point in data:
                coordinates = self.AD2pos(point[0], point[1], point[2]) # distance,
angle, robot_position
                self.LASERPOINTS.append([coordinates, point[1]])        # data
disimpan
        self.NP = len(self.LASERPOINTS) - 1


    def liniar_func(self, p, x):
        m, b = p
        return m * x + b

    def odr_fit(self, laser_points):
```

```
        x = np.array([i[0][0] for i in laser_points])
        y = np.array([i[0][1] for i in laser_points])
        linar_model = Model(self.liniar_func)
        data = RealData(x, y)

        odr_model = ODR(data, linar_model, beta0=[0., 0.])

        out = odr_model.run()
        self.res_line = out.sum_square
        m, b = out.beta
        return m, b


    ## CIRCLEEEEEEEEEEEEEEEEEEEEEEEEEEEEE!!!!!!!!!!

    def return_circle(self, c):
        x_c = c[1] / 2
        y_c = c[2] / 2
        r = c[0] + x_c ** 2 + y_c ** 2
        return x_c, y_c, np.sqrt(r)

    def circle_fit(self, laser_points):
        # y, x = pts[:, 0], pts[:, 1]
        x = np.array([i[0][0] for i in laser_points])
        y = np.array([i[0][1] for i in laser_points])
        N = len(x)
        x_mean = np.mean(x)
        y_mean = np.mean(y)
        u = x - x_mean
        v = y - y_mean

        S_uuu = np.sum(u**3)
        S_vvv = np.sum(v**3)

        S_uu = np.sum(u ** 2)
        S_vv = np.sum(v ** 2)
        S_uv = np.sum(u * v)

        S_uvv = np.sum(u*(v**2))
        S_uuv = np.sum((u**2)*v)


        u_c = (S_uuu*S_vv + S_uvv*S_vv - S_vvv*S_uv - S_uuv*S_uv)/(2*(S_uu * S_vv -
(S_uv)**2))
        v_c = (S_uu*S_vvv + S_uuv*S_uu - S_uuu*S_uv - S_uvv*S_uv)/(2*(S_uu * S_vv -
```

```
(S_uv)**2))

        x_c = u_c + x_mean
        y_c = v_c + y_mean

        R = np.sum(np.sqrt((u - u_c)**2 + (v - v_c)**2)) / N

        # Calcul des distances au centre (xc_1, yc_1)
        Ri_1     = np.sqrt((x-x_c)**2 + (y-y_c)**2)
        # residu2_2  = sum((Ri_2**2-R_2**2)**2)
        self.res_circle = np.sum((Ri_1-R)**2)
        # self.res_circle =np.sum((Ri_1-R))
        return [x_c, y_c, R]



    def distpoint2circle(self, circle_params, point):
        x_c, y_c, r_c = circle_params
        x, y = point
        dist =np.sqrt(abs((x-x_c)**2+(y-y_c)**2))-r_c
        return dist

    def projection_point2circle(self, point, circle_params):
        x, y = point
        x_c, y_c, r_c = circle_params
        delta_x = x-x_c
        delta_y = y-y_c
        theta = math.atan(delta_y/delta_x)
        intersection_x = x_c + (r_c*math.cos(theta))
        intersection_y = y_c + (r_c*math.sin(theta))
        return intersection_x, intersection_y

    def distpoint2point_incircle(self, params, first_point,
next_point):      #Belum dipakee
        x1, y1 = first_point
        x2, y2 = next_point
        x_c, y_c, r_c = params
        def slope(x1, y1, x2, y2): # Line slope given two points:
            return (y2-y1)/(x2-x1)

        def angle(s1, s2):
            return math.degrees(math.atan((s2-s1)/(1+(s2*s1))))

        slope1 = slope(x1, y1, x_c, y_c)
        slope2 = slope(x2, y2, x_c, y_c)
```

```
        ang = angle(slope1, slope2)
        # print('Angle in degrees = ', ang)

        dist = (ang/360)*2*math.pi*r_c
        return dist

    def dist_twocircles(self, params1, params2):
        x_c1, y_c1, r_c1 = params1
        x_c2, y_c2, r_c2 = params2
        point1 = [x_c1, y_c1]
        point2 = [x_c2, y_c2]

        outer_point1 = self.projection_point2circle(point2, params1)
        outer_point2 = self.projection_point2circle(point1, params2)
        dist = self.distpoint2point(outer_point1, outer_point2)
        return dist


    # END CIRCLEEEEEEEEEEEEEEEEEE

    def predictionPoint(self, line_params, sensed_point, robotpos):
        m, b = self.points_2Line(robotpos, sensed_point)
        params1 = self.lineForm_si2G(m, b)
        predx, predy = self.line_intersect_general(params1, line_params)
        return predx, predy

    def seed_segment_detection(self, robot_position, break_point_ind):
        flag = True
        self.NP = max(0, self.NP)
        self.SEED_SEGMENTS = []
        for i in range(break_point_ind, (self.NP - self.PMIN)):
            prediction_point_to_draw = []
            j = i + self.SNUM
            m, c = self.odr_fit(self.LASERPOINTS[i:j])
            params = self.lineForm_si2G(m, c)

            for k in range(i, j):
                flag = True
                prediction_point = self.predictionPoint(params,
self.LASERPOINTS[k][0], robot_position)
                prediction_point_to_draw.append(prediction_point)
                dl = self.distpoint2point(prediction_point, self.LASERPOINTS[k][0])

                if dl > self.DELTA:
```

```
                    flag = False
                    break

                if k<(j-1): # ilangin titk jarak jauh
                    dl_2 = self.distpoint2point(self.LASERPOINTS[k+1][0],
self.LASERPOINTS[k][0])
                    if dl_2 > self.DELTA:
                        flag = False
                        break

                d2 = self.distpoint21ine(params, prediction_point)

                if d2 > self.EPSILON:
                    flag = False
                    break


            if flag:
                self.LINE_PARAMS = params
                return [self.LASERPOINTS[i:j], prediction_point_to_draw, (i, j)]
        return False

    def circle_seed_segment_detection(self, robot_position, break_point_ind):
        flag = True
        self.NP = max(0, self.NP)
        self.SEED_SEGMENTS = []
        for i in range(break_point_ind, (self.NP - self.PMIN_CIRCLE)):
            prediction_point_to_draw = []
            j = i + self.SNUM_CIRCLE
            circle_params = self.circle_fit(self.LASERPOINTS[i:j])
            for k in range(i,j):
                flag = True
                self.break_point_backup = k
                prediction_point_to_draw = robot_position # dumb
                if k == 0:
                    continue
                dl = self.distpoint2point(self.LASERPOINTS[k][0],
self.LASERPOINTS[k-1][0])

                if dl > self.DELTA_CIRCLE:
                    # print(dl,(k-i), "hai",k)
                    flag = False
                    break
            if flag:
                self.CIRCLE_PARAMS = circle_params
```

```
                return [self.LASERPOINTS[i:j],
(self.LASERPOINTS[i][0],self.LASERPOINTS[j][0]), (i, j)]
        return False


    def seed_segment_growing(self, indices, break_point):
        line_eq = self.LINE_PARAMS
        i, j = indices
        PB, PF = max(break_point, i - 1), min(j + 1, len(self.LASERPOINTS) - 1)
        while self.distpoint21ine(line_eq, self.LASERPOINTS[PF][0]) < self.EPSILON:
            if PF > self.NP - 1:
                break
            elif self.distpoint2point(self.LASERPOINTS[PF][0], self.LASERPOINTS[PF-
1][0]) > self.GMAX:
                break
            else:
                m, b = self.odr_fit(self.LASERPOINTS[PB:PF])
                line_eq = self.lineForm_si2G(m, b)
                POINT = self.LASERPOINTS[PF][0]
            PF = PF + 1
            NEXTPOINT = self.LASERPOINTS[PF][0]
            self.test_p_np = [self.distpoint2point(POINT, NEXTPOINT), POINT,
NEXTPOINT]

        PF = PF - 1

        while self.distpoint21ine(line_eq, self.LASERPOINTS[PB][0]) < self.EPSILON:
            if PB < break_point:
                break
            elif self.distpoint2point(self.LASERPOINTS[PB][0],
self.LASERPOINTS[PB+1][0]) > self.GMAX:
                break
            else:
                m, b = self.odr_fit(self.LASERPOINTS[PB:PF])
                line_eq = self.lineForm_si2G(m, b)
                POINT = self.LASERPOINTS[PB][0]
            PB = PB - 1
            NEXTPOINT = self.LASERPOINTS[PB][0]
            self.test_p_np2 = [self.distpoint2point(POINT, NEXTPOINT), POINT,
NEXTPOINT]

        PB = PB + 1
        LR = self.distpoint2point(self.LASERPOINTS[PB][0], self.LASERPOINTS[PF][0])
        PR = len(self.LASERPOINTS[PB:PF])
```

```
        if (LR >= self.LMIN) and (PR >= self.PMIN):
            self.LINE_PARAMS = line_eq
            m, b = self.lineForm_G2SI(line_eq[0], line_eq[1], line_eq[2])
            self.two_points = self.line2points(m, b)
            self.LINE_SEGMENTS.append((self.LASERPOINTS[PB + 1][0],
self.LASERPOINTS[PF - 1][0]))
            self.LEN_LINE_SEGMENTS = self.distpoint2point(self.LASERPOINTS[PB +
1][0],self.LASERPOINTS[PF - 1][0])
            return [self.LASERPOINTS[PB:PF], self.two_points,
                    (self.LASERPOINTS[PB + 1][0], self.LASERPOINTS[PF - 1][0]), PF,
(m, b), line_eq]
        else:
            return False


    def circle_seed_segment_growing(self, indices, break_point):
        circle_eq = self.CIRCLE_PARAMS
        i, j = indices
        PB, PF = max(break_point, i - 1), min(j + 1, len(self.LASERPOINTS) - 1)
        while PF < (self.NP - 1):
            if self.distpoint2point(self.LASERPOINTS[PF-1][0],
self.LASERPOINTS[PF][0]) < self.DELTA_CIRCLE:
                if self.distpoint2circle(circle_eq, self.LASERPOINTS[PF][0]) <
self.EPSILON_CIRCLE:
                    circle_eq = self.circle_fit(self.LASERPOINTS[PB:PF])
                    PF = PF + 1
                else:
                  break
            else:
                break
        PF = PF - 1

        while PB > break_point:
            if self.distpoint2point(self.LASERPOINTS[PB+1][0],
self.LASERPOINTS[PB][0]) < self.DELTA_CIRCLE:
                if self.distpoint2circle(circle_eq, self.LASERPOINTS[PB][0]) <
self.EPSILON_CIRCLE:
                    circle_eq = self.circle_fit(self.LASERPOINTS[PB:PF])
                    PB = PB - 1
                else:
                    break
            else:
                break
        PB = PB + 1
```

```
        LR = self.distpoint2point(self.LASERPOINTS[PB][0], self.LASERPOINTS[PF][0])
        PR = len(self.LASERPOINTS[PB:PF])

        if (LR >= self.LMIN) and (PR >= self.PMIN_CIRCLE):
            self.CIRCLE_PARAMS = circle_eq
            self.two_points = 1000 # dumb var
            self.CIRCLE_SEGMENTS.append((self.LASERPOINTS[PB + 1][0],
self.LASERPOINTS[PF - 1][0]))
            return [self.LASERPOINTS[PB:PF], self.two_points,
                    (self.LASERPOINTS[PB + 1][0], self.LASERPOINTS[PF - 1][0]), PF,
circle_eq]
        else:
            return False
```