

# CAAP Quarterly Report

Date of Report: December 26, 2019

Prepared for: *U.S. DOT Pipeline and Hazardous Materials Safety Administration*

Contract Number: 693JK31950002CAAP

Project Title: AI-enabled Interactive Threats Detection using a Multi-camera Stereo Vision System

Prepared by: Arizona State University

Contact Information:

Dr. Yongming Liu (PI), Email: Yongming.Liu@asu.edu

Dr. Yang Yu (Co-PI), Email: yangyu18@asu.edu

For quarterly period ending: December 29, 2019

## **Business and Activity Section**

### **(a) Contract Activity**

Discussion about contract modifications or proposed modifications:

None

Discussion about materials purchased:

1. Stereo cameras
2. Robotic vehicle
3. LED light panels
4. PVC pipe

### **(b) Status Update of Past Quarter Activities**

The project kick-off meeting was held via teleconference on October 30th, 2019 to discuss the scope of work and expectations of the project. The participants of the meeting include Joshua Arnold (PHMSA), Zhongquan Zhou (PHMSA), Dr. Yongming Liu (ASU), Dr. Yang Yu (ASU), Rahul Rathnakumar (ASU), and Chinmay Dixit (ASU). A brief presentation was given by ASU to discuss the project objectives, methodologies, proposed tasks, timeline, and deliverables, followed by questions and comments by PHMSA on the technical approaches and project deliverables.

### **Student Training Activities**

- Chinmay Dixit (MS student) works on stereo vision algorithm development for pipeline imaging, prototype design and preliminary demonstration to test the system in representative pipeline component. (Task 1)
- Utkarsh Pujar (MS student) and Omar Serag (undergrad student) works on assembling a robotic vehicle as the carrier for the prototype device. (Task 1)
- Rahul Rathnakumar (PhD student) works on conducting literature review on AI-based pipeline threats detection and explore suitable methods for detection using depth map information. (Task 2)

### **(c) Cost share activity**

All cost share requirements have been satisfied in the past quarter and detailed financial report will be

## (d) Detailed Description of Work Performed

### 1. Background and Objectives in the 1st Quarter

Stereo vision uses two or more cameras to extract 3D information by estimating the relative depth of points observed in digital images. The principle of stereo vision is illustrated in Fig. 1. In Fig. 1(a), C1 and C2 represent the optical centers of two cameras;  $b$  is the baseline distance between two cameras; P is the object point; and P1 and P2 are the projection of point P in the image plane. Points C1, C2, and P form a plane known as the epipolar plane. Fig. 1(b) shows a top view of the epipolar plane where  $f$  is the focal length. Based on similar triangles, we have:

$$\frac{z}{f} = \frac{x}{x_l} \quad \frac{z}{f} = \frac{x-b}{x_r} \quad \frac{z}{f} = \frac{y}{y_l} = \frac{y}{y_r} \quad (1)$$

where  $(x,y,z)$  is the global coordinate of the object point P, and  $(x_l, y_l, z_l)$  and  $(x_r, y_r, z_r)$  is the coordinate of the projection of point P in the left and right image planes, respectively.

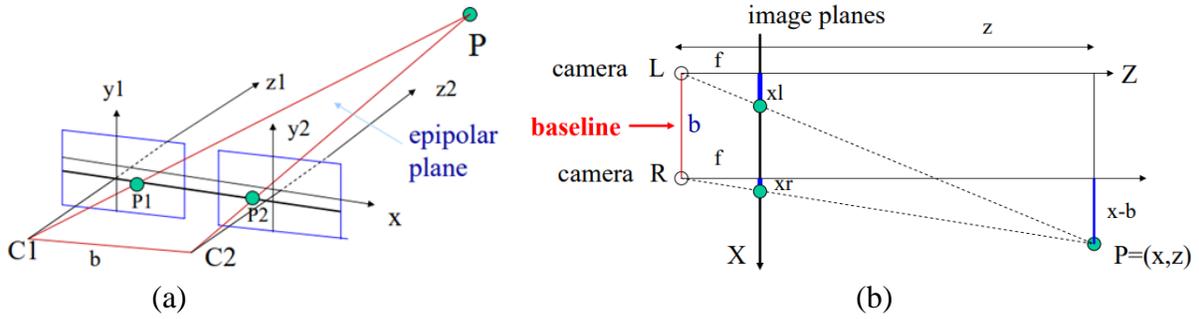


Fig. 1 Principle of stereo vision: (a) epipolar plane; (b) triangulation[1]

Based on the relationships given in Eq. (1), the global coordinate of point P can be calculated as:

$$z = \frac{fb}{(x_l - x_r)} = \frac{fb}{d} \quad x = \frac{b}{d} x_l \quad y = \frac{b}{d} y_l \quad (2)$$

where the difference  $d = (x_l - x_r)$  is known as the disparity. Using Eq. (2), we can determine the depth of any scene point and thus construct a depth map of the observed scene. This method of determining depth from disparity is called triangulation. Fig. 2 shows an example of the depth map constructed using a stereo image pair. In practice, we need to find the corresponding point  $(x_r, y_r)$  in the right image plane for each point  $(x_l, y_l)$  in the left image plane in order to compute the disparity. This is known as the stereo correspondence problem. The common solutions to this problem can be divided into two categories, i.e., correlation-based methods and feature-based methods. The appropriate method for the task of pipeline ILLI will be selected with the goal of achieving a balance between fast inspection speed and high detection accuracy.



Fig. 2 Stereo vision and depth map: (a) input stereo image pair; (b) output depth map[1]

AI technology offers a variety of tools for image processing and recognition. The recent advances in deep learning demonstrate the capability of deep neural networks especially deep convolutional neural networks (CNNs) in pattern recognition within images. In this project, a type of deep CNN architecture, which specializes in real-time object detection in images, known as the YOLO network [2,3] is used to detect pipeline anomalies. YOLO network is a supervised object detection algorithm which first determines whether an object exists in the image, classifies the object according to its category, and then localizes the object using bounding boxes. The YOLO is implemented as a CNN model and its architecture is shown in Fig. 3.

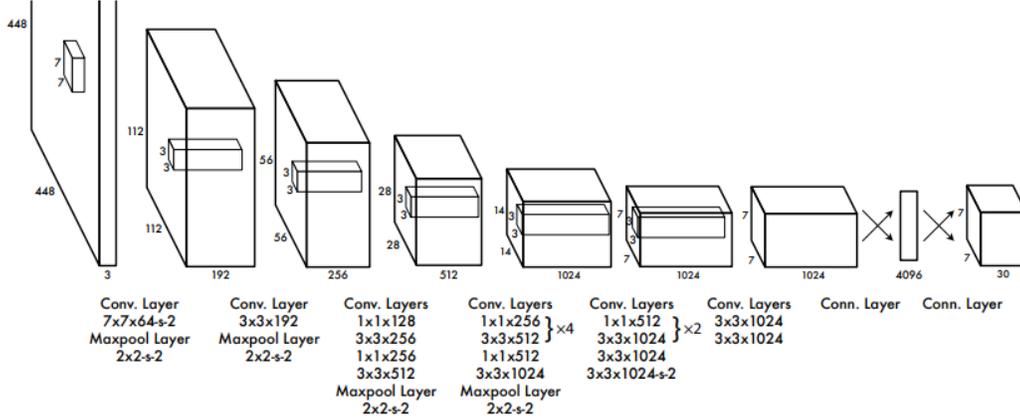


Fig. 3 The architecture of YOLO network[2]

The YOLO network has been widely used to detect common objects in street views such as person, cars, traffic lights, etc. For applications to detecting pipeline anomalies such as cracks and pits, the YOLO network needs to be trained using labeled data consisting of these anomalies. In the proposed study, we will use a popular method in deep learning known as the transfer learning. The idea is that prior to the application for pipeline anomaly detection, the YOLO network is pre-trained using a huge volume of image data which are not associated with the task of pipeline anomaly detection. The reason for this is that pipeline anomalies images are relatively scarce. However, there are large amount of labeled data for other tasks such as autonomous driving. With a large enough training set, the features extracted by the pre-trained network can be generalized to other, possibly very different problems, with a far lower number of training examples specific to the problem to be solved.

The objective of the research in the 1st quarter is to: (1) develop stereo vision algorithm that is able to generate real-time depth map of the pipeline inner surface; (2) design and assemble of prototype device parts including the camera housing module and the robot carrier; (3) conduct preliminary demonstration using the developed hardware and software; (4) conduct literature review on AI-based methods for anomaly detection and explore suitable methods for pipeline threats detection using depth map information.

## 2. Task 1: Development of A Novel Multi-Camera Stereo Vision System for Pipeline Inline Inspection

### 2.1 Stereo Vision Algorithm Development

#### 2.1.1 Stereo Camera Calibration

In order to extract 3D information using stereo vision described above, we first need to be able to map 3D points in the real world to 2D points in an image plane. This mapping can be represented in mathematical form as:

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = K[R|T] \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \quad (3)$$

$$K = \begin{bmatrix} fx & 0 & cx \\ 0 & fy & cy \\ 0 & 0 & 1 \end{bmatrix}$$

where  $K$  is the 3-by-3 camera matrix comprised of intrinsic parameters including the focal length and principle point;  $R$  and  $T$  are the 3-by-3 rotation matrix and 3-by-1 translation vector (extrinsic parameters), respectively;  $u, v$  are the coordinates of the 2D points in the image plane, and  $X, Y, Z$  is the coordinates of the 3D points in real world. The goal of camera calibration is to find the camera parameters including the intrinsic and extrinsic parameters that will us to perform the mapping given in Eq. (3).

In this study, we adopt the multiplane calibration method to compute the camera parameters by solving a homogeneous system of linear equations. Specifically, this involves placing a chessboard (a single planar surface) at multiple views and finding object points of the chessboard, i.e. the edges and corners and each square block in the chessboard. Fig. 5 shows the 3D representation the different positions at which the chessboard was placed. The procedures for camera calibration are given below. In this study, we used available functions from a library of programming functions mainly aimed at real-time computer vision known as OpenCV[4] to develop our stereo vision algorithm for 3D reconstruction of pipeline inner surface.

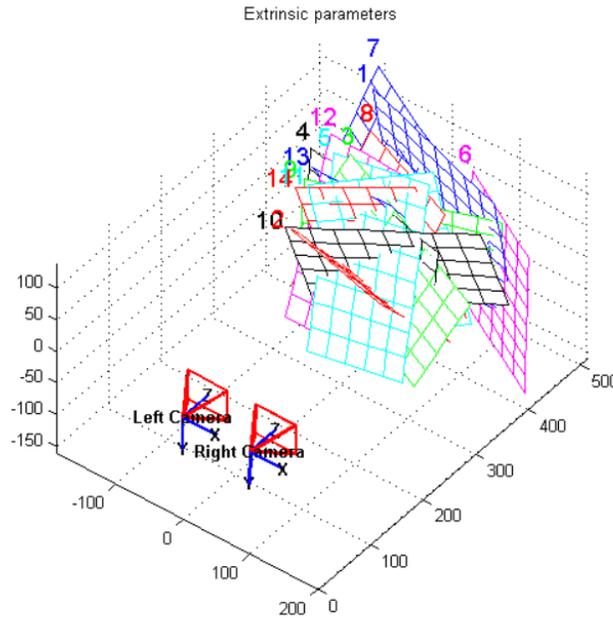


Fig. 5 3D representation of Calibration[5]

**Procedures for camera calibration of the camera**

- The set of images taken from each camera were fed to the algorithm.
- The size of the chess board was given so that it could detect the object points that is the dimension of each square and the corner points for the total area of the chessboard.
- The images were then converted to grayscale and the corners of the chessboard were found.
- The intrinsic and extrinsic parameters were found using the function `cv2.calibrateCamera(objectPoints, imagePoints, imageSize[, cameraMatrix[, distCoeffs[, rvecs[, tvecs[, flags[, criteria]]]]) → retval, cameraMatrix, distCoeffs, rvecs, tvecs` (from OpenCV).

The code used to achieve the calibration is given below:

```
#=====
# Camera calibration
#=====
#Define size of chessboard target.
```

```

chessboard_size = (7,5)
#read images
calibration_paths = glob.glob('./calibration_images/*')
#Define arrays to save detected points
obj_points = [] #3D points in real world space
img_points = [] #3D points in image plane
#Prepare grid and points to display
objp = np.zeros((np.prod(chessboard_size),3),dtype=np.float32)
objp[:,2] = np.mgrid[0:chessboard_size[0], 0:chessboard_size[1]].T.reshape(-1,2)
#Iterate over images to find intrinsic matrix
for image_path in tqdm(calibration_paths):
#Load image
image = cv2.imread(image_path)
gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
print("Image loaded, Analyzing...")
#find chessboard corners
ret, corners = cv2.findChessboardCorners(gray_image, chessboard_size, None)
if ret == True:
print("Chessboard detected!")
print(image_path)
# define criteria for subpixel accuracy
criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 30, 0.001)
# refine corner location (to subpixel accuracy) based on criteria.
cv2.cornerSubPix(gray_image, corners, (5, 5), (-1, -1), criteria)
obj_points.append(objp)
img_points.append(corners)
# Calibrate camera
ret, K, dist, rvecs, tvecs = cv2.calibrateCamera(obj_points, img_points, gray_image.shape[:-1], None,
None)
# Save parameters into numpy file
np.save("./camera_params/ret", ret)
np.save("./camera_params/K", K)
np.save("./camera_params/dist", dist)
np.save("./camera_params/rvecs", rvecs)
np.save("./camera_params/tvecs", tvecs)

```

Using the procedures described above, the parameters of the cameras used in this project were found and given below:

- Distortion coefficients  $[-6.04048165e-01, 7.85972333e+00, -5.99917447e-03, 3.84618272e-02, -4.24688166e+01]$
- Rotation vector  $[-0.16102839], [-0.41509481], [1.33232098]$
- Translation vector  $[0.43676107], [-3.56954998], [11.78469181]$
- The camera matrix was found to be 
$$\begin{bmatrix} 514.944 & 0 & 251.17 \\ 0 & 493.94811 & 251.17 \\ 0 & 0 & 01 \end{bmatrix}$$

It should be noted that since we adopted fisheye cameras in order to enable wide angle of view, we need to account for the radial and tangential lens distortion to obtain the undistorted images. The distortion coefficients given above will be used later to undistort the image.

## 2.1.2 Stereo Image Rectification

If the optical axes of the two adjacent cameras in the stereo vision system are not aligned to each other, the epipolar lines are not parallel in the two image planes. In this case, computing stereo correspondence is a 2D search problem[6]. In practice, image rectification is usually performed to simplify the stereo correspondence problem. Stereo image rectification projects images onto a common image plane which makes all the epipolar lines parallel and the corresponding points having the same row coordinates. By doing this, computing stereo correspondence is reduced to a 1D search problem. The coordinates in the rectified image  $u_l$  and  $u_r$  are obtained by rotating the rays about the camera centers, and then applying a pinhole projection using the left and right camera matrices  $K_l$  and  $K_r$ [7]:

$$u_l = K_l * (\tilde{R}_l * X_l) \quad (4)$$

$$u_r = K_r * (\tilde{R}_r * X_r) \quad (5)$$

$$K_l = \begin{bmatrix} f & 0 & u_{o_l} \\ 0 & f & v_{o_l} \\ 0 & 0 & 1 \end{bmatrix} \quad K_r = \begin{bmatrix} f & 0 & u_{o_r} \\ 0 & f & v_{o_r} \\ 0 & 0 & 1 \end{bmatrix}$$

where  $\tilde{R}_l$  and  $\tilde{R}_r$  are the rotation matrix used in rectification,  $u_{o_l}$  and  $v_{o_l}$  is the coordinate of the principal point in the left image, and  $u_{o_r}$  and  $v_{o_r}$  is the coordinate of the principal point in the right image.

Given below is the code for rectification:

```
# Rectification of images #
def plot_rectified_images(self, feat_mode="SURF"):
    """Plots rectified images
        This method computes and plots a rectified version of the two
        images side by side.

        :param feat_mode: whether to use rich descriptors for feature
            matching ("surf") or optic flow ("flow")
    """
    self._extract_keypoints(feat_mode)
    self._find_fundamental_matrix()
    self._find_essential_matrix()
    self._find_camera_matrices_rt()

    R = self.Rt2[:, :3]
    T = self.Rt2[:, 3]
    # perform the rectification
    R1, R2, P1, P2, Q, roi1, roi2 = cv2.stereoRectify(self.K, self.d, self.K, self.d,
self.img1.shape[:2], R, T, alpha=1.0)
    mapx1, mapy1 = cv2.initUndistortRectifyMap(self.K, self.d,
R1, self.K, self.img1.shape[:2], cv2.CV_32F)
    mapx2, mapy2 = cv2.initUndistortRectifyMap(self.K, self.d,
R2, self.K, self.img2.shape[:2], cv2.CV_32F)
    img_rect1 = cv2.remap(self.img1, mapx1, mapy1, cv2.INTER_LINEAR)
    img_rect2 = cv2.remap(self.img2, mapx2, mapy2, cv2.INTER_LINEAR)

    # draw the images side by side
    total_size = (max(img_rect1.shape[0], img_rect2.shape[0]),
img_rect1.shape[1] + img_rect2.shape[1], 3)
    img = np.zeros(total_size, dtype=np.uint8)
    img[:img_rect1.shape[0], :img_rect1.shape[1]] = img_rect1
```

```
img[:img_rect2.shape[0], img_rect1.shape[1]:] = img_rect2

# draw horizontal lines every 25 px across the side by side image
for i in range(20, img.shape[0], 25):
    cv2.line(img, (0, i), (img.shape[1], i), (255, 0, 0))

cv2.imshow('imgRectified', img)
```

### Procedures for stereo image rectification:

- From the above function the rotation vector and translation vector were extracted and stored in variable 'R' and 'T'.
- Each point in the right image and left image was mapped in X and Y coordinates.
- Each image was undistorted and rectified using the function `cv2.initUndistortRectifyMap(self.K, self.d, R1, self.K, self.img1.shape[:2], cv2.CV_32F)` (from OpenCV) where K is the Camera Matrix, d is the distortion coefficients vector, img is the image from each camera, and shape is the points in 2-D space.
- The rotations used in the rectification,  $\tilde{R}_l$  and  $\tilde{R}_r$ , rotate the cameras' principal axes so that they are orthogonal to the vector joining the camera centers (i.e. the baseline), and the epipoles in the rectified images are horizontally aligned.
- Once the images were rectified they were stored in variable `img_rect1` and `img_rect2`

### 2.1.3 Disparity Map Generation

In disparity map generation the depth between the objects is seen in terms of pixel density. In our experiment, the function `cv2.stereoSGBM` from OpenCV will give values to each pixel that it sees and the difference in the depth will be the disparity found by these pixels. Depth information was computed from a pair of stereo images by first computing the distance in pixels between the location of a feature in one image and its location in the other image. The reason for this is that pixels with larger disparities are closer to the camera, and pixels with smaller disparities are farther from the camera. Essentially, we'll be taking a small region of pixels in the right image, and searching for the closest matching region of pixels in the left image. When searching the right image, we'll start at the same coordinates and search the left and right up to some maximum distance. This search is conducted in one dimension to save time thanks to the rectification performed before. After stereo correspondence is determined, the disparity between each pixel can be computed and the disparity map is generated.

Given below is the code for real-time disparity map generation:

```
# Disparity Generation #
# Create a VideoCapture object and read from input file
# If the input is the camera, pass 0 instead of the video file name
cap1 = cv2.VideoCapture(1)
cap2 = cv2.VideoCapture(2)

# Check if camera opened successfully
if (cap1.isOpened() == False | cap2.isOpened() == False):
    print("Error opening video stream or file")

# Read until video is completed
while (1):
    # Capture frame-by-frame
    ret1, imgL = cap1.read()
```

```

ret2, imgR = cap2.read()

if (ret1 == True & ret2 == True):
    window_size = 8
    left_matcher = cv2.StereoSGBM_create(
        minDisparity=0,
        numDisparities=256, # max_disp has to be dividable by 16 f. E. HH 192, 256
        blockSize=8,
        P1=8 * 3 * window_size ** 2,
        P2=16 * 3 * window_size ** 2,
        disp12MaxDiff=1,
        uniquenessRatio=15,
        speckleWindowSize=200,
        speckleRange=2,
        preFilterCap=63,
        mode=cv2.STEREO_SGBM_MODE_SGBM_3WAY
    )

    right_matcher = cv2.ximgproc.createRightMatcher(left_matcher)

    # FILTER Parameters
    lmbda = 90000
    sigma = 1.2
    visual_multiplier = 1.5

    wls_filter = cv2.ximgproc.createDisparityWLSFilter(matcher_left=left_matcher)
    wls_filter.setLambda(lmbda)
    wls_filter.setSigmaColor(sigma)

    print('computing disparity...')
    displ=left_matcher.compute(imgL,imgR)#.astype(np.float32)/16
    dispr=right_matcher.compute(imgR,imgL)#.astype(np.float32)/16
    displ = np.int32(displ)
    dispr = np.int32(dispr)
    filteredImg=wls_filter.filter(displ,imgL,imgR,dispr)#important to put "imgL" here!!!

    filteredImg = cv2.normalize(src=filteredImg, dst=filteredImg,beta=0,alpha=255,
    norm_type=cv2.NORM_MINMAX);
    filteredImg = np.uint8(filteredImg)
    ret, thresh1 = cv2.threshold(filteredImg, 255, 255, cv2.THRESH_BINARY)

    # Display the resulting frame
    cv2.imshow('Disparity Map', filteredImg)
    # cv2.imshow('Frame1',thresh1)
    # cv2.imshow('Frame2',imgR)
    # Press Q on keyboard to exit
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

# Break the loop

```



## 2.2.2 Robot Carrier Development

For the robot carrier, we selected a commercial eight degrees of freedom (DOF) vehicle robot with 8-Axis RC robotic arm. The specification for this robot carrier is given below:

Y100 Tank chassis parameters:

- Name: New Y-100 Tank Chassis
- Main body: aluminum alloy
- Surface: sandblasting oxidation
- Track: engineering plastics
- Color: Silver
- Size: 300\*240\*122mm (length \* width \* height)
- Weight: 1.16kg
- Motor: 9V 150rpm with encoder motor

9V 150rpm with encoder motor:

- Working voltage: 9V
- Output rate: 150±10% rpm
- Load current: 200mA (Max)
- Stall current: 4500 mA (max)
- Stalling torque: 9.5kg NaN
- Load speed: 100±10% rpm
- Load torque: 3000gNaN
- Load current: 1200mA (Max)
- Encode parameters: 2 pulses / circle
- Sensor operating voltage: 3-5V

Robotic arm parameters:

- Mechanical arm body material: aluminum alloy
- Weight: 0.82kg (including servo weight)
- Color: silver
- Servo optional: MG996R metal gear large servo
- Features: The mechanical arm can be equipped with a mechanical gripper, the weight of the grip is about 500g; the base of the robot arm can be rotated 360 degrees.

MG996R servo parameters:

- Name: MG996R
- Net weight: 55 g
- Size: 40.7\*19.7\*42.9 mm
- Pulling force: 9.4 kg/cm (4.8 V), 11 kg/cm (6 V)
- Reaction rate: 0.17 sec / 60 degree (4.8 V), 0.14 sec / 60 degree (6 V)
- Working voltage: 4.8-7.2 V
- Working temperature: 0°C-55°C
- Gear form: metal gear
- Working dead zone: 5 us (microseconds)

### *Assembly of Tank*

#### 1. Bearing wheel installation

The materials required to assemble the bearing wheel are M3\*8 screw, 17 mm copper hex spacers, M2 screw, bearings, wheel discs, and stainless-steel connector.

## 2. Driving wheel installation

For driving wheel, the required materials are 28 mm copper hex screws, M3\*8 screw, stainless-steel connector, jackscrew, geared wheel discs, aluminium alloy coupling, and M4\*16 screw.

## 3. Tank Chassis installation

To assemble the Tank Chassis, the required materials include 9V 150 rpm encoder motor, LED lights, tracks, chassis panels, power cable, M3\*12 screw, M3 nut, M3\*10 screw are necessary for assembly of tank.

Fig. 7 shows the assembled tank chassis of the robot carrier. There are a total of ten bearing wheels and two driving wheels for this model. The track is installed and since the tracks are individually connected with needles, the length can be adjusted according to the needs.



Fig. 7 Vehicle Tank Assembly

## ***Assembly of Robot arm***

The robot arm consists of a rotating base, left and right swing arm, a rocking arm, a control board. Fig. 8 shows the assembled robotic arm of the robot carrier.

### 1. Rotating Base

The materials required are M4\*11 double pass coupling, M4\*6 flat head screws, M3\*8 inner hexagon screw, M3 nut, tray bearing, single pass coupling, 25T disc metal horns, and steering gear.

### 2. Left and Right Swing Arm

M3\*8 inner hexagon screws, 25T metal horn, M4\*6 screws, pendulum frame, M3\*5 screws, steering gear, left and right frames are assembled together. And this assembly is installed on the rotating base.

### 3. Rocking arm

M3\*8 inner hexagon screws, U bracket, 25T metal horn, M3\*8 flat head screw, steering gear, M4\*6 screws, steering gear bracket, swing fixing bracket, rocking fixing bracket, U shaped support frame, connecting rod are installed together according to the manual. This assembled part is installed to the previously connected servo arm with rotating base.

### 4. Control Board

The control board tray bracket is attached to the rotating base. Materials required are M3\*8 screws, M3\*6 screws, steering gear bracket. Bearing is attached to steering gear bracket along with a steering gear.

### 5. Metal claw

The assembly of the claw is divided into 3 parts, which are claw arm 1, claw arm 2, and base of claw. The parts include servo motor MG995, 25T metal horn, M3\*8 hexagon screw, M3 nut, single pass couplings, M3\*6 flat head screw, M3\*12 screw, and bearings.

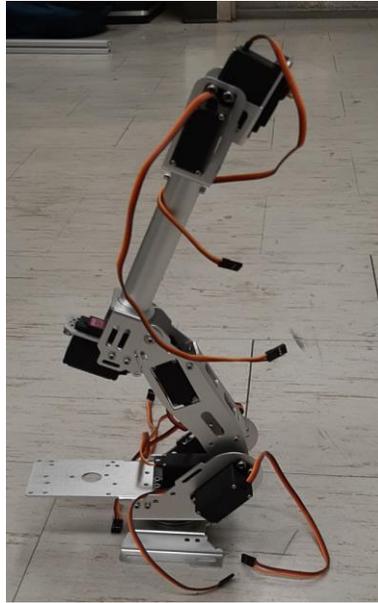


Fig. 8 Robot arm assembly

The robotic arm and the tank are then assembled together, and the final product is shown in Fig. 9.

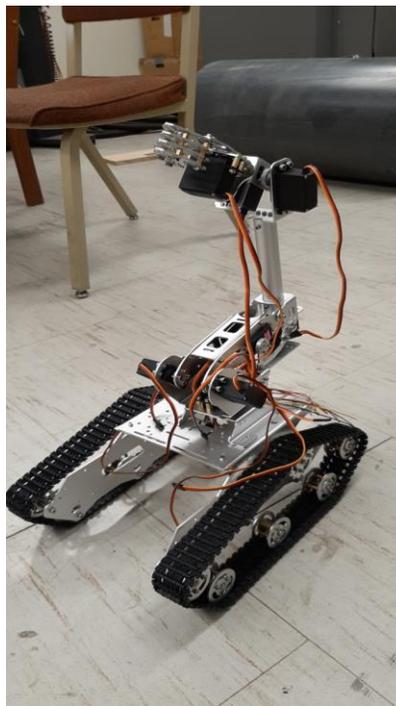


Fig. 9 Assembled robot carrier

### 2.2.3 Preliminary Demonstration

For this quarter, a preliminary demonstration was done to see whether we are able to generate a clear depth map which will be used to highlight different features in the inner surface of the pipe using the developed stereo vision algorithm and prototype device. For this purpose, some artificial defects were simulated

inside of a representative pipe component and the developed device is used to generate a depth map of the inner surface of the pipe. This depth map will help us locate irregularities on the surface such as cracks and corrosion pits.

**a) Hardware used:**

**1) Fish-eye Stereo Camera (Fig. 10):**

Specifications:

Performance: 1920x1080 MJPEG@30fps	S/N Ratio: 39db
Sensor: AR0330	Dynamic range: 72.4db
Pixel Size: 5.07um X 3.38um	
Lens Parameter: M12 fisheye lens	
Board size: 86x23 mm	
Mini illumination: 0.1lux	
Sensitivity: <a href="#">2.0 v/lux-sec @550nm</a>	
Voltage: DC5V	
Current: 220mA-280mA	



Fig. 10 Fisheye Stereo Camera[8]

**2) 3-D Printed Camera Housing Module (Fig. 11)**

The camera housing module designed above was manufactured using a 3D printer. For the purpose of demonstration, the camera housing module carried one pair of fisheye cameras. Each side of the module has two slopes: one for keeping the camera and the other for attaching the LED lights. Fig. 11 shows the printed camera housing module with one pair of fisheye stereo cameras installed.

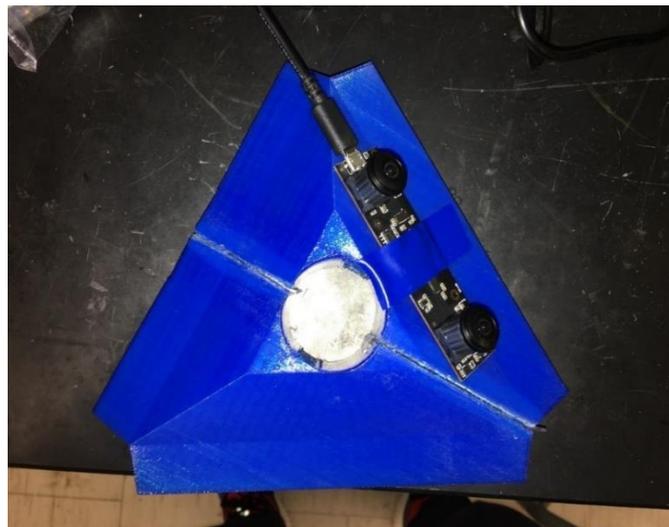


Fig. 11 Camera Housing Module

**b) Experiment setup:**

For the purpose of demonstration, the camera housing module was elevated to the axis center of the pipe and a ruler was placed at the end which was to be observed to show different contours. The surface also had an irregularity to simulate what rust would look like. This was done to demonstrate that the depth-map generated could identify both contour types, i.e. the crack propagating inwards and rust protruding outwards. Fig. 12 shows the experimental setup.

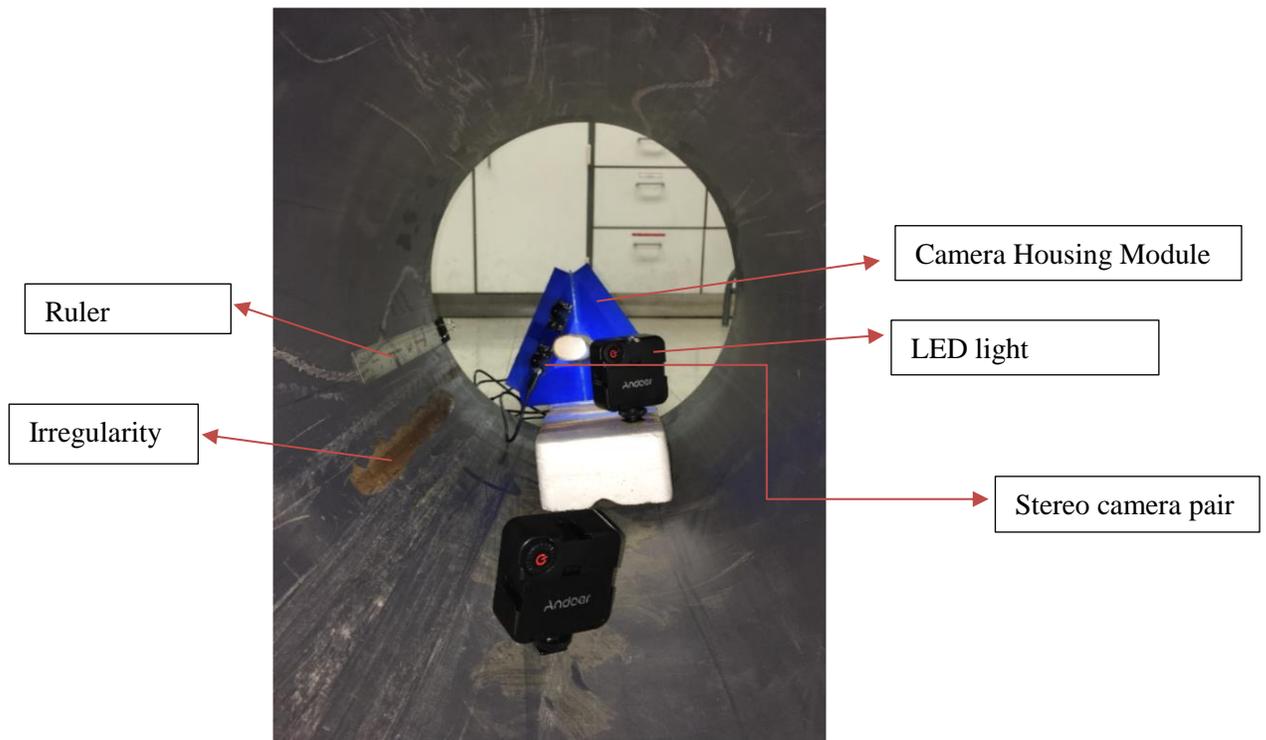


Fig. 12 Experimental Setup

**Results:**

Once the setup was complete, the stereo vision code developed before was run and the results were generated. The original image of the pipe taken before the depth map is shown in Fig. 13. This shows the two types of contours i.e. the ruler and the surface irregularity. It should be noted that this image is distorted around the edges as a fisheye camera was used.

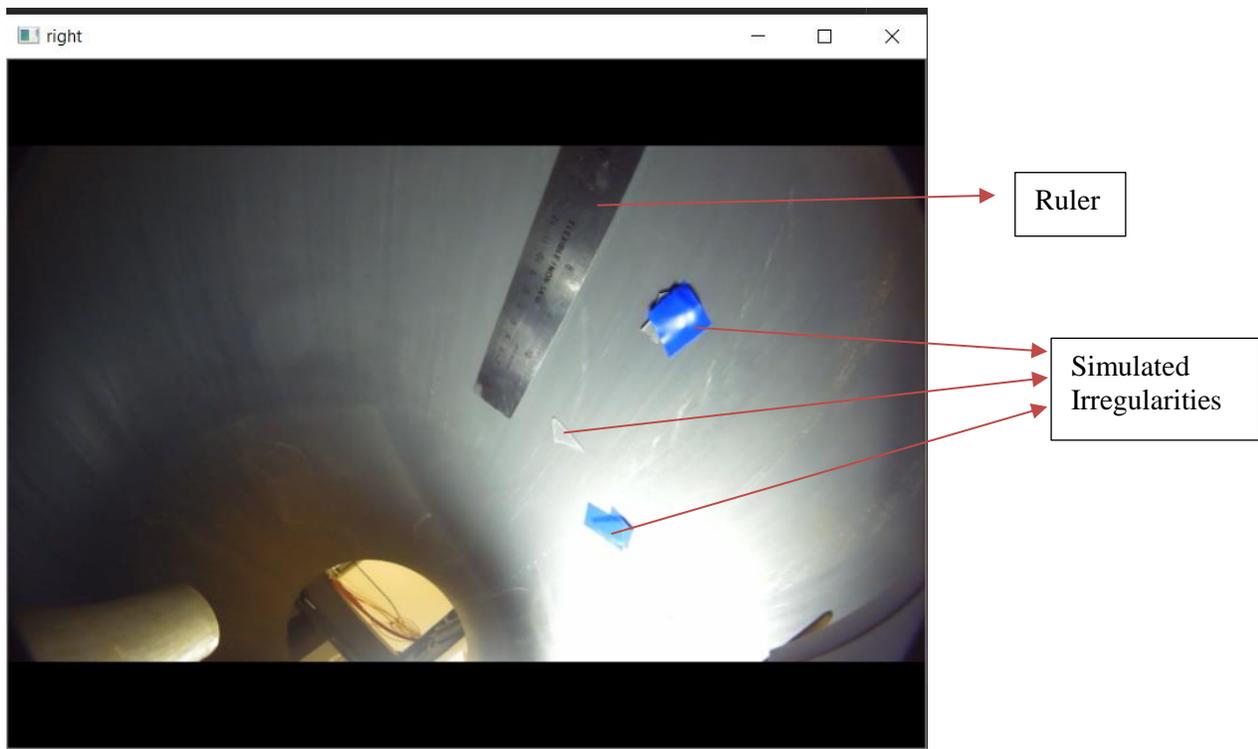


Fig. 13 Preliminary Image

The depth map was generated and shown in Fig. 14. It can be seen that the contours of the ruler and two surface irregularities can be clearly seen from the depth map. The ruler was placed to simulated protruding irregularities in the pipe and its edges worked as cracks to show the depth in the cracks. The other irregularity shown worked as simulated rust in the pipe. In addition, it can be seen that the depth map generated is sensitive to the lighting condition. In this experiment, the LED lights are not placed on the prototype device but in future experiment, the LED light panels will be attached to the prototype to enable better light conditions and thus a clearer depth map.



Fig. 14 Generated depth map

### 3. Task 2: AI-enabled Anomaly Detection and Physics-based Damage Prognostics for Pipeline Integrity Management

#### 3.1 Neural Networks for Threat Detection

Neural Networks (NNs) have shown great promise in the field of object detection and segmentation. Typical Deep Neural Network architectures have parameters to the order of  $\sim 10^7$  and need enough training data to tune these parameters for the task at hand. This section describes relevant ideas surrounding Deep Neural Networks in the context of threat detection in pipelines.

##### 3.1.1 Convolutional Neural Networks

Image data consists of a matrix of pixels and are high-dimensional. Learning from this high dimensional representation requires either feature engineering to reduce the dimensionality of the dataset to a form useful for the task at hand, or to find a computational method that can learn directly from this high-dimensional representation. Convolutional Neural Networks (CNN) are a special type of neural network that can learn to extract features, detect objects, segment objects and perform a myriad of other tasks. Convolutional networks leverage the convolution operation, which is an element-wise dot product between an image or a part of it, with an equal-sized kernel as the window, which consists of the weights to be learned using backpropagation.

##### 3.1.2 Configuring Neural Networks for Learning Tasks

Convolutional Neural Networks can be configured for a variety of learning tasks, including, but not limited to, object detection and image classification. The task that the network is designed to perform depends primarily on the optimization objective specified by the designer of the network, and the form of the training data presented to the network. Fully supervised image classification tasks, for instance, form a category of tasks for CNNs, and are fed raw image examples of each class to be detected, and the optimization objective consists of a simple cross-entropy loss. The objective to be optimized is simple and the data structures that the network uses to trained upon is straight-forward. Object detection and segmentation, on the other hand, have more complex objective functions. Multiple fully-supervised methods have become popular in the literature, with fast, single-pass methods like YOLO[2] providing real-time performance, and region-based methods like R-CNN, Fast R-CNN, Faster R-CNN and Mask R-CNN[9–12] providing slightly better detection metrics at the expense of real-time computational performance in general object detection tasks.

Our work with 2-D imaging has focused heavily on developing new weakly supervised methods for threat detection. Fully supervised networks need lots of training images with bounding boxes or segmentation ground truths, which is expensive to create. Weak supervision, on the other hand, relies on weak or partial labels for training. A formal framework for training with partial labels could look like the following:

Consider having a training data set  $T$  with  $n$ -instances, and  $m < n$  of those instances has labels.

$$T = \{(x_1, y_1) \dots (x_m, y_m), (x_{m+1}), \dots, (x_n, y_n)\} \quad (6)$$

To learn a mapping from this dataset, semi-supervised learning methods can be used. Weakly supervised learning instead leverages weak labels for training. These labels are often cheaper and faster to create, and this enables us to create large, weakly-labelled datasets for training models. Method development and modifications for a weakly-supervised learning model could be at the level of the algorithm architecture or at the level of the algorithmic components, such as optimization objective functions and data transformations.

The previous section presents in detail, a modification of a fully supervised pipeline into a weakly-supervised pipeline. We developed a region-based method for weakly-supervised detection of cracks. In the case of object detection, instead of providing training data with bounding boxes, we provide only textures of what we want the algorithm to find. The objective function for optimization depends on the weakly-supervised method used.

### 3.1.3 Implementation of CNNs for Learning to Detect Cracks in Pipelines

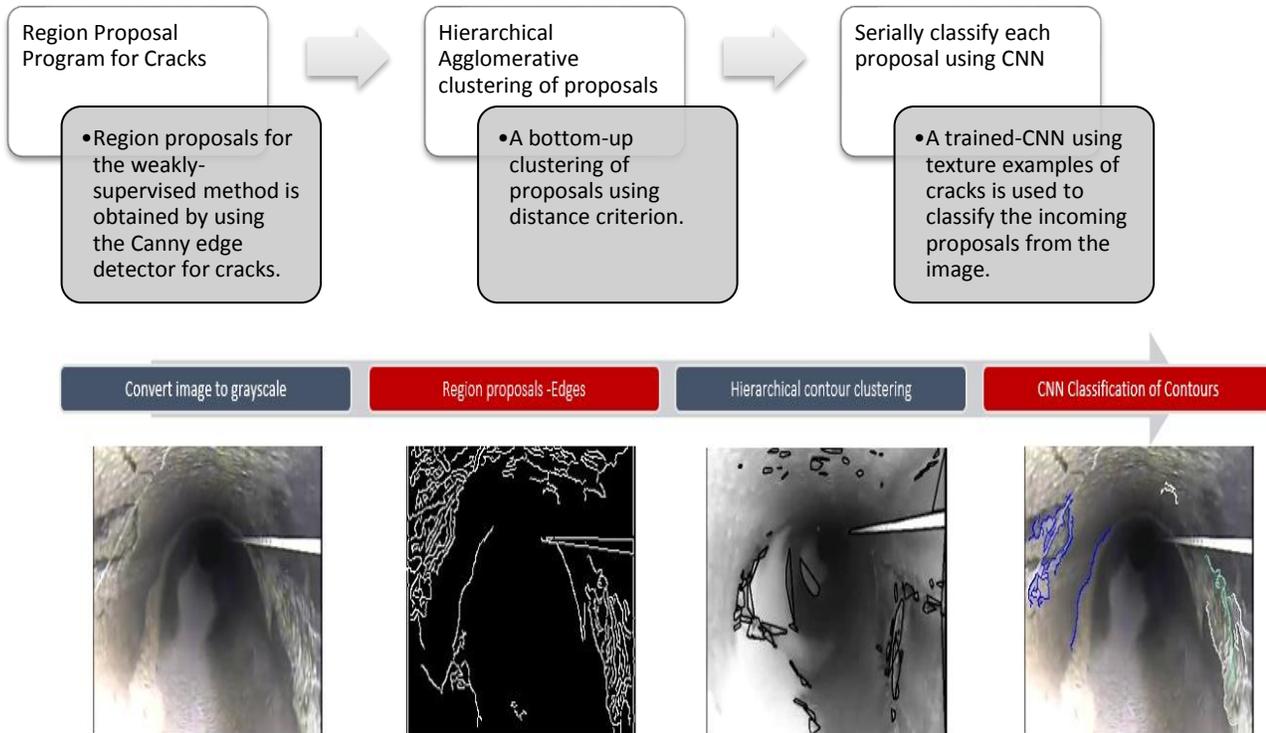


Fig. 15 Overview of implementation of the crack detection method

The above illustration summarizes the implementation of a weakly-supervised learning methodology. The method generalizes well to new lighting conditions and varying textures. The detection method involves defining a crack semantically, using a region extraction method to choose regions that have the highest likelihood of containing cracks, and then proceed to classify these patches. The method essentially reduces the search space for objects using an edge detector to act as a region proposal, and then serially classifies the edges as either belonging to a crack or not. A dataset of crack and non-crack images was procured, and a CNN classifier was trained upon it.

The edge detector that was used to extract edge pixels from the raw image was the Canny edge detector. This is a simple, fast, 2-parameter algorithm that works based on neighboring pixel gradients to yield a binary image of edges. Contours are then extracted from the representation after dilation is performed on the image. Dilation operates on a binary image by increasing the thickness of the extracted contours, removing disparate noisy edge pixels. The extracted contours are usually very large in number and are closely clustered with neighboring contours as disjoint pixel groups, however small, is considered a separate contour. Processing all such regions will be computationally inefficient, and a fast-agglomerative clustering scheme was used to group neighboring clusters together, as described below.

The set of contours obtained using the edge detector are disjoint, noisy and have high variance in contour area and length. Agglomerative clustering methods have been applied for discrete optimization tasks and for obtaining statistical metrics for numerical data. These methods do not require the number of clusters to

be prespecified, but the clusters emerge as a result of the distance metric used to form them. In this case, it is natural to use the L2-distance between contour centroids as the distance metric. Consider a set of  $S$  contours with centroids  $[c_{xi}, c_{yi}]$ . We first create a symmetric distance matrix  $d$  where  $d_{ij}=d_{ji}$  using the contour centroids.

$$d \in R^{S \times S} \text{ where } d_{ii} = 0 \wedge d_{ij} = d_{ji} \forall x, y \in S \quad (7)$$

where  $d$  is the distance matrix, and  $S$  is the set of contours.

The algorithm works by combining contours in a bottom-up fashion, first starting with the closest set of contours using the distance matrix, and progressively combining these contours until the matrix has no entries below the distance threshold parameter, using the single linkage formula, the simplest bottom-up method to combine clusters. The algorithm works at approximately  $O(N^2)$  and the number of initial contours is usually below 500 per image.

The contour set obtained from the hierarchical agglomerative clustering algorithm is used as the region proposal for the Convolutional Neural Network. The network used in our studies was the Inception v3 network, however, any other architecture could be substituted with this. The network with pretrained weights from the ImageNet database was used for training the network. Earlier layers in the CNN learn to recognize lower-level features such as edges and basic geometric shapes using pretraining, thereby reducing the data requirement when fine-tuning with the relevant dataset for the crack detection application. The trained network learns representations for crack and good regions, using textures as shown in the figure.

The training set consisted of 1000 images, equally distributed between crack and good texture samples. The test set consisted of 80 images, with ground truth segmentations. The performance metrics considered were the percentage of pixels the method was able to segment, which is called the pixel coverage, and the number of false positive segments in the image. The CNN takes as input region proposals, and outputs only those regions that are classified as a crack. The ground truth consists of a binary image, with white pixels corresponding to those regions with cracks, and the black pixels corresponding to the background region. In the pixel coverage metric, we calculate what proportion of pixels from the ground truth are covered by the regions that the network classifies as a crack. The calculations for all metrics are done using the minimal area bounding rectangle of the contour. Another metric that is considered is the number of false positive contours, and this is calculated as the number of contours that do not cover any ground truth pixel.

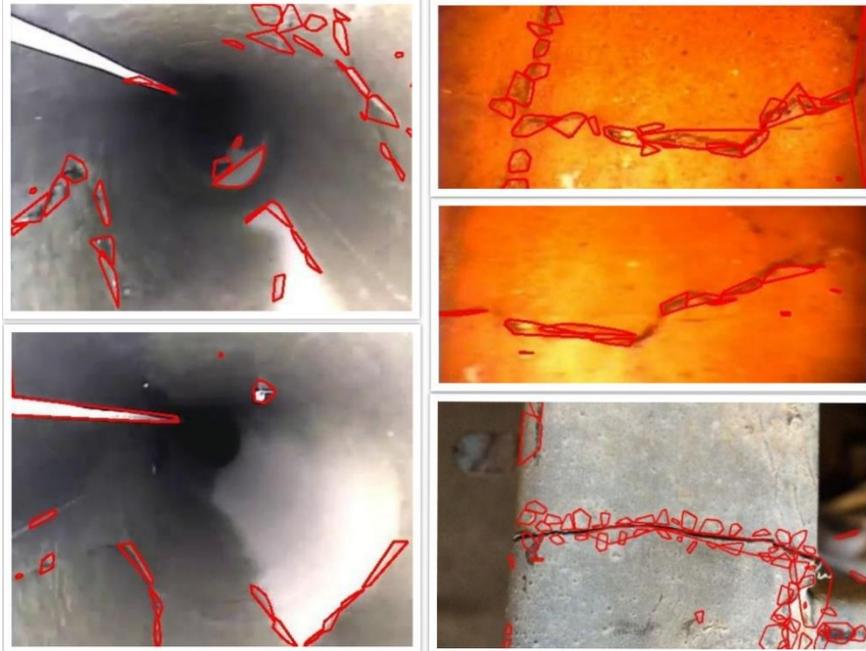
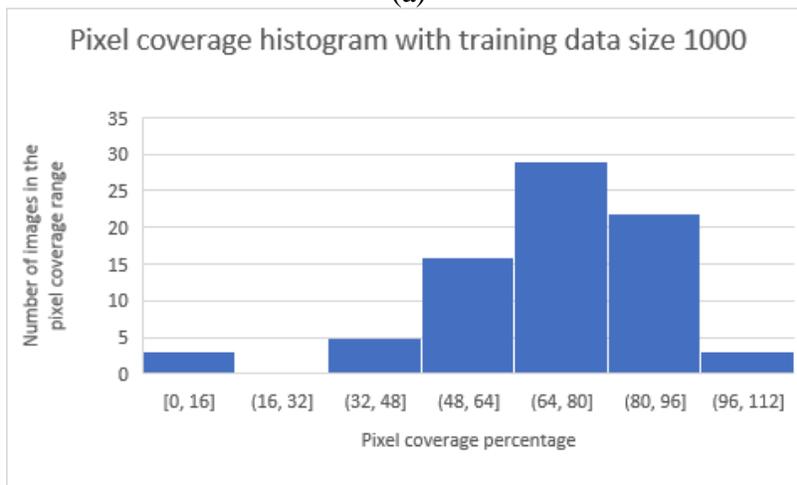


Fig. 16 Samples of detected contours by the model from the test set. While the three images on the right cover all the cracks in the image with hardly any false detection, the two images on the left cover all the cracks, but also generate false positives

Metric	Value
Average number of false positive contours in the test set	5.65
Average pixel coverage wrt ground truth	61.65%

(a)



(b)

Fig. 17: (a) Table summarizing the highlights of the metrics used for evaluation on a 1000 image training set (b) Pixel coverage histogram for testing on 80 images

### 3.2 Depth Information for Enhanced Threat Analysis

Fusing multiple sources of data helps us obtain a more complete picture of all the variables involved in a problem. In the case of threat analysis in pipelines using vision-based inspection techniques, depth information can potentially provide a new layer of information which can be leveraged for unique insights in the condition of pipeline systems. Stereovision algorithms enable us to use multi-camera setup to obtain multiple images, that can then be used to compute a disparity map between these images. We can use the disparity maps to create a depth map of the objects in the scene. In this section, we review the ways in which this additional per-pixel depth data can be used to enhance the data obtained from the 2-D images.

Scene segmentation using both color and depth information makes the process similar to the human visual system, where the disparity map from the images obtained from the two eyes is a dimension of information for scene recognition. Hierarchical fusion methods such as what is proposed by Caldereo et al. [13] in a hierarchical, iterative cooperative region merging scheme as shown in Fig. 18.

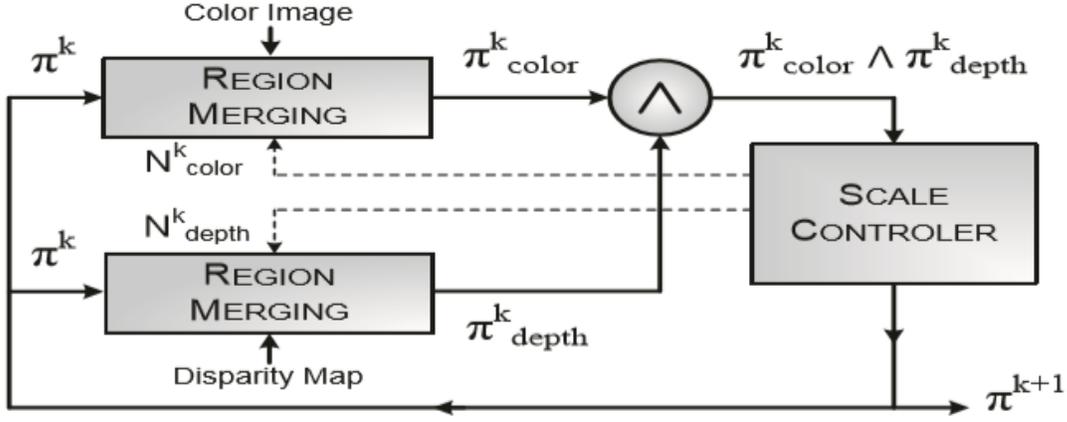


Fig. 18: Cooperative Region Merging Scheme for Information Fusion of RGB Image and Disparity Map[13]

Let  $\pi$  be an image partition, which is a division of the image into non-empty, disjoint sets. Let  $\Pi$  be the set of all possible partitions. The set  $\Pi$  is ordered by the refinement order, that is, the number of regions in each partition  $\pi_i$  in  $\Pi$ . Suppose we have a set of partitions, or profile given by  $\pi_p = \{\pi_1, \dots, \pi_n\}$  a median partition of the profile, denoted by  $\mu$  is defined as the partition minimizing the function:

$$f(\pi) = \sum_{\substack{1 \leq i \leq n \\ \pi_i \in \pi_p}} \delta(\pi, \pi_i) \quad (8)$$

The function  $\delta$  is the symmetric difference between the partitions  $\pi$  and  $\pi_i$ , that is, the minimum number of pixels whose region labels must change for both partitions to be identical. The median partition of the profile is defined as:

$$\mu = \arg \min_{\pi} f(\pi) \quad (9)$$

The median partition obeys the pareto principle: If  $\mu$  is a median partition of the profile  $\pi_p$ :

$$\bigwedge_{1 \leq i \leq n} \pi_i \leq \mu \quad (10)$$

i.e., The median partition is coarser than the supremum of all the partitions in the profile.

The initial conditions for the iterative procedure accept an initial number of regions, which can be set to the number of pixels in the color image and the depth image,  $N_{color}^0 = N_{color}^{MAX}$  and  $N_{depth}^0 = N_{depth}^{MAX}$  and initial partitions  $\pi_{color}^0$  and  $\pi_{depth}^0$ . Starting from this initial partition, the region merging step iterates until it obtains an output partition using information theoretic Bhattacharya merging criterion[14]. These bottom-up segmentation approaches are based on information at a local scale.

Region merging algorithms are specified by: i) a merging criterion, defining the cost of merging two regions, and ii) a merging order, determining the sequence in which regions are merged. This technique is unsupervised and uses statistical measures to make merging decisions. From a statistical standpoint, an image can be thought of as a matrix of iid pixel samples that are formed by a 2D-stochastic process. For a grey-level image, the samples can take a value from  $X = \{0, 1, \dots, 255\}$ . The empirical distribution  $P_x$  is defined as the relative proportion of occurrences of each value of  $X$ .

$$P_x(a) = \frac{N(a|x)}{n} \quad (11)$$

$$a \in X$$

$$x \in X^n$$

The probability of formation of a given sequence of iid observations  $x$  with probability distribution  $Q$ , is

given by:

$$\begin{aligned}
 Q^n(x) &= 2^{-n(H(P_x)+D(P_x||Q))} & (12) \\
 H(P(x)) &= -\sum_{a \in X} P(a) \log(P(a)) \\
 D(P_x||Q) &= \sum P_x(a) \log\left[\frac{P_x(a)}{Q(a)}\right] \\
 H &- \text{Shannon Entropy of the sequence} \\
 D &- \text{KL Divergence between P and Q}
 \end{aligned}$$

For a sufficiently large sample  $n$ , the KL divergence approaches 1, and probability of formation for a given sequence can be approximated as:

$$Q^n(x) \approx 2^{-nH(P_x)} \quad (13)$$

The Bhattacharya merging criterion is derived by considering an  $(n-1)$ -dimensional manifold defined by all possible empirical distributions for a sequence of  $n$ -samples. Chernoff information between the statistical distribution of any pair of classes is,

$$C(P_i, P_j) \triangleq -\min_{0 \leq \lambda \leq 1} \log(\sum_x P_i^\lambda(x) P_j^{1-\lambda}(x)) \quad (14)$$

This optimization problem merges adjacent regions with the maximum probability of fusion.

Once the regions are merged, the new partitions  $\pi^1$  from both the color and depth images, are passed into the information fusion step, where the greatest lower bound (supremum) of the two partitions is calculated. The scale controller block consists of a scale-based filter, and a scale adapter. These adjust the number of regions by ensuring that very small regions do not form. The iterations complete when the partitions do not change, and the hierarchy has reached the coarsest possible segmentation, yielding results with various segmentations corresponding to different coarseness levels.

## 4. Future Works

### 4.1 Task 1

In the next quarter, we will work on getting all the three pairs of stereo cameras onto the camera housing module and perform image stitching to generate a 360-degree view of the inner surface of the pipe. The obtained 360-degree reconstruction of the inner surface will be verified by matching the grid drawn inside of the pipeline. Using this device, we will conduct more sophisticated demonstration with more realistic simulation of pipeline defects. Furthermore, for the demonstration of next quarter, we will use the robotic vehicle assembled to carry the prototype to travel inside of the pipeline to simulate the pipeline inline inspection process. Currently, the basic components of the robotic vehicle have been assembled. In the next quarter, we will achieve the manual control of the robotic vehicle by attaching the control circuit and allow the robot to be controlled with a computer (wireless control) or handle control using a Joystick.

### 4.2 Task 2

In this quarter, we conducted literature review on an unsupervised technique for information fusion and segmentation of RGB-d images. The work in the next couple of quarters would focus on expanding on these techniques, both unsupervised and supervised, using benchmark datasets, to arrive at novel information fusion techniques for general RGB-d segmentation tasks, and also apply these techniques to the pipeline threat detection dataset, that we would collect in-house for task-specific performance.

## References

- [1] University of Washington. Lecture 16: Stereo and 3D Vision of CSE455: Computer Vision. 2018.
- [2] Redmon J, Divvala S, Girshick R, Farhadi A. You Only Look Once: Unified, Real-Time Object Detection n.d.
- [3] Redmon J, Farhadi A. YOLOv3: An Incremental Improvement 2018.
- [4] OpenCVTeam. OpenCV n.d. <https://opencv.org/>.
- [5] Matlab CCT for. Fifth calibration example - Calibrating a stereo system, stereo image rectification and 3D stereo triangulation n.d. [http://www.vision.caltech.edu/bouguetj/calib\\_doc/htmls/example5.html](http://www.vision.caltech.edu/bouguetj/calib_doc/htmls/example5.html).
- [6] Fusiello A. Tutorial on Rectification of Stereo Images. 1999.
- [7] Hansen P, Alismail H, Browning B, Rander P. Stereo visual odometry for pipe mapping. 2011 IEEE/RSJ Int. Conf. Intell. Robot. Syst., 2011, p. 4020–5. doi:10.1109/IROS.2011.6094911.
- [8] ELP 180 Degree Fisheye 2MP 1080P Stereo Webcam OTG UVC Plug Play Driverless Dual Lens USB Camera Module For 3D Video VR Virtual Reality n.d. <http://www.webcamerausb.com/elp-180-degree-fisheye-2mp-1080p-stereo-webcam-otg-uvc-plug-play-driverless-dual-lens-usb-camera-module-for-3d-video-vr-virtual-reality-p-289.html>.
- [9] S. Ren, K. He, R. Girshick and JS. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. IEEE Trans Pattern Anal Mach Intell 2017;doi: 10.11.
- [10] Girshick R, Donahue J, Darrell T, Malik J. Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation. 2014 IEEE Conf. Comput. Vis. Pattern Recognit., 2014, p. 580–7. doi:10.1109/CVPR.2014.81.
- [11] He K, Gkioxari G, Dollár P, Girshick R. Mask R-CNN. 2017 IEEE Int. Conf. Comput. Vis., 2017, p. 2980–8. doi:10.1109/ICCV.2017.322.
- [12] Girshick R. Fast R-CNN. 2015 IEEE Int. Conf. Comput. Vis., 2015, p. 1440–8. doi:10.1109/ICCV.2015.169.
- [13] Calderero F, Marques F. Hierarchical fusion of color and depth information at partition level by cooperative region merging. 2009 IEEE Int. Conf. Acoust. Speech Signal Process., 2009, p. 973–6. doi:10.1109/ICASSP.2009.4959748.
- [14] Calderero F, Marques F. General region merging approaches based on information theory statistical measures. 2008 15th IEEE Int. Conf. Image Process., 2008, p. 3016–9. doi:10.1109/ICIP.2008.4712430.

## Appendix: Complete Code of the Stereo Vision Algorithm

```
from __future__ import print_function
import cv2
from sklearn.preprocessing import normalize
import argparse
import numpy as np
import glob
from tqdm import tqdm
import PIL.ExifTags
import PIL.Image
import argparse
import sys
import os
#=====
# Camera calibration
#=====
#Define size of chessboard target.
chessboard_size = (7,5)
```

```

#read images
calibration_paths = glob.glob('./calibration_images/*')
#Define arrays to save detected points
obj_points = [] #3D points in real world space
img_points = [] #3D points in image plane
#Prepare grid and points to display
objp = np.zeros((np.prod(chessboard_size),3),dtype=np.float32)
objp[:,2] = np.mgrid[0:chessboard_size[0], 0:chessboard_size[1]].T.reshape(-1,2)
#Iterate over images to find intrinsic matrix
for image_path in tqdm(calibration_paths):
#Load image
image = cv2.imread(image_path)
gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
print("Image loaded, Analyzing...")
#find chessboard corners
ret, corners = cv2.findChessboardCorners(gray_image, chessboard_size, None)
if ret == True:
print("Chessboard detected!")
print(image_path)
# define criteria for subpixel accuracy
criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 30, 0.001)
# refine corner location (to subpixel accuracy) based on criteria.
cv2.cornerSubPix(gray_image, corners, (5, 5), (-1, -1), criteria)
obj_points.append(objp)
img_points.append(corners)
# Calibrate camera
ret, K, dist, rvecs, tvecs = cv2.calibrateCamera(obj_points, img_points, gray_image.shape[:-1], None,
None)
# Save parameters into numpy file
np.save("./camera_params/ret", ret)
np.save("./camera_params/K", K)
np.save("./camera_params/dist", dist)
np.save("./camera_params/rvecs", rvecs)
np.save("./camera_params/tvecs", tvecs)

# Rectification of images #
def plot_rectified_images(self, feat_mode="SURF"):
    """Plots rectified images

    This method computes and plots a rectified version of the two
    images side by side.

    :param feat_mode: whether to use rich descriptors for feature
        matching ("surf") or optic flow ("flow")
    """
    self._extract_keypoints(feat_mode)
    self._find_fundamental_matrix()
    self._find_essential_matrix()
    self._find_camera_matrices_rt()

    R = self.Rt2[:, :3]

```

```

T = self.Rt2[:, 3]
# perform the rectification
R1, R2, P1, P2, Q, roi1, roi2 = cv2.stereoRectify(self.K, self.d,
                                                self.K, self.d,
                                                self.img1.shape[:2],
                                                R, T, alpha=1.0)
mapx1, mapy1 = cv2.initUndistortRectifyMap(self.K, self.d, R1, self.K,
                                           self.img1.shape[:2],
                                           cv2.CV_32F)
mapx2, mapy2 = cv2.initUndistortRectifyMap(self.K, self.d, R2, self.K,
                                           self.img2.shape[:2],
                                           cv2.CV_32F)
img_rect1 = cv2.remap(self.img1, mapx1, mapy1, cv2.INTER_LINEAR)
img_rect2 = cv2.remap(self.img2, mapx2, mapy2, cv2.INTER_LINEAR)

# draw the images side by side
total_size = (max(img_rect1.shape[0], img_rect2.shape[0]),
              img_rect1.shape[1] + img_rect2.shape[1], 3)
img = np.zeros(total_size, dtype=np.uint8)
img[:img_rect1.shape[0], :img_rect1.shape[1]] = img_rect1
img[:img_rect2.shape[0], img_rect1.shape[1]:] = img_rect2

# draw horizontal lines every 25 px across the side by side image
for i in range(20, img.shape[0], 25):
    cv2.line(img, (0, i), (img.shape[1], i), (255, 0, 0))

cv2.imshow('imgRectified', img)

# Create a VideoCapture object and read from input file
# If the input is the camera, pass 0 instead of the video file name
cap1 = cv2.VideoCapture(1)
cap2 = cv2.VideoCapture(2)

# Check if camera opened successfully
if (cap1.isOpened() == False | cap2.isOpened() == False):
    print("Error opening video stream or file")

# Read until video is completed
while (1):
    # Capture frame-by-frame
    ret1, imgL = cap1.read()
    ret2, imgR = cap2.read()

    if (ret1 == True & ret2 == True):
        window_size = 8
        left_matcher = cv2.StereoSGBM_create(
            minDisparity=0,
            numDisparities=256, # max_disp has to be dividable by 16 f. E. HH 192, 256
            blockSize=8,
            P1=8 * 3 * window_size ** 2,

```

```

P2=16 * 3 * window_size ** 2,
disp12MaxDiff=1,
uniquenessRatio=15,
speckleWindowSize=200,
speckleRange=2,
preFilterCap=63,
mode=cv2.STEREO_SGBM_MODE_SGBM_3WAY
)

right_matcher = cv2.ximgproc.createRightMatcher(left_matcher)

# FILTER Parameters
lmbda = 90000
sigma = 1.2
visual_multiplier = 1.5

wls_filter = cv2.ximgproc.createDisparityWLSFilter(matcher_left=left_matcher)
wls_filter.setLambda(lmbda)
wls_filter.setSigmaColor(sigma)

print('computing disparity...')
displ = left_matcher.compute(imgL, imgR) # .astype(np.float32)/16
dispr = right_matcher.compute(imgR, imgL) # .astype(np.float32)/16
displ = np.int32(displ)
dispr = np.int32(dispr)
filteredImg = wls_filter.filter(displ, imgL, imgR, dispr) # important to put "imgL" here!!!

filteredImg = cv2.normalize(src=filteredImg, dst=filteredImg, beta=0, alpha=255,
norm_type=cv2.NORM_MINMAX);
filteredImg = np.uint8(filteredImg)
ret, thresh1 = cv2.threshold(filteredImg, 255, 255, cv2.THRESH_BINARY)

# Display the resulting frame
cv2.imshow('Disparity Map', filteredImg)
# cv2.imshow('Frame1',thresh1)
# cv2.imshow('Frame2',imgR)
# Press Q on keyboard to exit
if cv2.waitKey(1) & 0xFF == ord('q'):
    break

# Break the loop
else:
    break
cap1.release()
cap2.release()
cv2.destroyAllWindows()

```