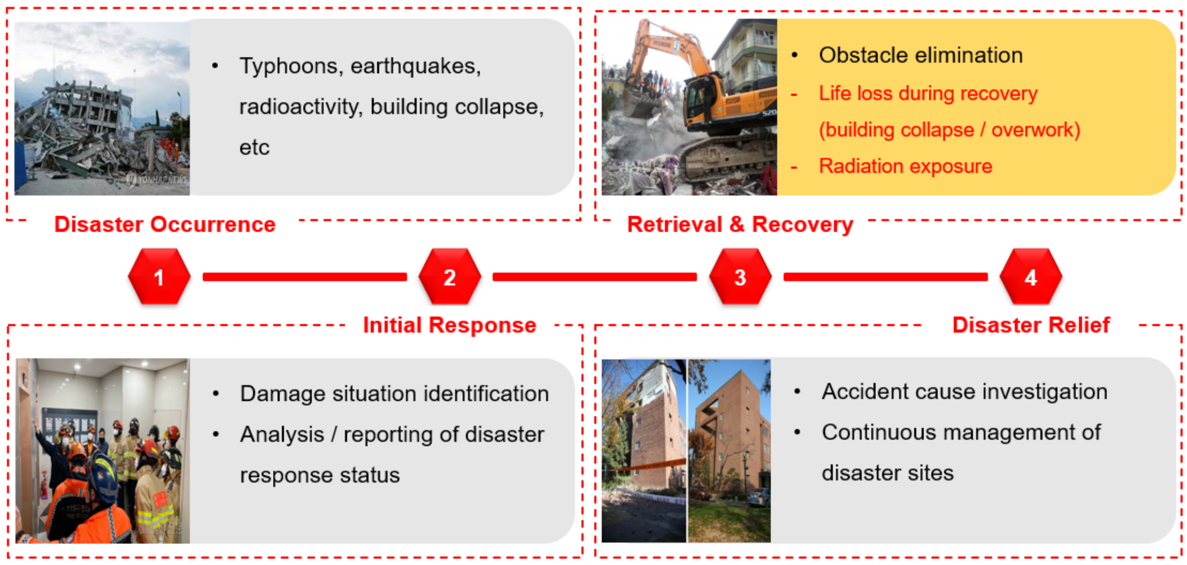
Robot System for Motion Tracking and Precise Parsing in Disaster Sites

This repository includes a **tutorial** on the **Robot System for Motor Tracking and Precise Parsing System in Disaster Sites**, the final project of the 2023-1 Industrial AI & Automation class.

1. Project Background


Typhoons, earthquakes, radioactivity, building collapse, etc
Disaster Occurrence
Obstacle elimination
- Life loss during recovery (building collapse / overwork)
- Radiation exposure
Retrieval & Recovery
1
2
3
4
Initial Response
Disaster Relief
Damage situation identification
Analysis / reporting of disaster response status
Accident cause investigation
Continuous management of disaster sites

In 4 steps of the disaster response process, the project focuses on the third step, **Retrieval & Recovery**. This stage includes the process of removing debris after a disaster, and during the process, human damage is caused due to building collapse, overwork, and radiation exposure.

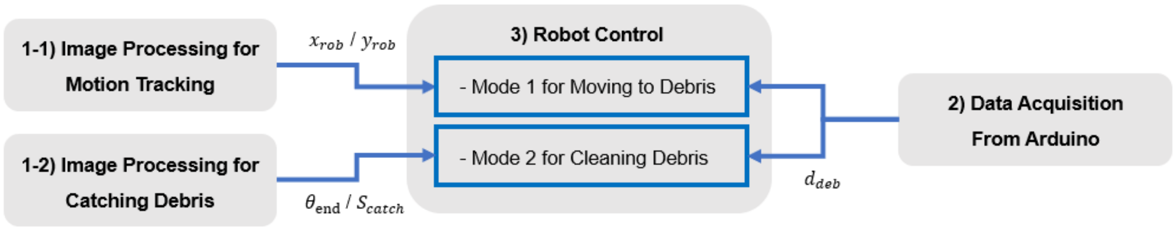
2. Project Goal

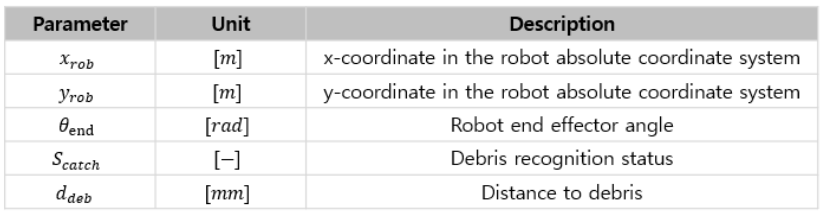
To solve the human casualty problem, this project aims to develop two algorithms.

- **Vision sensor- based motion tracking control**
  - Aim: Relative angle error within 10°
- **Vision sensor-based robot end-effector angle control**

3. Project Contents

3.1. Algorithm


1-1) Image Processing for Motion Tracking
1-2) Image Processing for Catching Debris
x_{rob} / y_{rob}
θ_{end} / S_{catch}
3) Robot Control
- Mode 1 for Moving to Debris
- Mode 2 for Cleaning Debris
d_{deb}
2) Data Acquisition From Arduino



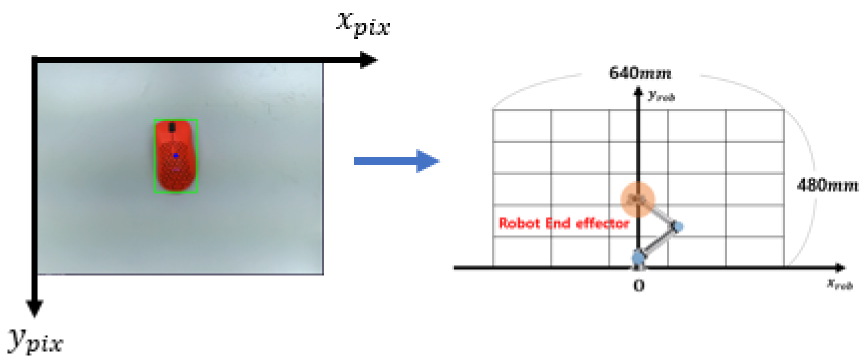
Parameter	Unit	Description
x_{rob}	$[m]$	x-coordinate in the robot absolute coordinate system
y_{rob}	$[m]$	y-coordinate in the robot absolute coordinate system
θ_{end}	$[rad]$	Robot end effector angle
S_{catch}	$[-]$	Debris recognition status
d_{deb}	$[mm]$	Distance to debris

First, the algorithm flow chart is shown in figure above. A brief description of each algorithm is given below.

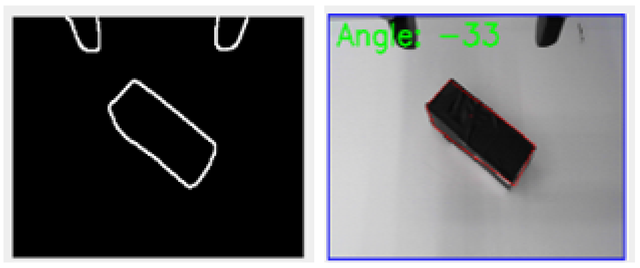
(1) Image Processing

Image processing is largely divided into two parts: **Motion tracking** and **Debris catching**.

- **Motion Tracking**
  - Color segmentation for object tracking
  - Pixel to robot absolute coordinate (See figure below)
  - Sending x_rob / y_rob to robot control part



- **Debris Catching**
  - Color segmentation for detecting debris
  - Get θ_image using geometry approximation algorithm
  - Coordinate conversion (θ*image to θ*end)
  - Sending θ_end to robot control part

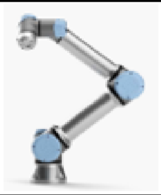
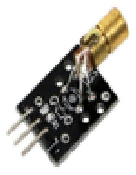
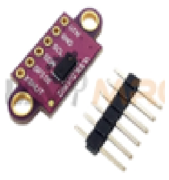
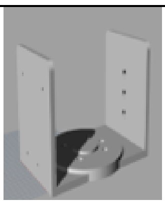


(2) Arduino / Robot Control

- **Arduino**
    - Connection distance sensor with Arduino
    - Serial communication between robot and Arduino
- **Robot Control (Mode 1)**
    - Receiving x_rob / y_rob / S_catch from image processing part
    - x_rob / y_rob input to robot / Change Mode 2
- **Robot Control (Mode 2)**
    - Receiving θ_end from image processing part
    - Receiving d_deb from Arduino
    - θ_end / d_deb input to robot

3.2. Hardware

Table 1. Hardware description

Hardware	Purpose
 Ur5e collaborative robot	It is used as a final actuator to clear debris in disaster sites.
 Laser pointer	It is used to know the robot's current position for precise debris gripping.
 Laser distance measurement sensor	It measures the distance to debris and limits the range of movement of the robot's end effector.
USB Camera (1)	It is used for observing the robot's movement in real time and recognizing the angle of the debris to control the angle of the robot's end effector.
USB Camera (2)	It is used for object recognition for motion tracking.
 Camera Adapter	It is used to connect the camera and the robot, and the drawing was directly drawn and output with a 3d printer.

4. Tutorial

4.1. Code Description & Installation

(1) Arduino Code for Data Acquisition

- **Bluetooth communication module download**
    1. Open Arduino IDE
    2. Tools -> Library management
    3. Install ***Adafruit_VL53L0X***

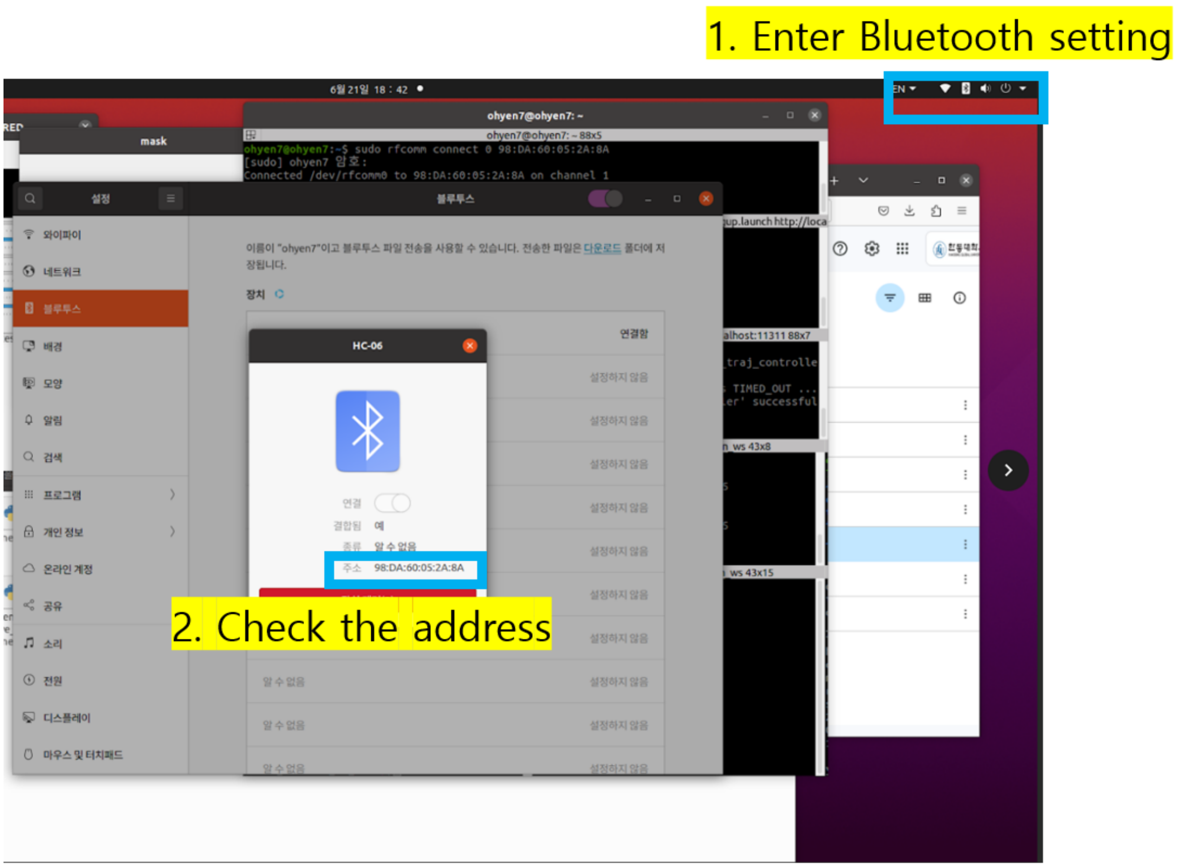
- **Code description**
  - Current bluetooth pins: 10(TX), 11(RX)
  - Communication speed: about 100Hz

```
SoftwareSerial BTSerial(10, 11); # Bluetooth communication pin number setting

...

void loop() {

    ...

    delay(10); # Communication speed can be adjusted by changing the delay value

}
```

- **Connect Arduino to computer and embed code through Arduino IDE**

(2) Bluetooth Module Connection

- **Check bluetooth device address**



```
ohyen7@ohyen7:~$ sudo rfcomm connect 0 98:DA:60:05:2A:8A
[sudo] ohyen7 암호:
Connected /dev/rfcomm0 to 98:DA:60:05:2A:8A on channel 1
```

이름이 "ohyen7"이고 블루투스 파일 전송을 사용할 수 있습니다. 전송한 파일은 다운로드 폴더에 저장됩니다.

결합됨 예
종류 알 수 없음
주소 98:DA:60:05:2A:8A

- **Bluetooth device connection code**

```
sudo rfcomm connect <number> <address>
# In my case, sudo rfcomm connect 0 98:DA:60:05:2A:8A
```

(3) Image_processing.py

It is a code that obtain coordinates of an object of a specific color through a camera in a tracking stage.

- **Code description**
  - Color segmentation

```
    def __init__(self):

...

        self.lower_bound = np.array([0, 192, 37])   # Initial color lower
bound
        self.upper_bound = np.array([66, 255, 193]) # Initial color upper
bound

        self.cap_ms = cv2.VideoCapture(4)
```

Lower bound and upper bound are the parts that define the initial HSV color boundary values, but if the specific color is not recognized properly, it can be adjusted through the track bar.

  - Image size & Track bar

```
def ImaegProcessing(self):

    self.cap_ms.set(cv2.CAP_PROP_FRAME_WIDTH, 320)
    self.cap_ms.set(cv2.CAP_PROP_FRAME_HEIGHT, 240)

    ...

    while True:

        h_min = cv2.getTrackbarPos("H_min", trackbarName)
        s_min = cv2.getTrackbarPos("S_min", trackbarName)
        v_min = cv2.getTrackbarPos("V_min", trackbarName)
        h_max = cv2.getTrackbarPos("H_max", trackbarName)
        s_max = cv2.getTrackbarPos("S_max", trackbarName)
        v_max = cv2.getTrackbarPos("V_max", trackbarName)

        self.lower_bound = np.array([h_min, s_min, v_min])
        self.upper_bound = np.array([h_max, s_max, v_max])

        ret, image = self.cap_ms.read()
        hsv_image = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)

        mask = cv2.inRange(hsv_image, self.lower_bound, self.upper_bound) #
extracts a specific color boundary
        result = cv2.bitwise_and(image, image, mask=mask)
        contours, _ = cv2.findContours(mask, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)
```

The size of the image was adjusted to 320 x 240 [pixels]. Too large image sizes causes large amount of computer processing, so we are set small image size. **Recommended not to change this size because it applies to pixel calculations of subsequent codes**

- Coordinate transformation

```
              for contour in contours:

                  if cv2.contourArea(contour) > 200: # Set minimum size of
detected object
                      x, y, w, h = cv2.boundingRect(contour)
                      cv2.rectangle(image, (x, y), (x + w, y + h), (0, 255, 0), 3)
# Draw rectangle as green.


                      self.flag_find = True

                      self.cx = ((x + int(w / 2)) - 160)/700 + 0.13   # 0.13:
Initial x point of robot end-effector

                      self.cy = (240 - (y + int(h / 2)))/500 + 0.49   # 0.49:
Initial 4 point of robot end-effector

                                                                    # (cx, cy):
pixel to robot absolute coordinate

                      self.radi = math.sqrt(self.cx ** 2 + self.cy ** 2)

                      if self.radi > 0.82:  # Set maximum range of robot move
                          self.cx = self.cx_prev
                          self.cy = self.cy_prev

                      self.cx_prev = self.cx
                      self.cy_prev = self.cy
                  else:
                      self.flag_find = False
                      self.cx = self.cx_prev
                      self.cy = self.cy_prev

              ...

              image = cv2.rotate(image, cv2.ROTATE_180) # Rotate image for user-
friendly view
```

(4) Image_processing_for_roboteye.py

It is a code that checks the angle of an object through a camera attached to the robot arm.

- **Code description**
  - Debris angle calculation

```
def calculate_aspect_ratio(self, box): # angle can be calculated only by knowing where the long side is.
    # Function to calculate the aspect ratio of the given cuboid
    edge1 = np.linalg.norm(box[0] - box[1])
    edge2 = np.linalg.norm(box[1] - box[2])
    edge3 = np.linalg.norm(box[2] - box[3])
    edge4 = np.linalg.norm(box[3] - box[0])

    max_length = max(edge1, edge2, edge3, edge4)
    min_length = min(edge1, edge2, edge3, edge4)

    aspect_ratio = max_length / min_length
```

```
 def calculate_angle(self, box):
        # Function to calculate the angle between the longest edge of the given cuboid
        edge_lengths = [np.linalg.norm(box[i] - box[(i + 1) % 4]) for i in range(4)]
        longest_edge_index = np.argmax(edge_lengths) #  Find the longest side

        # Find the coordinates of the two vertices of the longest side.
        vertex1 = box[longest_edge_index][0]
        vertex2 = box[(longest_edge_index + 1) % 4][0]

        angle = math.atan2(vertex2[1] - vertex1[1], vertex2[0] - vertex1[0]) * 180 / math.pi
        angle -= 90  # Subtract 90 degrees to make the y-axis the reference for 0 degree

        if angle < -90:
            angle += 180

        elif angle > 90:
            angle -= 180
```

(5) com_arduino_to_ros.py

It is a code that reads the sensor value of Arduino and send data to ROS.

- **Code description**
  - Communication speed

```
def __init__(self):
        self.COMMU_HZ = 20 # If you want to change communication speed, you
should change this value
        self.ser = serial.Serial("/dev/rfcomm0", 9600,
timeout=1/self.COMMU_HZ)
        self.ser.flushInput()
        self.bridge          = CvBridge()
        self.msg_object_info = object_info()
        self.arduino_pub     = rospy.Publisher("arduno_to_ros", object_info,
queue_size=10)
```

- Flush

```
def run(self):

    rospy.init_node('arduino_node', anonymous=True)  # 노드 이름
"camera_node"로 초기화
    rate = rospy.Rate(5)                            # 루프 실행 주기 : 30hz

    while not rospy.is_shutdown():                   # ROS가 종료되지 않은 동안

        if self.ser.inWaiting() > 0:

            recv_val = self.ser.readline().decode().rstrip()

            self.ser.flushInput()
            # If there is no flush function, the data can be accumulated in
communication packet.
            # Therefore, flush function can solve the communication delay
problem.
```

(6) demo_control_with_cameras.py

It is a code that controls the robot arm using the values given by the camera.

- **Code description**
  - Mode 1 robot control

```
def mode1_control(self):
    ...
    # Set minimum range to move robot
    if abs(self.cmd_x_ms - self.pre_cmd_x) > 0.005 or abs(self.cmd_y_ms
- self.pre_cmd_y) > 0.005:

        # Move robot end-effector with constant height
        target_pose_abs_xyz = [self.cmd_x_ms, self.cmd_y_ms,
self.cur_pose[2]]
```

  - Mode change flag

```
def mouse_click(self, event, x, y, flags, param):
        if event == cv2.EVENT_FLAG_L_BUTTON: # Mouse left button clicked
and debris is detected -> Mode change 1 to 2
            self.MODE = self.MODE2
            print("MODE CHANGED..!")
```

- End-effector rotation

```
def rotate_end_effector(self):
    # Rotate robot end-effector using detected debris angle value
    while(abs(self.cmd_angle_end_effector) > 0.1):
                ...
                if self.cmd_angle_end_effector < 0:
                    ...
                    cur_joint[5] = 1.13 * (cur_joint[5] +
self.cmd_angle_end_effector)
                else:
                    cur_joint[5] = cur_joint[5] +
self.cmd_angle_end_effector
```

This is code for reading the angle of an object in Mode 2 to align it with the gripper. If the difference in angle is 0.1[rad], the code is set so that the gripper goes down and picks up the object. Different numbers are set depending on the angle (+,-) of the object. If the angle is not right in the new environment, you can use it by changing it to the appropriate value through experiments.

- End-effector down

```
def go_down_to_barrier(self):
        ...
        target_pose[2] = cur_pose[2] - self.sensor_dist1 + 0.12 # 0.12
is obtained experimentally
```

This is a code that allows Mode 2 to go down by the distance value of the obstacle. If this also doesn't work well in the new environment, you can change it to the appropriate value and use it. (Unit: [m])

- Grip-off location

```
def go_down_garbage_dump(self):
        ...
        if self.i == 0:
            self.target_pose[0] = cur_pose[0] + 0.15
        elif self.i == 1:
            self.target_pose[0] = cur_pose[0] - 0.05
        elif self.i == 2:
            self.target_pose[0] = cur_pose[0] - 0.22
        ...
        # 물체를 내려 놓기 위한 z 위치
        self.target_pose[2] = cur_pose[2] - 0.27
```

This is a code for putting down obstacles. Currently, we are supposed to put the object down in the three positions we set. If necessary, you can change the x and z values and set the location where you want to drop them. (Unit: [m])

(7) Precautions

- There are two cameras (On robot and On tracking station), but the USB number is different depending on the computer, so you have to change the number if there is a camera index error.
- At the start of the program, set the initial position to x = 0.13, and y = 0.49.
- There is a limit to the length of the robot arm. If robot attitude is out of range, there will be error
- After Mode 2 is over, the tracking object should be set to x = 0.13, y = 0.49, before it moves to Mode 1.

4.2. Code Implementation Procedure

(1) Source Code Directory

```
catkin_ws/
|--- src/
    |--- ur_python/
        |--- src/
            |--- image_processing_230607.py
            |--- image_processing_for_roboteye_230531.py
            |--- com_arduino_to_ros.py
            |--- demo_control_with_cameras_230611.py
            |--- move_group_python_interface.py
```

(2) Implementation Codes and Procedures

1. Bluetooth device connection

```
sudo rfcomm connect 0 98:DA:60:05:2A:8A
```

2. UR5e robot software connection

```
roslaunch ur5e_rg2_moveit_config move_group.launch
```

3. Teaching pendant

```
roslaunch ur5e_rg2_moveit_config move_group.launch
```

4. Image processing for motion tracking

```
rosrun ur_python image_processing_230607.py
```

5. Image processing for debris angle detection

```
rosrun ur_python image_processing_for_roboteye_230531.py
```

6. Arduino-ROS communication

```
rosrun ur_python com_arduino_to_ros.py
```

7. Robot Control

```
rosrun ur_python demo_control_with_cameras_230611.py
```

5. Demonstration

Demo video

6. Trouble shooting

(1) Bluetooth error

Permission error

- The computer name must register under the following command. (user) is a computer name.

```
sudo usermod -a- -G dialout $(USER)
```

- reboot code

```
sudo reboot now
```

- Depending on USB recognition, the code should be different depending on where the port is connected.

```
sudo chmod a+rw /dev/rfcomm0
```

- When using Raspberry Pi's pinheader (/dev/tyAMA0) for serial communication

```
sudo usermod -a -G tty $USER
sudo chmod g+r/dev/ttyAMA0
```

tty checking method

- Check serial number

```
dmesg | grep tty
```

- Check serial port settings status

```
stty -F [devicename]
```

(2) USB port number error

- Check USB port code

```
lsusb
```

- You have to enter right port number for camera
    - In *image_processing_230607.py*

    ```
     self.cap_ms = cv2.VideoCapture(4)
    ```

    - In *image_processing_for_roboteye_230531.py*

    ```
    self.cap_re = cv2.VideoCapture(2)
    ```